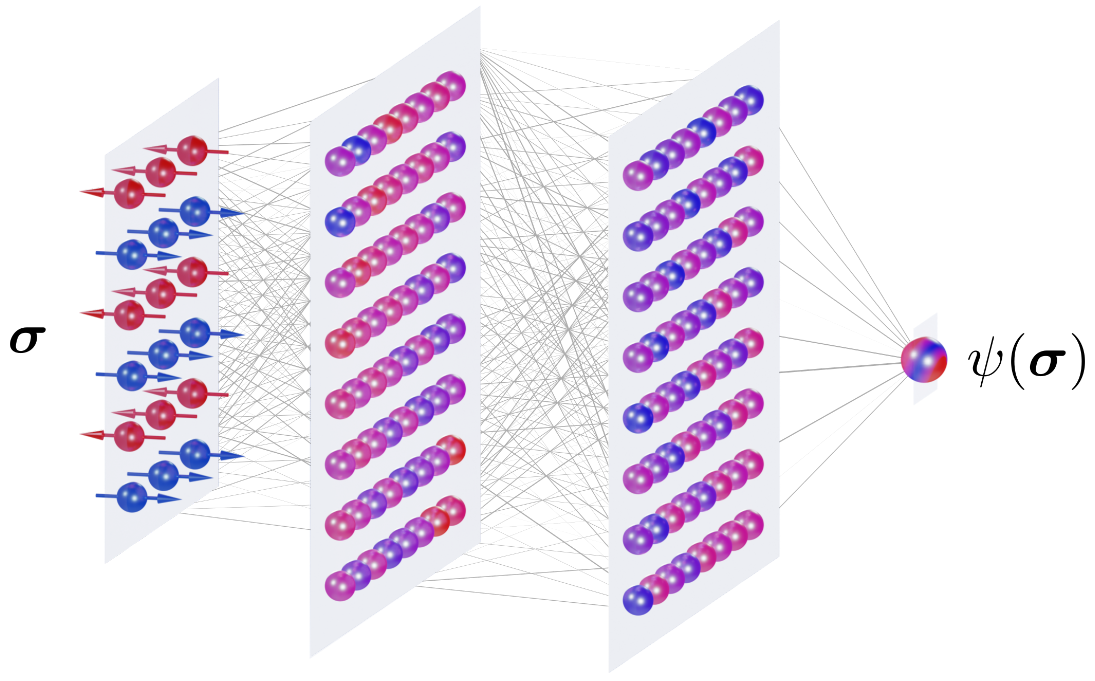
Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

mpsd
Max-Planck-Institut für
Struktur und Dynamik der Materie

Neural quantum state methods for simulating quantum many-body systems



Dissertation
zur Erlangung des Doktorgrades
an der Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Physik
der Universität Hamburg

vorgelegt von
Damian Hofmann

Hamburg
2023

Gutachter/innen der Dissertation:	Prof. Dr. Michael Sentef Prof. Dr. Nina Rohringer
Zusammensetzung der Prüfungskommission:	Prof. Dr. Daniela Pfannkuche Prof. Dr. Arwen Pearson Prof. Dr. Nina Rohringer Prof. Dr. Ángel Rubio Prof. Dr. Michael Sentef
Vorsitzende der Prüfungskommission:	Prof. Dr. Daniela Pfannkuche
Datum der Disputation:	31.05.2023
Vorsitzender des Fach-Promotionsausschusses PHYSIK:	Prof. Dr. Günter H. W. Sigl
Leiter des Fachbereichs PHYSIK:	Prof. Dr. Wolfgang J. Parak
Dekan der Fakultät MIN:	Prof. Dr.-Ing. Norbert Ritter

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Dissertation selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Hamburg, den 02.02.2023

Damian Hofmann

Preface

This is a cumulative dissertation on neural quantum state methods for simulating quantum many-body systems. It is based on work done during my doctoral studies between December 2018 and January 2023 at the Max Planck Institute for the Structure and Dynamics of Matter (MPSD) and the University of Hamburg under the supervision of Michael Sentef, Ángel Rubio, and Nina Rohringer. During my studies, I have been part of the International Max Planck Research School for Ultrafast Imaging and Structural Dynamics (IMPRS-UFAST).

Part of the work has been done during my stay as a Summer Research Associate at Flatiron Institute, a division of The Simons Foundation, in New York City from June to August 2019 under the supervision of Giuseppe Carleo.

The publications included in this thesis are listed in Chapter 1.

Abstract

Computational methods for the efficient simulation of quantum many-body systems are crucial for the study of condensed matter physics.

In this thesis, we investigate numerical properties of neural quantum states (NQS), a machine-learning-inspired variational ansatz based on using an artificial neural network to represent the quantum wave function. This representation can be used to stochastically estimate quantum expectation values, and the NQS ansatz can be trained to approximate ground states as well as real-time dynamics of quantum systems by classical optimization algorithms.

First, we investigate the stability of NQS time propagation with the time-dependent variational Monte Carlo method. Using the antiferromagnetic Heisenberg ladder as a benchmark system, we find that stochastic noise inherent to Monte Carlo sampling can be amplified by the variational equation of motion which can cause numerical instabilities. We propose an error diagnostic that can be used to quantify this effect and demonstrate the influence of regularization methods for the equation of motion on the stability of the dynamics. Subsequently, we discuss the importance of symmetries for improving NQS ground state calculations and propose a symmetry-projection scheme for the honeycomb Kitaev model. Furthermore, we present results of a systematic study of the capabilities of NQS based on feed-forward neural networks to represent highly entangled ground states in the Sachdev-Ye-Kitaev model. In this case, we find that this NQS ansatz does not learn a more efficient representation compared to the exponential scaling of the exact quantum states. This observation highlights the importance of further study to determine which properties decide whether a quantum state is amenable to an efficient approximation by neural quantum states. Finally, we present NetKet, an open-source project and software framework for numerical calculations in quantum many-body systems based on the NQS ansatz and variational Monte Carlo.

Altogether, our work highlights important challenges for the NQS approach and presents ways to help overcome those challenges and develop NQS into a reliable part of the toolbox for simulating quantum many-body physics.

Zusammenfassung

Rechenmethoden zur effizienten Simulation von Quantenvielteilchensystemen sind von essentieller Bedeutung für die Erforschung der Physik kondensierter Materie.

In dieser Arbeit werden numerische Eigenschaften von *Neural-Quantum-States* (NQS) untersucht, einem von Machine-Learning-Methoden inspirierten Variationsansatz, der auf der Darstellung der quantenmechanischen Wellenfunktion durch ein künstliches neuronales Netz basiert. Diese Darstellung kann verwendet werden, um mittels stochastischer Methoden quantenmechanische Erwartungswerte abzuschätzen. Der NQS-Ansatz kann mit klassischen Optimierungsverfahren trainiert werden, um Grundzustände sowie die Realzeitentwicklung von Quantensystemen zu approximieren.

Zunächst untersuchen wir die Stabilität von NQS-Zeitentwicklung unter Verwendung der zeitabhängigen Variational-Monte-Carlo-Methode. Wir zeigen mithilfe der antiferromagnetischen Heisenberg-Leiter als Benchmarksystem, dass stochastisches Rauschen, welches als natürliche Konsequenz des Monte-Carlo-Verfahrens auftritt, durch die variationelle Bewegungsgleichung verstärkt werden und dadurch numerische Instabilitäten verursachen kann. Wir stellen ein Diagnoseverfahren vor, um diesen Effekt zu quantifizieren und betrachten den Einfluss von Regularisierungsmethoden für die Bewegungsgleichung auf die Stabilität der Zeitentwicklung. Anschließend diskutieren wir die Bedeutung von Symmetrien für die Verbesserung von NQS-Grundzustandsrechnungen und schlagen ein Symmetrie-Projektionsverfahren für das Honeycomb-Kitaev-Modell vor. Weiterhin zeigen wir Ergebnisse einer systematischen Untersuchung der Fähigkeit von auf Feed-Forward-Neural-Networks basierenden NQS zur Darstellung stark verschränkter Grundzustände im Sachdev-Ye-Kitaev-Modell. In diesem Fall zeigt sich, dass dieser NQS-Ansatz im Vergleich zu den exponentiell skalierenden exakten Zuständen keine effizientere Darstellung lernt. Diese Beobachtung unterstreicht die Notwendigkeit weiterer Forschung im Hinblick auf die Frage, welche Eigenschaften eines Quantenzustands dafür entscheidend sind, ob er effizient durch NQS approximiert werden kann. Schlussendlich wird NetKet vorgestellt, ein Open-Source-Projekt und Software-Framework für numerische Rechnungen in Quantenvielteilchensystemen mit Variational-Monte-Carlo und dem NQS-Ansatz.

Insgesamt zeigt diese Arbeit wichtige Herausforderungen für die Anwendung von NQS-Methoden auf und stellt Wege vor, die dabei helfen können, diese zu überwinden und so NQS zu einem zuverlässigen Werkzeug zur Simulation der Quantenvielteilchenphysik zu entwickeln.

Acknowledgments

I am deeply grateful to my supervisor, Michael Sentef, who has advised and supported me throughout my work on this thesis. It has been a great pleasure to be part of Michael's research group, and I will miss this fantastic environment now that my time as a doctoral student is coming to an end.

I am also grateful to my co-supervisor, Ángel Rubio, for his steady support that has allowed me to pursue many exciting opportunities, to my second co-supervisor, Nina Rohringer, for motivating discussions at several stages of my doctoral studies, and to Giuseppe Carleo, who taught me about Monte Carlo methods and neural quantum states and hosted me during my visits to Flatiron Institute and EPFL.

Furthermore, I sincerely thank Giammarco Fabiani and Johan Mentink, who hosted me during my visit to Radboud University and who helped me a lot to better understand neural quantum state dynamics; my other collaborators, as well as the many people I had the opportunity to get to know as part of the NetKet and NQS community, related workshops, and other scientific events, including Atithi Acharya, Fabien Alet, Nikita Astrakhantsev, Denitsa Baykusheva, Sylvain Capponi, Kenny Choo, Martin Claassen, Stefanie Czischek, Jiahao Fan, Clemens Giuliani, Eliska Greplova, Lukas Grunwald, Mona Kalthoff, Dante Kennes, Marta Mauri, Christian Mendl, Matteo Mitrano, Pit Neitemeier, Evert van Nieuwenburg, Jannes Nys, Kris Pan, Giacomo Passetti, Gabriel Pescia, Christopher Roth, James E. T. Smith, Tomasz Sowiński, Attila Szabó, Giacomo Torlai, Agnes Valenti, Vladimir Vargas-Calderon, Filippo Vicentini, Tom Vieijra, Tom Westerhout, Alexander Wietek, Krzysztof Wohlfeld, Piotr Wrzosek, Dian Wu, and Yiqing Zhou.

Working at the MPSD and being part of the IMPRS-UFAST graduate school has been an exciting time and I thank all my colleagues, particularly Mona Kalthoff, Kevin Lively, and Lukas Windgätter, who have been great office mates; former and current members of the Sentef group, Brieuc Le Dé, Christian Eckhardt, Lukas Grunwald, José Pizarro, Gabriel Topp, Riku Tuovinen, Xiao Wang, and Tao Yu; and many other people, who are too numerous to list but have contributed to the amazing work environment at MPSD. I also wish to thank Neda Lotfiomran, Frauke Kleinwort, Ute Ramseger, and Kathja Schroeder for their great administrative support.

Finally, I thank my family and friends, who were there for me throughout my studies and whose support was crucial in helping me succeed.

Contents

1 Overview of publications

This cumulative dissertation is based on the four publications listed below. Declarations of contributions are provided on a separate cover page preceding each publication. Furthermore, this dissertation contains a section (Section 5.2) based on results that have not yet been independently published.

Included publications

Publication [P1] (included in Chapter 4) concerns stability properties of neural quantum state (NQS) time-propagation with the time-dependent variational Monte Carlo (t-VMC) method. Publication [P2] (included in Chapter 6) discusses numerical experiments on learning volume-law entangled ground states in the Sachdev-Ye-Kitaev (SYK) model using NQS. Finally, Publications [P3] and [P4] (included in Chapter 7) describe two major versions of the NetKet framework, which provides models and algorithms for NQS methods and to which I have contributed in the course of my studies. Note that the publications and results are included in a non-chronological order chosen for clarity of presentation.

[P1] **D.H.,** Giammarco Fabiani, Johan H. Mentink, Giuseppe Carleo, Michael A. Sentef.
Role of stochastic noise and generalization error in the time propagation of neural-network quantum states.
SciPost Physics 12, 165 (2022). DOI 10.21468/SciPostPhys.12.5.165.

[P2] Giacomo Passetti, **D.H.,** Pit Neitemeier, Lukas Grunwald, Michael A. Sentef, Dante M. Kennes.
Can neural quantum states learn volume-law ground states?
Preprint (2022). DOI 10.48550/arXiv.2212.02204.

[P3] Giuseppe Carleo, Kenny Choo, **D.H.,** James E. T. Smith, Tom Westerhout, Fabien Alet, Emily J. Davis, Stavros Efthymiou, Ivan Glasser, Sheng-Hsuan Lin, Marta Mauri, Guglielmo Mazzola, Christian B. Mendl, Evert van Nieuwenburg, Ossian O'Reilly, Hugo Théveniaut, Giacomo Torlai, Filippo Vicentini, Alexander Wietek.
NetKet: A machine learning toolkit for many-body quantum systems.
SoftwareX 10, 100311 (2019). DOI 10.1016/j.softx.2019.100311.

[P4] Filippo Vicentini, **D.H.,** Attila Szabó, Dian Wu, Christopher Roth, Clemens Giuliani, Gabriel Pescia, Jannes Nys, Vladimir Vargas-Calderón, Nikita Astrakhantsev, Giuseppe Carleo.
NetKet 3: Machine Learning Toolbox for Many-Body Quantum Systems.
SciPost Physics Codebases 7 (2022). DOI 10.21468/SciPostPhysCodeb.7.

Other publications

During my doctoral studies, I have contributed to the following additional publications which are not part of this cumulative dissertation.

[O1] Piotr Wrzosek, Krzysztof Wohlfeld, **D.H.,** Tomasz Sowiński, Michael A. Sentef.
Quantum walk versus classical wave: Distinguishing ground states of quantum magnets by spacetime dynamics.
Physical Review B 102, 024440 (2020). DOI 10.1103/PhysRevB.102.024440.

[O2] **D.H.,** Michael A. Sentef.
Resonant laser excitation and time-domain imaging of chiral topological polariton edge states.
Physical Review Research 2, 033386 (2020). DOI 10.1103/PhysRevResearch.2.033386.

[O3] Denitsa R. Baykusheva, Mona H. Kalthoff, **D.H.,** Martin Claassen, Dante M. Kennes, Michael A. Sentef, Matteo Mitrano.
Witnessing nonequilibrium entanglement dynamics in a strongly correlated fermionic chain.
Preprint (2022), accepted in *Physical Review Letters.* DOI 10.48550/arXiv.2209.02081.

2 Introduction

The quantum physics of condensed matter systems is fundamental to modern material science and technology. Quantum phenomena play a crucial role in the behavior of light-matter coupled systems [1, 2], strongly correlated electron systems [3], the formation of superconductivity [4], topologically protected states of matter [5], and quantum spin liquids [6, 7]. Understanding the physics of such systems can contribute to technological progress, especially in areas such as quantum computing [8–10], engineering of electronic and photonic devices [11], and materials design [12, 13].

The quantum many-body problem

Condensed matter systems are described by quantum many-body theory, the foundations of which have been developed over the past century. While the axioms and equations governing quantum many-body systems are well established, the emergent complexity of quantum many-body systems [14] still poses a formidable challenge for solving real-world problems, be it through numerical simulation or analytical means.

To model fundamental quantum effects, such as superposition and entanglement, the state of a quantum system needs to be described by a quantum wave function, which is an object of exponential dimensionality in the size of the physical system. This scaling limits the exact classical simulation of arbitrary quantum systems to only a few particles or lattices of tens of sites at most. The challenge presented by the vast complexity of the quantum state space has been known and discussed since the early days of quantum physics, as illustrated by the following widely-known quote by Dirac:

> "The underlying physical laws necessary for the mathematical theory of a large part of physics and the whole of chemistry are thus completely known, and the difficulty is only that the exact application of these laws leads to equations much too complicated to be soluble. It therefore becomes desirable that approximate practical methods of applying quantum mechanics should be developed, which can lead to an explanation of the main features of complex atomic systems without too much computation."
> – P. A. M. Dirac, *Proc. Royal Soc. A* 123 (1929) [15].

Simulating quantum systems in a scalable manner was also the primary motivation behind Richard Feynman's proposal to simulate quantum mechanics using quantum-mechanical systems, which sparked research into quantum computing [16]. While practically useful universal quantum computers are yet to be realized [8, 10, 17], quantum simulation using more specialized experimental setups is an integral part of current physics research [18, 19]. Implementing and effectively utilizing such quantum simulators still requires a solid understanding of the underlying physics and, thus, efficient simulation methods realizable on classical computers.

Many approaches have been used over the past century to study the physics of quantum many-body systems while avoiding the exponential barrier imposed by the dimensionality of the quantum state space. Ideally, analytical expressions for quantities of interest can be derived, which can be evaluated by hand or using comparatively modest computational resources. Examples of analytically solvable systems include the quantum transverse-field Ising model in one dimension [20], the Bethe ansatz for some interacting one-dimensional systems [21–23], or the Kitaev model on the honeycomb lattice, which can be transformed into a system of free fermions under certain constraints [24]. However, such general analytical results are scarce and often limited to equilibrium settings.

Many computational methods focus on *reduced quantities* [25] such as the single-particle density, which is a fundamental object in density functional theory [26] and time-dependent density functional theory [27, 28], or n-body correlators, which are central to nonequilibrium Green's function theory [29, 30]. Approaches such as these make it possible to circumvent the complexity of the many-body quantum state while still giving access to key physical properties of the systems under consideration. However, some types of quantum systems, particularly strongly correlated systems out of equilibrium, still pose a significant challenge for these techniques [3].

In the variational approach, the concept of the full quantum state is retained by parametrizing a subset of the full Hilbert space, the variational manifold, using a computationally tractable number of parameters [31, 32]. Whether the variational manifold contains the states required to describe the physics of a specific system is a central question for the applicability of a particular variational method. For some classes of variational states, there is a solid understanding of the types of systems that are well captured. For instance, matrix-product states (MPS) can efficiently represent the weakly entangled ground states of local, gapped Hamiltonians, particularly in one dimension [33–35], while Gaussian states can capture the physics of non-interacting bosonic and fermionic Hamiltonians [36–38]. However, some physical systems are currently beyond the capabilities of established variational techniques. Particular challenges again arise in strongly correlated quantum systems [2]. Therefore, finding suitable classes of ansätze for such systems is an important goal of current research.

Machine learning and neural quantum states

Over the past years, machine learning methods have started to play an increasingly important role in the physical sciences [39–45]. This trend has been inspired by the success of such approaches, particularly deep learning [46–48], in solving problems in a diverse set of fields such as image classification [49, 50], natural language processing [51], protein folding [52], numerical linear algebra [53], and even playing complex games [54–56]. In computational quantum physics, machine-learning-based methods have been explored for a variety of purposes, including classifying phases and identifying phase transitions [57–62], optimizing driving protocols [63] and quantum circuits [64], Hamiltonian learning [65–67], and experiment design [68, 69].

Artificial neural networks are mathematical models with powerful approximation capabilities and represent important building blocks of many machine learning algorithms. The method of neural quantum state (NQS), introduced in a seminal paper by Carleo and Troyer [70] in 2017, is based on the idea of using a neural network as a variational parametrization of a generic many-body quantum state. This ansatz can be evaluated and optimized stochastically through variational Monte Carlo (VMC) [71, 72]. NQS methods can be applied to learning ground and excited states as well as the dynamics of quantum systems [42], and has been used for spin [70, 73–90], bosonic [91–95], and fermionic [96–100] models, often achieving ground state energies improving on or competitive with the state-of-the-art [73, 81, 86, 88, 100]. Theoretical results that demonstrate the capabilities of NQS to represent quantum states of interest, such as random volume-law entangled states [101, 102] and chiral spin liquids [103–105], have increased the interest in this ansatz due to its potential to fill an important gap in the landscape of computational methods. A particular focus has been the investigation of frustrated spin systems [75, 81, 86, 88, 89] and nonequilibrium dynamics [76, 78, 79, 87, 106].

The availability of open-source software projects that make state-of-the-art methods available to researchers and other interested parties with a low barrier of entry has been an important factor in the evolution of machine learning methods over the past decade [107, 108]. This has also benefited the application of machine learning methods in physics, and the NQS community itself has given rise to several open-source frameworks [109, 110, P3, P4].

There are important open questions in NQS research. Since NQS algorithms are based on VMC, stochastic noise can affect estimates and cause numerical instabilities; this has been encountered especially when simulating real-time dynamics [78, 111–113, P1], though stochastic effects also play a role in ground state optimization [94, 114, 115]. Another important frontier is the question of representability: Investigation into the representation capabilities of specific network architectures has shown that NQS can efficiently represent a wide variety of quantum states [101–105, 116–120]. However, it is yet to be understood which specific features of a quantum state allow it to be efficiently learned by an NQS ansatz and how this could be quantified [106, P2].

After providing a review of relevant background information on the variational approach, neural quantum states, and variational Monte Carlo (Chapter 3), this thesis will address several aspects of these questions and developments. Specifically, this is done by investigating the numerical stability of NQS time propagation (Chapter 4), the utilization of symmetries to improve the accuracy of VMC calculations, with a particular application to a quantum spin liquid system (Chapter 5), and the learnability of highly entangled ground states in the Sachdev-Ye-Kitaev (SYK) model (Chapter 6). Furthermore, we will discuss the development of the open-source framework NetKet for NQS simulations (Chapter 7). Finally, Chapter 8 will conclude this thesis with a discussion and outlook.

3 Variational and neural quantum states

This chapter reviews the fundamental concepts of variational quantum states, neural quantum states, and their numerical treatment by variational Monte Carlo (VMC) methods, which form the basis of the work presented in subsequent chapters of this thesis.

Recently, several reviews have been published that cover neural quantum state (NQS) and further machine learning applications beyond the overview provided here. In particular, the reader is referred to Ref. [42, Chapter 3] for a set of lecture notes containing additional details and examples of neural quantum states as well as Refs. [41–44] for broader coverage of the many applications of machine learning methods in the physical sciences.

3.1 Variational quantum states

Condensed matter systems can generally be simulated using quantum-mechanical lattice models. The Hilbert space of such a model can be expressed as a tensor product $\mathcal{H} = \bigotimes_{i\in\Lambda} \mathcal{H}_i$ of local Hilbert spaces $\mathcal{H}_i$ over a lattice Λ. The lattice is a (finite or infinite) collection of sites, which can be represented by positions in a d dimensional space, $\Lambda \subseteq \mathbb{R}^d$, or an arbitrary set of vertices in a graph (allowing for arbitrary labels of the lattice sites). The local Hilbert spaces have the form

$$\mathcal{H}_i = \operatorname{span}\{|s\rangle \mid s \in \mathcal{L}_i\}, \tag{3.1}$$

where $\mathcal{L}_i$ is an arbitrary (and potentially infinite) set of size $|\mathcal{L}_i| = \dim \mathcal{H}_i$ labeling the states of a local orthonormal basis, $\langle s|s'\rangle = \delta_{s,s'}$. A basis state of the full Hilbert space is then identified by a *configuration* $\boldsymbol{s} \in \mathcal{S} = \prod_{i\in\Lambda} \mathcal{L}_i$. These configurations correspond to the tensor-product basis states

$$|\boldsymbol{s}\rangle = \bigotimes_{i\in\Lambda} |s_i\rangle \in \mathcal{H} \tag{3.2}$$

which, too, are orthonormal. In the simplest case, the local Hilbert space can have the form of a two-level system (a *qubit*), where $\mathcal{H}_i \cong \mathbb{C}^2$.[1] More generally, the local state space can be a d_i-level system (a *qudit*) $\mathcal{H}_i \cong \mathbb{C}^{d_i}$ or, in the case of a bosonic system, the countably infinite space of square-summable sequences $\mathcal{H}_i \cong \ell^2(\mathbb{C})$. The labels of the basis are, in principle, arbitrary, e.g., a qubit system can be described by basis states $\{|0\rangle, |1\rangle\}$, or as an equivalent spin-1/2 system with $\hat{\sigma}^z$ basis states $\{|\pm 1\rangle\}$.

[1]The actual quantum state space is $\mathcal{H}_i$ modulo the gauge equivalence $|\psi\rangle \equiv \gamma|\psi\rangle$ for all $|\psi\rangle \in \mathcal{H}_i, \gamma \in \mathbb{C}$, which is a *projective Hilbert space* [121]. We will not explicitly make this distinction in the following and work with the plain vector space $\mathcal{H}_i$, keeping in mind the gauge equivalence where it is relevant. Note that we will not assume (variational) pure states to be normalized, instead using the gauge-invariant definitions of expectation values and other observable quantities.

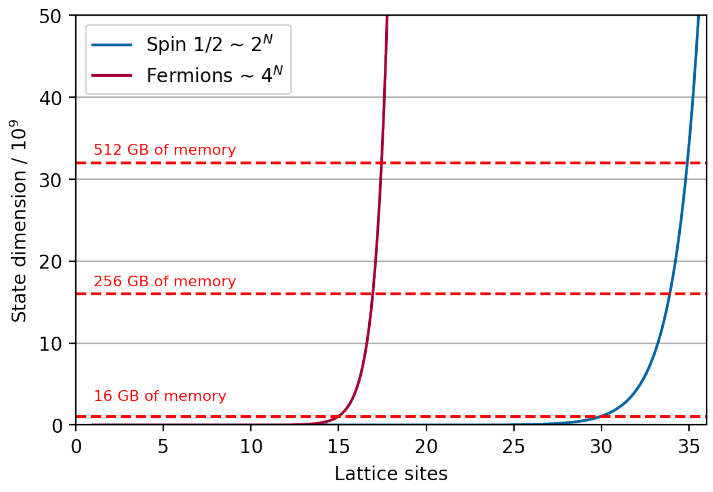


Figure 3.1 | Exponential scaling of memory requirements for storing a single quantum state in the full state vector representation of a lattice system with spin-1/2 (blue) or two-orbital fermion (red) degrees of freedom at each site. The memory requirements assume the state is stored as a vector of complex double-precision floating point numbers, each taking up 128 bits.

In the remainder of this thesis, we will focus on finite Hilbert spaces where both $|\Lambda| = N \in \mathbb{N}_{>0}$ and $|\dim \mathcal{H}_i|$ are finite and the many-body Hilbert space is thus given by

$$\mathcal{H} = \operatorname{span}\{|\boldsymbol{s}\rangle \mid \boldsymbol{s} \in \mathcal{S} = \mathcal{L}_1 \times \cdots \times \mathcal{L}_N\}. \tag{3.3}$$

This definition encompasses all spin and qudit systems, fermionic systems, and bosonic systems with bounded local occupation numbers. The Hilbert space dimension of a lattice system is the product of its local dimensions, $\dim \mathcal{H} = \prod_{i\in\Lambda} \dim \mathcal{H}_i$. In a uniform system of local dimension d, the Hilbert space dimension thus scales exponentially as $\dim \mathcal{H} = d^N$. To illustrate this scaling, Fig. 3.1 shows the memory required to store a single state vector $|\psi\rangle \in \mathcal{H}$ naively as a vector of double-precision floats (requiring 128 bits for every complex number) for qubit ($d = 2$) and spinful fermion ($d = 4$) local bases. Memory size is not the only limiting factor. Algorithms based on exact state vector simulation need to process the data contained in $|\psi\rangle$ to perform tasks such as calculating observables or evolving the state. In the generic case, this requires examining all amplitudes of a given state at least once, giving an exponential lower bound for the runtime complexity of these operations. Due to this scaling, only small quantum systems (up to a limit of approximately 20-30 qubits depending on the specific system and task) can be simulated using non-specialized hardware and exact algorithms based on the full quantum state.

The scaling of state vector simulations can be improved for specific tasks, particularly finding ground states by exact diagonalization (ED), through exploiting symmetries, which can extend the regime of systems accessible by ED methods up to about 50 sites [122]. Similar system sizes can theoretically be simulated for general quantum states in specialized settings using a significant amount of high-performance computing (HPC) resources. However, the associated cost makes such an undertaking almost impossible in practice [123–125].

An arbitrary pure quantum state in the Hilbert space (3.3) is represented by a vector

$$|\psi\rangle = \sum_{\boldsymbol{s}\in\mathcal{S}} \psi(\boldsymbol{s})|\boldsymbol{s}\rangle. \tag{3.4}$$

Thus, knowing the mapping $\psi\colon \boldsymbol{s} \mapsto \psi(\boldsymbol{s}) = \langle \boldsymbol{s}|\psi\rangle$, the *quantum wave function*, fully specifies the state.

The fundamental assumption underlying any variational approach is that for physical states of interest, $\psi(\boldsymbol{s})$ depends on $\boldsymbol{s}$ in a structured manner and, thus, can be represented using a significantly smaller amount of information compared to independently memorizing all coefficients [126]. This is known to be true, e.g., for ground states of gapped, local lattice models in one dimension, which can always be expressed as an matrix-product state (MPS) of subexponential size, and this representation can often be efficiently computed using the density matrix renormalization group (DMRG) algorithm [33–35, 127]. Ground states of more general physically motivated Hamiltonians are still significantly different from random Hilbert space elements [128]. However, the specific structure of the relevant states is not necessarily known.

Given the structured nature of physically relevant states, it makes sense to express the quantum state through a parametrized function (an *ansatz*) $(\boldsymbol{s}, \boldsymbol{\theta}) \mapsto \psi_{\boldsymbol{\theta}}(\boldsymbol{s})$ that depends on a set of variational parameters[2] $\boldsymbol{\theta} \in \mathcal{X} \subseteq \mathbb{K}^M$. For our purposes, the parameters space can be a real or complex vector space, i.e., the underlying field is $\mathbb{K} \in \{\mathbb{R}, \mathbb{C}\}$. Since $\mathbb{C}^M \simeq \mathbb{R}^{2M}$ are isomorphic as vector spaces, this distinction is mainly for notational convenience. A generic *variational state* (or *trial state*) is given by

$$|\psi_{\boldsymbol{\theta}}\rangle = \sum_{\boldsymbol{s} \in \mathcal{S}} \psi_{\boldsymbol{\theta}}(\boldsymbol{s}) |\boldsymbol{s}\rangle. \tag{3.5}$$

The set of states that can be expressed in a given parametrization is the *variational manifold*

$$\mathcal{M} = \{ |\psi_{\boldsymbol{\theta}}\rangle \mid \boldsymbol{\theta} \in \mathcal{X} \}. \tag{3.6}$$

We refer to the function $\boldsymbol{\theta} \mapsto |\psi_{\boldsymbol{\theta}}\rangle$ as the *variational mapping*. Under suitable mathematical assumptions, in particular regarding the differentiability of the variational mapping, $\mathcal{M}$ does indeed carry the mathematical structure of a submanifold of $\mathcal{H}$ [129].

In general, there are several properties that families of variational states need to possess to be useful for computational purposes:

1. *Compression.* A variational ansatz needs to be able to parametrize a region of interest within the full Hilbert space at a sufficiently small size to store and process. In particular, the required parameter-space dimension should not scale exponentially with the system size.

2. *Efficient evaluation of observables.* The quantum wave function fully defines the state of a system. However, to understand a system's physical properties and obtain data that can be compared to experiments or predictions from other simulation methods, it is important to compute observable quantities of interest. Therefore, even an efficiently compressed representation of the quantum state is only useful if it also supports the efficient evaluation of relevant classes of observables.

3. *Efficient optimization.* Physical applications rely on optimizing the variational parameters to reach a target state with desired properties. Typically, optimization targets are the minimization of the energy expectation value $\langle \hat{H} \rangle_{|\psi_{\boldsymbol{\theta}}\rangle}$ in ground state search (Section 3.3), optimizing the overlap with an analytically or otherwise known state (Section 3.2.2), or following an equation of motion (Section 4.1). Therefore, a useful variational ansatz should also provide efficient means to perform such optimizations, either by specialized algorithms (like DMRG) or, more generally, by providing an algorithm to compute gradients of the ansatz wave function that, in turn, can be used to obtain gradients of objective functions.

[2] We will also refer to the number of variational parameters as the *size* of the ansatz, or *network size* for NQS, in the following.

As formulated above, the definition of a variational state encompasses many different families of states which vary widely in their physical characteristics, computational complexity, parameter space dimension, and flexibility. Linear combinations of a subset of basis functions can be seen as a particularly simple form of variational ansatz, where the variational parameters are the coefficients, and the variational manifold is a linear subspace. At minimum, a variational state can have only one parameter (e.g., a single-mode coherent state). In contrast, the size of many-variable variational ansätze increases with the system size; examples include MPS and tensor-network states (TNS) [35, 127], Gaussian states and their generalizations [130, 131], or fermionic pair-product states [132, 133]. For practical applicability, the scaling of the required size needs to be tractable and, in particular, subexponential.

An important class of ansätze are those that are *systematically improvable*, which means that the variational manifold (and thus the representation capabilities) of the ansatz can be expanded by increasing the number of parameters. Ideally, the ansatz should be able to represent any quantum state in the limit of infinite size. This is the case for MPS, which are based on decomposing the full state (interpreted as a rank-N tensor) into a product of lower-dimensional matrices [134]

$$\psi_{\boldsymbol{A}}(\boldsymbol{s}) = \mathbf{A}_1^{(s_1)} \cdots \mathbf{A}_N^{(s_N)}, \qquad \mathbf{A}_i^{(s_i)} \in \mathbb{C}^{\chi_i \times \chi_{i+1}}, \qquad \chi_1 = \chi_{N+1} = 1. \tag{3.7}$$

The size of the MPS ansatz is controlled by the bond dimensions[3] χ_i, $i \in [N]$. This ansatz can represent any quantum state once the bond dimensions become sufficiently large. While the exact representation of arbitrary states requires exponentially growing bond dimensions, sufficiently structured states can be represented efficiently [33]. The suitability of a systematically improvable ansatz for a given problem can be verified by assessing convergence as a function of the bond dimensions.

3.2 Neural quantum states

Neural quantum states (NQS)[4] are variational states that use an artificial neural network to represent the quantum wave function. Section 3.2.1 will review the concept of artificial neural networks, Section 3.2.2 will define NQS, and Section 3.2.3 will give an overview of neural network architectures used in the context of NQS.

3.2.1 Artificial neural networks

Artificial neural networks are a rich class of mathematical functions that can approximate complicated functional relationships. Initially inspired by the behavior of biological neurons in the human brain [135–137], neural networks have developed from these roots into a graphical language for describing classes of nonlinear approximating functions [46–48, 138], which do not necessarily resemble specific biological processes [139].

[3]Here and in the following, we write $[N] = \{n \in \mathbb{N} \mid 1 \leq n \leq N\}$ to denote the set of integers between one and $N \in \mathbb{N}$ (inclusive).

[4]Note that the terms "neural-network quantum state" and "neural quantum state" are both used in the literature and refer to the same concept. While the first term is the one originally used by Carleo and Troyer [70] and also in two of the included Publications [P1, P3], parts of the literature, including the Publications [P2, P4] as well as the main text of this dissertation, have adopted the more concise second term.

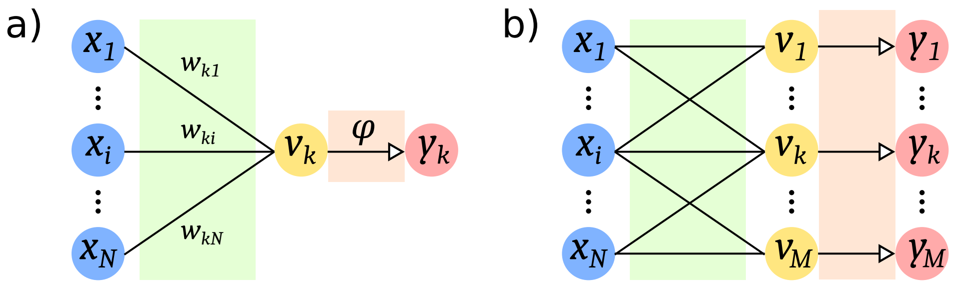


Figure 3.2 | a) Graphical representation of a single artificial neuron (without bias) with index k. The neuron performs a weighted sum over the input signals $\boldsymbol{x} \in \mathbb{R}^N$ and passes the resulting signal $v_k \in \mathbb{R}$ through a nonlinear activation function φ, generating a scalar output signal $y_k \in \mathbb{R}$.
b) Graphical representation of a single fully-connected neural network layer (without bias) with M artificial neurons. The depicted layer performs a linear transformation [with weight matrix $W = (w_{ki})$ of shape $M \times N$] on the input data $\boldsymbol{x} \in \mathbb{R}^N$ and passes the resulting vector of signals $\boldsymbol{v} = W\boldsymbol{x} \in \mathbb{R}^M$ through a nonlinear activation function to generate the output vector $\boldsymbol{y} = \Phi(\boldsymbol{v}) \in \mathbb{R}^M$.

Key theoretical results underlying the power of neural networks are so-called universal approximation theorems, which show that broad classes of networks are dense in the space of continuous functions [140–143]. Such neural networks can therefore approximate any continuous function to arbitrary precision in the limit of infinite network size.

The paradigmatic building block of larger neural networks is the artificial neuron (Fig. 3.2a) [138]. A single neuron accepts a set of input signals, modeled as a vector $\boldsymbol{x} \in \mathbb{R}^N$, and performs the transformation

$$y_k = \varphi(v_k), \qquad v_k = \sum_j w_{kj} x_j + b_k \tag{3.8}$$

where we denote by k the index of the neuron. The action of a single neuron on the input signal thus is comprised of three steps: First, a weighted linear combination of the input signals with *weights* w_{kj} is formed and then offset by the *bias* $b_k \in \mathbb{R}$. Finally, a nonlinear *activation function* φ is applied, yielding an output signal $y_k \in \mathbb{R}$. For some standard activation functions such as the rectified linear unit (ReLU) [144] $\varphi(v) = \max(0, v)$ or the sigmoid function $\varphi(v) = \sigma(v) := (1 + e^{-v})^{-1}$, this process can indeed be seen as a (rough) approximation of a biological neuron, which collects electrical input signals that are forwarded and amplified only when they exceed a certain potential threshold. In this picture, the artificial neuron's output can be roughly interpreted as the firing rate of a biological neuron [139]. This analogy does not necessarily extend to other common activation functions, such as hyperbolic functions like $\tanh$ or $\cosh$.

More recently suggested activation functions are designed to overcome various issues in network optimization, especially the so-called vanishing gradient problem in deep learning [145]. Examples are the Swish activation $\varphi(v) = v\,\sigma(v)$ [146], the penalized tangent [147, 148], the exponential linear unit (ELU) [149], or the scaled exponential linear unit (SELU) [145].

While the choice of activation function impacts performance and stability of training, results vary between different applications [146, 148] and, in principle, vast classes of choices for the nonlinear activation function retain the universal approximation property of neural networks. Common theoretical requirements on activations functions are that they are continuous, non-polynomial, and differentiable, at least on some subset of their domain. However, even parts of these very general requirements can be relaxed depending on the specific proof [140–143]. For practical applications, it is favorable if an

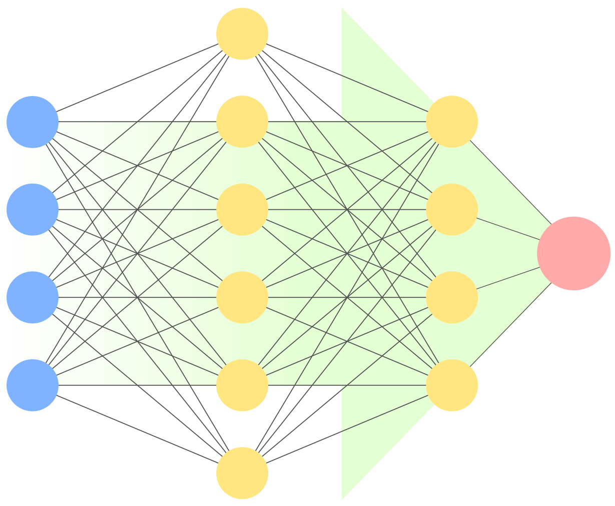


Figure 3.3 | Graphical representation of a feed-forward neural network with an input layer of size 4, two hidden layers with 6 and 4 neurons, respectively, and scalar output. Each layer applies an affine transformation followed by a nonlinear activation function, as depicted in Fig. 3.2b.

activation function is smooth (or at least continuously differentiable) almost everywhere in its domain to allow for stable computation of gradients.

Typically, neural networks are organized in layers of artificial neurons (Fig. 3.2b). A layer consists of M artificial neurons, each receiving the same set of inputs (but with separate weights and biases). Then, the layer's output is given by

$$\boldsymbol{y} = \Phi(\boldsymbol{v}), \qquad \boldsymbol{v} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b}, \tag{3.9}$$

which is an affine transformation with weight matrix $\mathbf{W} \in \mathbb{R}^{M \times N}$ and bias vector $\boldsymbol{b} \in \mathbb{R}^{M}$ followed by a nonlinear activation $\Phi\colon \mathbb{R}^M \to \mathbb{R}^M$. This activation function usually consists of a component-wise application of a scalar activation function, $\Phi(\boldsymbol{v})_i = \varphi(v_i)$. However, it can also include other operations, such as normalization. The layer given by Eq. (3.9) is called a *fully-connected layer* or *linear layer*. Sequentially applying several fully-connected layers defines a so-called feed-forward neural network (FFNN) or *multi-layer perceptron* (Fig. 3.3) [48]. A layer's output dimension is often called its *width*, while a network's number of layers is called its *depth*. The term *deep learning* thus refers to using networks with many layers, though already two-layer networks can exhibit characteristics of deep networks [150].

While real-valued signals are the most common choice in standard machine learning applications, complex-valued signals can also be used. These are particularly common in neural quantum states [42] but also find applications in areas such as signal processing [151]. In this case, the definition of artificial neurons and network layers is unchanged. However, the activation functions need to be well-defined for complex inputs. This is the case for hyperbolic functions such as $\tanh$ and $\cosh$. Furthermore, any real activation function φ_R can be applied to a complex signal by separating real and imaginary parts, e.g., as $\varphi(z) = \varphi_\mathrm{R}(\operatorname{Re} z) + i\varphi_\mathrm{R}(\operatorname{Im} z)$.

Beyond the fully-connected layer shown here, many more types of network layers exist. Several other architectures that have found applications in NQS simulations will be shown in Section 3.2.3. The reader is referred to the literature [41, 42, 48, 138] for a broader overview of models and network architectures.

3.2.2 Learning quantum states

The key idea of the NQS ansatz is to use a neural network to parametrize the variational mapping, giving a trial state of the form

$$|\psi_{\boldsymbol{\theta}}\rangle = \sum_{\boldsymbol{s}\in\mathcal{S}} \mathrm{NN}(\boldsymbol{s};\boldsymbol{\theta})|\boldsymbol{s}\rangle \tag{3.10}$$

or

$$|\psi_{\boldsymbol{\theta}}\rangle = \sum_{\boldsymbol{s}\in\mathcal{S}} e^{\mathrm{NN}(\boldsymbol{s};\boldsymbol{\theta})}|\boldsymbol{s}\rangle. \tag{3.11}$$

As a consequence of the universal approximation property of neural networks, NQS are a systematically improvable variational ansatz.

The neural network output is often taken as the log-probability amplitude $\ln\psi_{\boldsymbol{\theta}}(\boldsymbol{s}) = \mathrm{NN}(\boldsymbol{s};\boldsymbol{\theta})$ as in Eq. (3.11). This choice has the benefit that the network can more easily learn to represent probability amplitudes spanning several orders of magnitude. Furthermore, neural network libraries provide easy access to computing derivatives of the network weights with respect to network parameters via automatic differentiation functionality (Section 7.2). Direct access to log-amplitude derivatives can simplify the implementation of VMC algorithms which rely on these quantities in the computation of gradients (Sections 3.3 and 4.2). Therefore, parametrizing the log-probability amplitudes is the implementation choice used in NetKet [P3, P4].

Optimization tasks are defined by a target function that is to be minimized. In machine learning, this objective function is usually called the *loss function* [138], which depends on the neural network model. Training the network is then accomplished by finding optimal network parameters that minimize the selected loss function.

The prototypical machine learning task is that of *supervised learning*. Here, datasets consisting of pairs of the form $(x_i, y_i) \in D \subseteq \mathbb{X}\times\mathbb{Y}$ are given, where $\mathbb{X}$ is the space of inputs and $\mathbb{Y}$ the space of corresponding outputs. It is assumed that the input and output data are related by an unknown mapping $y\colon \mathbb{X}\to\mathbb{Y}, x\mapsto y(x)$, which the network should learn to approximate. In this setting, the loss function needs to quantify the difference between the outputs $\tilde{y}_i(\boldsymbol{\theta}) = \mathrm{NN}(x_i;\boldsymbol{\theta})$ predicted by the network and the corresponding true output y_i given as part of the training dataset D. The simplest example is the mean-squared loss [48]

$$L(\boldsymbol{\theta}; D) = \frac{1}{|D|}\sum_{i=1}^{|D|} \|\tilde{y}_i(\boldsymbol{\theta}) - y_i\|^2. \tag{3.12}$$

In the context of NQS, the input space is then the space of configurations $\mathbb{X} = \mathcal{S}$, and the network outputs are (log-)probability amplitudes, $\mathbb{Y} = \mathbb{C}$. Given access to training samples $D = \{(\boldsymbol{s}_i, \ln\psi(\boldsymbol{s}_i))\}_{i=1}^{|D|}$, the mean-squared loss (3.12) does train the network to reproduce the desired quantum wave function. However, since this loss does not take into account the gauge degrees of freedom, it can be beneficial to use instead a loss function based on the overlap with the target state, such as the negative log-overlap loss

$$L(\boldsymbol{\theta}) = -\ln\frac{|\langle\psi_{\boldsymbol{\theta}}|\psi\rangle|^2}{\langle\psi_{\boldsymbol{\theta}}|\psi_{\boldsymbol{\theta}}\rangle\langle\psi|\psi\rangle} \tag{3.13}$$

evaluated on subsets of configurations [152]. Supervised approaches rely on the availability of sample probability amplitudes from the target quantum state. Such amplitudes can be available in some cases, such as when the target state amplitudes can be computed from another trial state, as it is done for implementing the actions of quantum gates on an NQS [152], or when the target state is available as a full state vector in a sufficiently small system, which can be used for studying the ability of a given NQS ansatz to learn states obtained from ED to test the representation capabilities of the NQS [106, P1, P2].

NQS can also be trained from density measurements $|\psi_{\boldsymbol{\theta}}(\boldsymbol{s})|$, which corresponds to quantum state tomography (QST) in physical terms [153, 154]. In this application, density measurements can be performed in different computational bases, which makes it possible to learn the relative phases of the probability amplitudes in addition to their moduli [153].

For ground-state search, the loss function can simply be chosen as the physical energy of the system [70],

$$L(\boldsymbol{\theta}) = \langle \hat{H} \rangle_{|\psi_{\boldsymbol{\theta}}\rangle}. \tag{3.14}$$

In contrast to MPS, but analogously to other variational ansätze, there are no general *exact* algorithms for efficiently evaluating quantum expectations with NQS. However, it is possible to estimate the energy and other observables based on evaluating the ansatz $\psi_{\boldsymbol{\theta}}(\boldsymbol{s})$ on a subset of configurations obtained by stochastic sampling. This approach is known as VMC and is discussed in Section 3.3.

It can be beneficial to combine different training schemes. For instance, Czischek et al. [155] have observed that pre-training of an NQS ansatz via QST can significantly improve the convergence rate of a subsequent VMC energy optimization.

Beyond pure quantum states, it is also possible to represent mixed quantum states using a variational ansatz by parametrizing the corresponding density operator. When using a neural-network ansatz, the resulting operator is called a neural density operator (NDO). The NDO ansatz can be used for finding steady states and simulating dissipative dynamics in open quantum systems [156–159]. However, we will focus on pure quantum states in the remainder of this thesis.

3.2.3 Examples of neural quantum states

This section provides several examples of network architectures used in NQS, focusing on those most relevant in the context of this dissertation.

Restricted Boltzmann machines

The first network architecture used in an NQS ansatz in the original publication by Carleo and Troyer [70] was the so-called restricted Boltzmann machine (RBM) [160–163]. These networks belong to the class of *energy-based models* [164] and are universal approximators for discrete probability distributions [165].

We follow the presentation in Refs. [162, 163]. An RBM is defined as a set of units arranged in two layers: N_v units in a *visible layer* and N_h units in a *hidden layer*. These units can each take values in

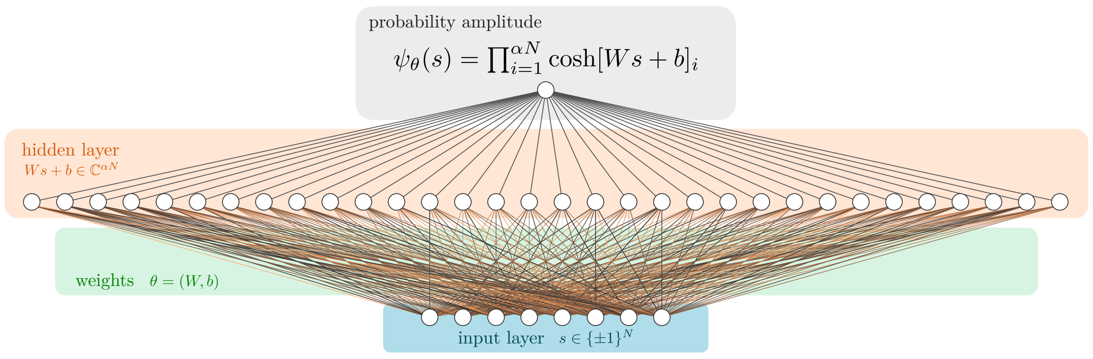

probability amplitude
$\psi_\theta(s) = \prod_{i=1}^{\alpha N} \cosh[Ws+b]_i$
hidden layer
$Ws + b \in \mathbb{C}^{\alpha N}$
weights $\theta = (W, b)$
input layer $s \in \{\pm 1\}^N$

Figure 3.4 | Graphical representation of a restricted Boltzmann machine (RBM) quantum state (without visible bias term). The depicted structure corresponds to the effective one-layer network with (ln) cosh activation, which is equivalent to an RBM quantum state after tracing out the hidden-layer configurations [compare Eq. (3.19)]. This illustration is taken from Publication [P1].

$\mathcal{L} = \{\pm 1\}$ and a configuration $(\boldsymbol{v}, \boldsymbol{h}) \in \mathcal{L}^{N_v} \times \mathcal{L}^{N_h}$ has the associated energy[5]

$$\mathcal{E}(\boldsymbol{v}, \boldsymbol{h}; \boldsymbol{\theta}) = -\boldsymbol{h}^\top \mathbf{W} \boldsymbol{v} - \boldsymbol{a} \cdot \boldsymbol{v} - \boldsymbol{b} \cdot \boldsymbol{h}. \tag{3.15}$$

Here, $\boldsymbol{\theta} = (\mathbf{W}, \boldsymbol{a}, \boldsymbol{b})$ denotes the collection of parameters (the weight matrix $\mathbf{W}$ connecting hidden and visible units and bias vectors $\boldsymbol{a}, \boldsymbol{b}$ for each layer). This energy corresponds to a classical, bipartite Ising model with couplings only between (not within) the two layers[6] and biases as external fields. To each configuration $(\boldsymbol{v}, \boldsymbol{h})$, a joint Boltzmann probability

$$p(\boldsymbol{v}, \boldsymbol{h}; \boldsymbol{\theta}) \propto \exp \mathcal{E}(\boldsymbol{v}, \boldsymbol{h}; \boldsymbol{\theta}) \tag{3.16}$$

is assigned. The probability assigned to a visible configuration is then obtained by tracing over all hidden degrees of freedom, i.e.,

$$p(\boldsymbol{v}; \boldsymbol{\theta}) \propto \sum_{\boldsymbol{h} \in \mathcal{L}^{N_h}} p(\boldsymbol{v}, \boldsymbol{h}; \boldsymbol{\theta}). \tag{3.17}$$

The RBM can be used in an unsupervised learning setting, where a dataset $V = \{\boldsymbol{v}_i\}_{i=1}^{|V|}$ drawn from an unknown probability distribution is given as input. The network model is then optimized to learn this distribution by maximizing the probability assigned to the training data. This can be achieved, e.g., by minimizing the loss function $L(\boldsymbol{\theta}; V) = -\sum_{\boldsymbol{v} \in V} \ln p(\boldsymbol{v}; \boldsymbol{\theta})$ [162, 163].

In the context of neural quantum states, the quantity to be learned is the quantum wave function. Thus, the output of the RBM needs to be complex-valued in general, which can be achieved simply by choosing complex weights, resulting in a complex-valued RBM energy (3.15) and, hence, network output. Systems with local dimension two can be directly mapped to a spin-1/2 system with configuration space $\mathcal{S} = \{\pm 1\}^N$ (i.e., the width of the visible layer equals the physical lattice size) while qudit data can be encoded into such a representation, e.g., using a one-hot encoding [74, 93] (in which case the visible layer needs to be larger than the physical lattice). A visible spin configuration is then mapped to the

[5]The energy (3.15) should not be confused with the energy of a variational state parametrized by an RBM quantum state.
[6]This restriction to bipartite couplings distinguishes the RBM from a general (unrestricted) Boltzmann machine.

probability amplitude

$$\psi_{\boldsymbol{\theta}}(\boldsymbol{s}) = \sum_{\boldsymbol{h}\in\mathcal{L}^{N_h}} \exp \mathcal{E}(\boldsymbol{s}, \boldsymbol{h}; \boldsymbol{\theta}) \tag{3.18}$$

in analogy with Eq. (3.17). Due to the bipartite structure of the RBM model, it is possible to analytically perform the summation over the hidden degrees of freedom, resulting in [70]

$$\psi_{\boldsymbol{\theta}}(\boldsymbol{s}) = \exp(\boldsymbol{a}\cdot\boldsymbol{s}) \prod_{j=1}^{N_h} \cosh[\mathbf{W}\boldsymbol{s} + \boldsymbol{b}]_j \tag{3.19}$$

(up to a constant factor that is neglected here). This expression can be efficiently evaluated in polynomial time with respect to the visible and hidden layer sizes. In NQS applications, the width of the RBM is usually parametrized in terms of the hidden unit density $\alpha = N_h/N_v$. While the RBM wave function obtained in this way is not normalized, this is not a requirement for VMC and, therefore, unproblematic. Since we parametrize the log-probability amplitudes, the RBM model is implemented in practice as

$$\ln \psi_{\boldsymbol{\theta}}(\boldsymbol{s}) = \boldsymbol{a}\cdot\boldsymbol{s} + \sum_{j=1}^{N_h} \ln\cosh[\mathbf{W}\boldsymbol{s} + \boldsymbol{b}]_j. \tag{3.20}$$

While derived from a conceptually different energy-based model, the RBM as determined by Eq. (3.20) is formally equivalent to a feed-forward network with one hidden layer parametrized by weights $\mathbf{W}$ and hidden bias $\boldsymbol{b}$ and using a $\ln\cosh$ activation function, which is followed by a second layer that performs an (unweighted) sum of the resulting activations and the visible bias term $\boldsymbol{a}\cdot\boldsymbol{s}$ (Fig. 3.4).

RBM quantum states have achieved state-of-the-art results for ground state search in lattice spin models [75, 86]. They have also been used to complement fermionic pair-product wave functions [73, 81], applied to the Bose-Hubbard model [91, 93], and for studying the dynamics of quantum magnetism [76, 79]. It is possible to implement translation-invariance in an RBM ansatz by restricting the weights [70] or using generic symmetry-projection schemes [82] (Section 5.1), which can significantly improve the accuracy of optimization results.

A key result that raised significant interest in the capabilities of RBM quantum states was the proof by Deng, Li, and Das Sarma [101] in 2017 that randomly initialized RBMs can efficiently represent states with volume-law entanglement scaling, which is out of reach of more established tensor-network approaches [2, 166]. However, it is not yet clear what such results for random NQS [101, 102] imply regarding the representability of highly entangled quantum states of physical interest (Chapter 6).

Feed-forward neural networks

Fully-connected feed-forward neural networks (FFNNs), as defined in Section 3.2.1, have been used in several applications of NQS. Due to the similarity of the RBM to a one-layer FFNN, multi-layer FFNNs can be seen as a generalization of the RBM that can still be efficiently evaluated. Adding layers to an FFNN increases the representation capabilities of the network and can result in more efficient compression of quantum states [118]. However, increased depth also increases the difficulty of training the network [50].

We utilize FFNN quantum states in our study of the Sachdev-Ye-Kitaev (SYK) model in Publication [P2].

Convolutional neural networks

Convolutional neural networks (CNNs) are designed to make use of spatial structure in the input data to enable more efficient processing of such data compared to fully-connected networks and are most prominently used in image processing and object recognition [154, 167–169]. It has been shown that CNN-based NQS can achieve a polynomially increased efficiency for encoding entanglement in some systems [118] and have been successfully applied to ground state search and time propagation in two-dimensional spin systems [75, 78, 87, 89].

In a convolutional layer, the fully-connected affine transformation of the FFNN is replaced by a convolution. This reduces the connectivity between the layers (and, thus, the number of variational parameters) and enables the network to more efficiently encode local structure in the input data [42]. The output of the convolution operation (and, thus, the network output) is translation-invariant. It is possible to construct a generalization of CNNs, the group-convolutional neural networks (GCNNs), that supports invariance (and, more generally, equivariance) under symmetry operations beyond lattice translations [170, 171] (see Section 5.1 for details).

3.3 Variational Monte Carlo

The usefulness of a variational ansatz for the quantum wave function is determined not just by its capacity to compress the information contained in the quantum state. The existence of efficient algorithms that compute quantities of interest, such as expectation values and variances of observables as well as their gradients, is also crucial.

Paraphrasing Clark [172], the variational Monte Carlo (VMC) approach is concerned with two main tasks:

1. Given a trial state $|\psi\rangle$, compute approximate expectation values of observables.
2. Given a family of variational states $|\psi_{\boldsymbol{\theta}}\rangle$, find optimal parameters $\boldsymbol{\theta}$ for a given objective function.

Crucially, VMC can be applied to a wide variety of variational ansätze. This flexibility is possible because VMC relies on a stochastic approximation of quantum expectation values but comes at the cost of introducing stochastic error that needs to be controlled.

3.3.1 Estimating quantum expectation values

The VMC [72, 173] approach is based on expressing the quantum expectation value $\langle \hat{H} \rangle$ of an observable $\hat{H}$ as a *classical* expectation value $\mathbb{E}_\pi[\tilde{H}]$ of a random variable $\tilde{H}$ using the identity

$$\langle \hat{H} \rangle_{|\psi_{\boldsymbol{\theta}}\rangle} = \frac{\langle \psi_{\boldsymbol{\theta}} | \hat{H} | \psi_{\boldsymbol{\theta}} \rangle}{\langle \psi_{\boldsymbol{\theta}} | \psi_{\boldsymbol{\theta}} \rangle} = \frac{\sum_{\boldsymbol{s},\boldsymbol{s}' \in \mathcal{S}} \psi_{\boldsymbol{\theta}}^*(\boldsymbol{s}') \langle \boldsymbol{s}' | \hat{H} | \boldsymbol{s} \rangle \psi_{\boldsymbol{\theta}}(\boldsymbol{s})}{\sum_{\boldsymbol{s} \in \mathcal{S}} |\psi_{\boldsymbol{\theta}}(\boldsymbol{s})|^2} = \sum_{\boldsymbol{s} \in \mathcal{S}} \pi(\boldsymbol{s}) \tilde{H}(\boldsymbol{s}) = \mathbb{E}_\pi[\tilde{H}] \tag{3.21}$$

where

$$\pi(\boldsymbol{s}) = \frac{|\psi_{\boldsymbol{\theta}}(\boldsymbol{s})|^2}{\sum_{\boldsymbol{s}' \in \mathcal{S}} |\psi_{\boldsymbol{\theta}}(\boldsymbol{s}')|^2} \tag{3.22}$$

is the Born probability density [174] associated with the quantum state and

$$\tilde{H}(\boldsymbol{s}) = \frac{\langle \boldsymbol{s}|\hat{H}|\psi_{\boldsymbol{\theta}}\rangle}{\langle \boldsymbol{s}|\psi_{\boldsymbol{\theta}}\rangle} = \sum_{\boldsymbol{s}' \in \mathcal{S}} \frac{\psi_{\boldsymbol{\theta}}(\boldsymbol{s}')}{\psi_{\boldsymbol{\theta}}(\boldsymbol{s})} \langle \boldsymbol{s}|\hat{H}|\boldsymbol{s}'\rangle \tag{3.23}$$

is a *local estimator* of the observable $\hat{H}$ (if $\hat{H}$ is the Hamiltonian of the system, $\tilde{H}$ is typically called the *local energy*)[7] [72]. The local estimator is only well defined when the quotient $\psi_{\boldsymbol{\theta}}(\boldsymbol{s})/\psi_{\boldsymbol{\theta}}(\boldsymbol{s}')$ is finite. Since $\psi(\boldsymbol{s}) = 0 \Leftrightarrow \pi(\boldsymbol{s}) = 0$, only configurations with well-defined $\tilde{H}$ contribute to the expectation value and Eq. (3.21) can be written as

$$\mathbb{E}_{\pi}[\tilde{H}] = \sum_{\boldsymbol{s} \in \operatorname{supp} \pi} \pi(\boldsymbol{s}) \tilde{H}(\boldsymbol{s}) \tag{3.24}$$

where $\operatorname{supp} \pi = \{\boldsymbol{s} \in \mathcal{S} \mid \pi(\boldsymbol{s}) \neq 0\}$ is the support of π. The amplitude ratio $\psi_{\boldsymbol{\theta}}(\boldsymbol{s})/\psi_{\boldsymbol{\theta}}(\boldsymbol{s}')$ can still pose a challenge in numerical applications when encountering amplitudes of vastly different orders of magnitude.

While exactly evaluating Eq. (3.24) via full summation over the Hilbert space is still exponentially costly, the classical expectation value can be estimated using a Monte Carlo algorithm [173, 175, 176]. Given a sequence of configurations $\mathfrak{C} = (\boldsymbol{s}_i)_{i=1}^{N_{\mathrm{MC}}}$ drawn from the Born distribution, the expectation (3.24) can be approximated by the sum[8]

$$\mathbb{E}[\tilde{H}] \approx \mu[\tilde{H}] = \frac{1}{N_{\mathrm{MC}}} \sum_{i=1}^{N_{\mathrm{MC}}} \tilde{H}(\boldsymbol{s}_i). \tag{3.25}$$

This estimate converges ($\mu[\tilde{H}] \to \mathbb{E}[\tilde{H}]$ as $N_{\mathrm{MC}} \to \infty$) by the central limit theorem and the standard error of the estimate (3.25) is given by the Monte Carlo standard error (MCSE)

$$\varepsilon_{\mathrm{MC}}[\tilde{H}] = \sqrt{\frac{\sigma^2[\tilde{H}]}{N_{\mathrm{MC}}}}, \tag{3.26}$$

where $\sigma^2[\tilde{H}]$ is the empirical standard deviation of $\tilde{H}$ over the samples $\mathfrak{C}$. The Monte Carlo estimate of the expectation value thus converges to the true expectation value with an error asymptotically proportional to $1/\sqrt{N_{\mathrm{MC}}}$. This scaling implies that, to improve the error of an estimate by one decimal place, a hundred times more samples need to be generated. However, generating a suitable sequence $\mathfrak{C}$ is not a trivial task since the probability distribution π is usually as intractable as the full quantum state. Typically, Monte Carlo samples are generated via Markov chain Monte Carlo (MCMC) algorithms (Section 3.3.3). Specific NQS ansätze, so-called neural autoregressive quantum states (NAQS) [177] and states based on recurrent neural networks [178], also allow direct sampling from π via specialized algorithms.

[7] Note that both $\tilde{H}$ and π depend on the current variational state $|\psi_{\boldsymbol{\theta}}\rangle$, although we omit this dependency in the notation for the sake of readability.

[8] We write $\mathbb{E}$ instead of $\mathbb{E}_{\pi}$ in the following when the probability distribution with respect to which the expectation values are computed is clear from the context.

While the Monte Carlo approximation does eliminate one summation over the exponentially large Hilbert space, the local estimator (3.23) still formally contains an unconstrained and thus exponentially costly sum. However, this sum is effectively restricted to the set of connected configurations

$$\mathrm{Conn}(\boldsymbol{s}; \hat{H}) = \{\boldsymbol{s}' \in \mathcal{S} \mid \langle \boldsymbol{s}|\hat{H}|\boldsymbol{s}'\rangle \neq 0\}, \tag{3.27}$$

i.e., the configurations for which the corresponding matrix element is nonzero. The local estimator thus becomes tractable to evaluate if the operator $\hat{H}$ is sufficiently sparse in the computational basis and if an efficient algorithm exists which, given a configuration $\boldsymbol{s}$, yields the connected configurations $\boldsymbol{s}' \in \mathrm{Conn}(\boldsymbol{s}; \hat{H})$ and their matrix elements $\langle \boldsymbol{s}|\hat{H}|\boldsymbol{s}'\rangle$. This is true in particular for sufficiently local operators in lattice models. Efficient computation of matrix elements for fermionic lattice models in the Fock basis, in which matrix elements of typical operators contain non-local parity information, typically require more advanced encoding schemes [97, 179].

In summary, the requirements on a given ansatz and operator to support efficient VMC estimation are the availability of algorithms for

1. the efficient evaluation of the probability amplitudes $\psi_{\boldsymbol{\theta}}(\boldsymbol{s})$;
2. the efficient generation of samples from the Born distribution $\pi(\boldsymbol{s}) \propto |\psi_{\boldsymbol{\theta}}(\boldsymbol{s})|^2$; and
3. the efficient computation of the connected configurations $\mathrm{Conn}(\boldsymbol{s}; \hat{H})$ and corresponding matrix elements for all observables of interest.

3.3.2 VMC optimization

The most common task in VMC simulations is to find a variational approximation of the ground state, i.e., the minimum energy eigenstate for a given Hamiltonian $\hat{H}$. This means finding optimal variational parameters to minimize the Rayleigh quotient [72]

$$\boldsymbol{\theta}_\star = \arg\min_{\boldsymbol{\theta}\in\mathcal{X}} \langle \hat{H} \rangle_{|\psi_{\boldsymbol{\theta}}\rangle} = \arg\min_{\boldsymbol{\theta}\in\mathcal{X}} \frac{\langle \psi_{\boldsymbol{\theta}}|\hat{H}|\psi_{\boldsymbol{\theta}}\rangle}{\langle \psi_{\boldsymbol{\theta}}|\psi_{\boldsymbol{\theta}}\rangle}, \tag{3.28}$$

which is typically done by gradient-based optimization. It is possible to optimize the parameters by simple gradient descent, i.e., by iteratively performing the update

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \tau_n \nabla_{\boldsymbol{\theta}} \langle \hat{H} \rangle_{|\psi_{\boldsymbol{\theta}_n}\rangle} \tag{3.29}$$

with discrete steps $n \in [N_{\text{steps}}]$ and a step size τ_n (potentially varying over the course of the optimization). In VMC, the gradient of an observable can be estimated using the same Monte Carlo samples as for computing expectation values [see Eq. (4.20) for the explicit formula]. Alternatively, gradients can be obtained while evaluating the corresponding expectation value using automatic differentiation (Section 7.2). It can be helpful to apply more advanced optimization schemes to improve convergence [180]. These include adaptive-gradient methods such as RMSProb [181], Adam [182], or AdaGrad [183], which utilize a per-parameter adaptive learning rate that varies based on the history of previous gradients. While such methods can achieve faster convergence in many cases, they can also be more prone to overfitting in machine learning applications [184, 185]. It is also possible to perform the ground state optimization by approximating the time-evolution according to the imaginary-time Schrödinger equation (Section 4.1.3). This approach is known as stochastic reconfiguration (SR) in the VMC literature [72, 186, 187]. A formally equivalent training procedure for classical probability distributions has been independently developed in the machine learning literature under the name natural gradient descent (NGD) [188, 189]. In any case, the best choice of optimization algorithm will depend on the specific problem and can therefore be seen as an additional hyperparameter of the training scheme.

Since the minimum and maximum eigenvalues $E_{\mathrm{min/max}}$ provide bounds for the energy expectation of any quantum state,

$$E_{\mathrm{min}}(\hat{H}) \leq \langle \hat{H} \rangle_{|\psi\rangle} \leq E_{\mathrm{max}}(\hat{H}) \qquad \text{for any } |\psi\rangle \in \mathcal{H}, \tag{3.30}$$

the variational energy expectation value $\langle \hat{H} \rangle_{|\psi_{\boldsymbol{\theta}}\rangle}$ provides an upper bound for the true ground state energy for arbitrary parameters $\boldsymbol{\theta} \in \mathcal{X}$. If the variational manifold includes the true ground state of the system (or states sufficiently close to it), the global variational minimum can provide a good approximation of the physical ground state. However, a VMC-based optimization run is not guaranteed to find such a global energy minimum. Nor is the global minimum achievable by a given variational ansatz necessarily close to the true ground state. For systems with a small energy gap, this can be true even when the variational energy is low. In this case, non-zero components of low-lying excited states may still significantly influence observables. Besides monitoring energy convergence, it is thus vital to assess whether the final variational minimum is a valid candidate for the true ground state by evaluating the physical properties of the state.

The optimal variational parameters are not generally unique. Different sets of parameters can describe the same quantum state (up to a gauge degree of freedom $\gamma \in \mathbb{C} \setminus \{0\}$), i.e., $|\psi_{\boldsymbol{\theta}_1}\rangle = \gamma|\psi_{\boldsymbol{\theta}_2}\rangle$ for $\boldsymbol{\theta}_1 \neq \boldsymbol{\theta}_2$. Furthermore, if the ground state of $\hat{H}$ is degenerate, any element of the minimum-eigenvalue eigenspace is a valid solution to Eq. (3.28).

A key tool to monitor VMC convergence is the *vanishing-variance property* [72, 114, 190]: The variance of $\hat{H}$ is zero in every $\hat{H}$-eigenstate $|\eta\rangle$, i.e.,

$$\langle (\hat{H} - \langle \hat{H} \rangle_{|\eta\rangle})^2 \rangle_{|\eta\rangle} = 0. \tag{3.31}$$

Consequently, the local estimator (3.23) is constant in an eigenstate,

$$\forall \boldsymbol{s} \in \mathcal{S}\colon\ \tilde{H}(\boldsymbol{s}) = \langle \hat{H} \rangle_{|\eta\rangle} \qquad \Leftrightarrow \qquad \mathrm{Var}[\tilde{H}] = 0, \tag{3.32}$$

and, therefore, the estimated variance can be used to monitor the convergence of the VMC optimization to an eigenstate. Furthermore, since the MCSE (3.26) is proportional to the square root of the variance, Monte Carlo estimates become more accurate with the convergence of the optimization run. This property is crucial for obtaining high-accuracy VMC estimates [114]. Note, however, that the vanishing-variance property cannot be used to distinguish ground states from excited eigenstates.

While VMC can be categorized as a quantum Monte Carlo (QMC) technique, it differs from typical QMC approaches in two main characteristics:

On the one hand, VMC is *biased* by the ansatz [172]: While computed quantities ideally converge to the true values for a target state in the limit of infinite Monte Carlo samples in other, non-variational QMC methods [72], VMC estimates only converge to the expectation values for the current trial state $|\psi_{\boldsymbol{\theta}}\rangle$. The accuracy of VMC results is thus determined by the expressiveness of the ansatz and the ability of the optimization scheme to reach the target state.

On the other hand, VMC does not suffer from the *QMC sign problem*, a phenomenon which causes an exponential slowdown of convergence for estimates in systems with non-stoquastic[9] Hamiltonians for some QMC schemes [71, 72, 192, 193].

However, being free from the QMC sign problem does not mean that VMC algorithms are unaffected by the sign structure or, more generally, the structure of relative phases of the target ground state [194–197]. Suppose the initial state is determined by randomly drawn weights, as is typically done in NQS simulations. This initial state will generally have a uniform sign structure. If the same is not true for the target state, the training algorithm needs to optimize both modulus and relative phases of the probability amplitudes, the latter typically being harder to learn [194]. This increases the training difficulty, and the VMC optimization can become stuck in local energy minima with a sign structure distinct from the actual ground state [178, 195]. This difficulty can be alleviated by using more elaborate training schemes; proposals in the literature include alternating between modulus and phase training steps [195], using a genetic algorithm [196], or obtaining the phase structure from separate optimization using a simulated-annealing process [197]. Enforcing symmetries of the problem in the NQS ansatz has also been observed to help [82, 88, 89] (Section 5.1.2).

3.3.3 Markov chain Monte Carlo and error diagnostics

Estimating expectation values in VMC requires the ability to efficiently draw samples distributed according to the Born distribution (3.22). However, doing this exactly for an arbitrary variational state is still an exponentially hard problem. The sampling problem can be made tractable for general NQS (as well as other types of variational states) by using a Markov chain Monte Carlo (MCMC) approach, specifically the Metropolis-Hastings algorithm [175, 176], which is reviewed here following Refs. [198, 199].

Markov chains

MCMC works by constructing a stochastic process of samples $\{\boldsymbol{s}^{(i)}\}_{i=1}^{N_{\mathrm{MC}}}$, the *Markov chain*, in which each element is obtained from its predecessor by an update rule. This update is described by the *jump distribution* $T(\boldsymbol{s}^{(i+1)}|\boldsymbol{s}^{(i)})$, which represents the probability of transitioning to $\boldsymbol{s}^{(i+1)}$ from $\boldsymbol{s}^{(i)}$. If the jump distribution *preserves* the probability distribution π, i.e., if

$$\pi(\boldsymbol{s}') = \sum_{\boldsymbol{s}\in\mathcal{S}} T(\boldsymbol{s}'|\boldsymbol{s})\pi(\boldsymbol{s}) \tag{3.33}$$

holds for all $\boldsymbol{s}' \in \mathcal{S}$, and the initial sample $\boldsymbol{s}^{(1)}$ is drawn from π, then the chain is *stationary* with π as its *equilibrium distribution*[10]. Averages of random variables $A^{(i)} = A(\boldsymbol{s}^{(i)})$ over a stationary chain can

[9]A Hermitian operator is called stoquastic if all its off-diagonal matrix elements in the computational basis are non-positive, which ensures that the ground state wave function can be expressed using only non-negative (and, hence, real-valued) probability amplitudes in that basis [191].

[10]A Markov chain is said to be stationary if its samples follow the same equilibrium distribution regardless of their position in the chain.

then be used to estimate expectations values with respect to π

$$\mathbb{E}_\pi[A] = \lim_{N_{\mathrm{MC}}\to\infty} \frac{1}{N_{\mathrm{MC}}} \sum_{i=1}^{N_{\mathrm{MC}}} A^{(i)} \tag{3.34}$$

by the Markov chain central limit theorem (MC-CLT). This is possible even when it is not tractable to sample from π directly, as long as the update steps can be performed efficiently. However, since it is not generally possible to draw the initial sample $\boldsymbol{s}^{(1)}$ from π either, the chain must first converge to the equilibrium distribution when started from an arbitrary configuration. In this initial phase (which is referred to as *burn-in* or *warm-up phase*), chain averages can be biased and provide poor estimates of expectation values. A number of initial samples are usually discarded to counteract this bias. Nonetheless, convergence to the equilibrium distribution can be difficult to assess. While diagnostic criteria (as introduced below) can increase the confidence in the convergence of results, they can still be fooled by intermittent "pseudo-convergence" to a seemingly converged state [115, 198], which needs to be kept in mind in practical simulations.

Metropolis-Hastings algorithm

The Metropolis-Hastings algorithm provides a transition rule that preserves the distribution π as required by the MC-CLT. The update from configuration $\boldsymbol{s}^{(i)}$ to $\boldsymbol{s}^{(i+1)}$ is performed by first generating a proposed update $\boldsymbol{s}'$ according to a *proposal distribution* $Q(\boldsymbol{s}'|\boldsymbol{s}^{(i)})$ and then accepting this proposed sample randomly with probability

$$P_{\mathrm{acc}} = \min\left(1, \frac{\pi(\boldsymbol{s}')Q(\boldsymbol{s}^{(i)}|\boldsymbol{s}')}{\pi(\boldsymbol{s}^{(i)})Q(\boldsymbol{s}'|\boldsymbol{s}^{(i)})}\right), \tag{3.35}$$

which simplifies to $P_{\mathrm{acc}} = \min(1, \pi(\boldsymbol{s}')/\pi(\boldsymbol{s}^{(i)}))$ for a symmetric proposal distribution $Q(\boldsymbol{s}|\boldsymbol{s}') = Q(\boldsymbol{s}'|\boldsymbol{s})$. By this rule, an update is automatically accepted if it leads to a configuration with a higher probability mass. If it leads to a configuration with a lower probability mass, it is accepted randomly according to the probability ratio. If the update is accepted, then $\boldsymbol{s}^{(i+1)} = \boldsymbol{s}'$. Otherwise, it is rejected and $\boldsymbol{s}^{(i+1)} = \boldsymbol{s}^{(i)}$.

The Metropolis-Hastings algorithm yields a jump distribution that satisfies the so-called *detailed balance* condition

$$\pi(\boldsymbol{s})T(\boldsymbol{s}'|\boldsymbol{s}) = T(\boldsymbol{s}|\boldsymbol{s}')\pi(\boldsymbol{s}') \qquad \forall \boldsymbol{s}, \boldsymbol{s}' \in \mathcal{S}. \tag{3.36}$$

If, additionally, the process is *ergodic*, i.e., any configuration with non-zero probability can be reached in a finite number of steps through applying the Metropolis-Hastings update, the algorithm generates a Markov chain with equilibrium distribution π as required for MCMC sampling. However, the convergence rate of the estimate depends on the details of the update and, thus, the proposal rule. In particular, it depends on the rate at which suggested updates are accepted (see Section 5.2.2 for an example).

The following are examples for common proposal rules from $\boldsymbol{s}$ to $\boldsymbol{s}'$:

1. *Local update.* Select a lattice site $j \in [N]$ uniformly at random and replace s_j by another uniformly drawn value $\sigma \in \mathcal{L}_j \setminus \{s_j\}$. The proposal is then given by $s_i' = s_i(1-\delta_{ij}) + \sigma\delta_{ij}$. For spin-1/2 systems, this corresponds to flipping a single spin.

2. *Exchange update.* Select a pair of distinct sites $(j,k) \in [N]^2$, $j \neq k$, uniformly at random and exchange the values at both sites. This update rule preserves the number of occurrences of each value in $\boldsymbol{s}$. It is thus suitable for systems with constraints such as particle number conservation or, in spin systems, conserved magnetization $\sum_{i=1}^{N} s_i$ (Section 5.1.1).

3. *Hamiltonian update.*[11] In order to preserve symmetries of the Hamiltonian, it can be helpful to propose updates corresponding to its non-zero matrix elements, for instance, by choosing $\boldsymbol{s}' \in \mathrm{Conn}(\boldsymbol{s}; \hat{H})$ uniformly at random. This approach is used, e.g., in Ref. [200].

It is also possible to combine update rules. For instance, Ref. [78] suggests combining local spin-flip updates with an occasional global flip of all spins in simulations of the spin-1/2 transverse-field Ising model. This additional global update leads to a more symmetric exploration of the full configuration space and can improve the convergence of expectation estimates.

Autocorrelation and MCMC standard error

While the elements of a stationary Markov chain are distributed according to the invariant distribution, they are not statistically independent. Thus, the definition of the MCSE (3.26), which requires independent samples, is not applicable, and the autocorrelation between samples must be taken into account. This is done in the the MCMC standard error, which is given by

$$\varepsilon_{\mathrm{MCMC}}[A] = \sqrt{\frac{\sigma^2[A]}{N_{\mathrm{eff}}[A]}}. \tag{3.37}$$

As in the MCSE formula, $\sigma^2[A]$ is the empirical variance of A over the chain but the sample size is replaced with the effective sample size N_{eff} given by

$$N_{\mathrm{eff}}[A] = \frac{N_{\mathrm{MC}}}{1 + 2\tau[A]} \tag{3.38}$$

where $\tau[A]$ is the integrated autocorrelation time of the sequence $\{A^{(i)}\}$ [201]. Note that the autocorrelation time and, therefore, the effective sample size depends on the observable. For an uncorrelated chain, $N_{\mathrm{eff}} = N_{\mathrm{MC}}$ so that $\varepsilon_{\mathrm{MCMC}} = \varepsilon_{\mathrm{MC}}$.

The autocorrelation between samples can be reduced by *thinning*, i.e., by averaging over a reduced chain that includes a subset of samples. Usually, this is done by including only every m-th sample for sufficiently large m. While thinning reduces autocorrelation, since more updates are performed between each included sample, the effective sample size is also reduced by a factor of m. Thus, thinning the chain beyond the autocorrelation time can be counterproductive. Generally, it is advisable only to employ thinning as a performance optimization [198], e.g., to reduce the number of evaluations of local estimators (3.23).

[11]Note that this definition of a *Hamiltonian update*, which is also used in the NetKet framework [P3, P4], is unrelated to the *Hamiltonian Monte Carlo* algorithm, which is an MCMC sampling algorithm for continuous distributions based on (classical) Hamiltonian dynamics [199].

Diagnosing MCMC convergence

Unbiased estimates of expectation values require the chain to be stationary, and the necessary time to convergence is not known a priori. Additionally, the chain can become stuck in a high-probability configuration if the probability ratio of the proposed updates becomes so low that they are constantly rejected by the Metropolis-Hastings scheme, and sudden jumps in the estimated expectations can occur when the chain starts to explore previously undersampled regions of the configuration space after [115]. While these issues do not theoretically prevent the convergence of the estimate (3.34) in the limit $N_{\mathrm{MC}} \to \infty$, they can introduce bias in practical simulations with finite sample size. Therefore, it is vital to implement diagnostics that can help detect such errors; these include the autocorrelation time, effective sample size, and MCMC standard error, as discussed above.

MCMC simulations are typically performed using multiple chains that are updated in parallel. This is beneficial for computational performance: Applying operations, particularly evaluating the probability distribution π, to a *batch* of configurations at once can be more efficient than performing the same task sequentially (especially in GPU computing) [P4]. Furthermore, VMC simulations are typically distributed over several nodes in an HPC cluster to allow for a massive increase in tractable sample sizes [71]. In this case, independent chains are run on each node so that communication between nodes can be restricted to collecting ensemble properties.

Beyond performance benefits, the availability of multiple chains introduces the possibility of comparing ensemble quantities between chains to detect convergence issues. One way to condense this information to a single, real-valued quantity is the so-called *Gelman-Rubin diagnostic* $\hat{R}$, which we present here following Ref. [202][12]. Let $\boldsymbol{s}^{(m,i)}$ denote the $(i \in [N_{\mathrm{MC}}])$-th configuration of the $(m \in [N_{\mathrm{C}}])$-th independent Markov chain and $A^{(m,i)} = A(\boldsymbol{s}^{(m,i)})$ the corresponding local observable. Defining the average per-chain variance estimator

$$W[A] = \frac{1}{M} \sum_{m=1}^{N_{\mathrm{C}}} W_m[A], \qquad W_m[A] = \frac{1}{N_{\mathrm{MC}}} \sum_{i=1}^{N_{\mathrm{MC}}} (A^{(m,i)} - \mu_m[A])^2, \tag{3.39}$$

and the between-chain variance estimator

$$B[A] = \frac{1}{N_{\mathrm{C}} - 1} \sum_{i=1}^{N_{\mathrm{C}}} (\mu_m[A] - \mu[A])^2, \tag{3.40}$$

where $\mu_m[A] = N_{\mathrm{MC}}^{-1} \sum_{i=1}^{N_{\mathrm{MC}}} A^{(m,i)}$ and $\mu[A] = N_{\mathrm{C}}^{-1} \sum_{i=1}^{N_{\mathrm{C}}} \mu_m[A]$ denote the per-chain and global averages, respectively, $\hat{R}$ is given by

$$\hat{R}[A] = \sqrt{1 + \frac{B[A]}{W[A]}}. \tag{3.41}$$

This factor converges to one in the limit of infinite sample size, $\lim_{N_{\mathrm{MC}} \to \infty} \hat{R} = 1$. Parts of the statistics literature suggest empirical limits of $\hat{R} < 1.1$ [202] or, in more recent works, $\hat{R} < 1.01$ [203] as a criterion to consider an estimate sufficiently converged. However, acceptable limits depend on the specific application.

[12] In this context, the hat notation used for $\hat{R}$ is used to denote a statistical estimator, not a quantum operator.

Several non-stationary chains, where the values of $A^{(i)}$ systematically increase (or decrease), can interact in such a way that this issue is not detected by the plain $\hat{R}$ diagnostic of Eq. (3.41). This can be remedied in some cases by a simple measure: Splitting each chain in the middle and treating both halves as separate chains will cause the chain averages to disagree if the process exhibits a drift. This adjustment yields the so-called split-$\hat{R}$ diagnostic [202]. Several extensions of this diagnostic have been suggested in the literature to further increase the detectable types of convergence issues, particularly the rank-normalized $\hat{R}$ in Ref. [203].

Diagnostic criteria are helpful to increase confidence in the correctness of MCMC estimates and for early identification of errors. However, it is important to remember that while a diagnostic may detect the presence of errors, it cannot fully prove their absence [198].

4 Time propagation of neural quantum states

Neural quantum states can be used to study the dynamics of quantum systems. The following sections will provide an overview of the time-dependent variational principle (TDVP) and its application to NQS via time-dependent variational Monte Carlo (t-VMC). This is followed by Publication [P1], which analyzes in more detail the influence of stochastic noise on t-VMC propagation and provides an interpretation of this effect in machine-learning terms.

4.1 Variational time propagation

At the core of NQS time propagation algorithms lies the idea of encoding the time-dependence of the system in a trajectory $t \mapsto \boldsymbol{\theta}(t)$ of variational parameters. This curve in parameter space directly corresponds to a curve $|\psi_{\boldsymbol{\theta}(t)}\rangle$ in the full Hilbert space via the variational mapping (3.5). In the full Hilbert space, quantum dynamics are governed by the time-dependent Schrödinger equation (TDSE)[13]

$$i\frac{\mathrm{d}}{\mathrm{d}t}|\psi_{\boldsymbol{\theta}(t)}\rangle = \hat{H}(t)|\psi_{\boldsymbol{\theta}(t)}\rangle. \tag{4.1}$$

The goal of variational time propagation algorithms is to compute a parameter-space trajectory that approximates the dynamics induced by the TDSE. Since the states are constrained to the variational manifold, $|\psi_{\boldsymbol{\theta}(t)}\rangle \in \mathcal{M}$, an evolution leading the state outside this manifold cannot be fully captured. However, it is possible to derive an effective equation of motion for the parameters relying on the projection of the exact time evolution to the tangent space of the variational manifold (Fig. 4.1) [204, 205]. Ideally, this yields a trajectory that approximates the quantum time evolution to a sufficient degree in order to extract observables of interest, even in the presence of projection error induced by the restriction to $\mathcal{M}$.

There are various ways to obtain a suitable parameter-space trajectory. One set of methods, which are predominantly applied in the context of NQS, is based on obtaining an equation of motion

$$\mathcal{F}(t, \boldsymbol{\theta}(t), \dot{\boldsymbol{\theta}}(t)) = 0 \tag{4.2}$$

for the variational parameters, which approximates the TDSE (4.1). Such an equation of motion can then be solved by standard means for differential equations. In particular, numerical solvers using Runge-Kutta schemes can be employed, which is done in our work [P1, P4] as well as other publications in the NQS literature [70, 76, 78, 79, 87, 110] (see Section 4.3 of Ref. [P4] for details on the Runge-Kutta scheme).

There are various ways of deriving a variational equation of motion of the form (4.2). Generally, derivations of a variational equation of motion lead to one of the three forms discussed in the following. The reader is referred to Refs. [32, 190] for further, more detailed discussions.

[13]Note that here and throughout this thesis, we use units where the Planck constant is set to unity, $\hbar = 1$.

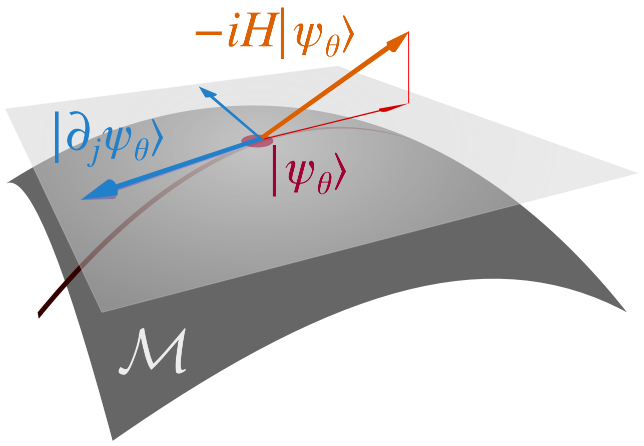


Figure 4.1 | Graphical representation of the TDVP time evolution, which is obtained by locally projecting the flow of the exact TDSE, the direction of which is locally given by $-i\hat{H}|\psi_{\boldsymbol{\theta}(t)}\rangle$, to the tangent space of the variational manifold $\mathcal{M}$, which is spanned by the partial derivatives $\{|\partial_j\psi_{\boldsymbol{\theta}(t)}\rangle\}_{j=1}^{M}$.
This illustration is based on Ref. [206], which has been released into the public domain by its authors.

4.1.1 Time-dependent variational principle for holomorphic mappings

First, we consider the case where the variational mapping is holomorphic, which means that $\boldsymbol{\theta} \in \mathcal{X} \subseteq \mathbb{C}^M$ is a complex vector and all of the functions $\theta_j \mapsto \psi_{\boldsymbol{\theta}}(\boldsymbol{s})$, $j \in [M]$, are complex differentiable (the nonholomorphic and real-parameter cases are discussed in Section 4.1.2). This condition is equivalent to requiring that the Cauchy-Riemann differential equations

$$\frac{\partial\psi_{\boldsymbol{\theta}}(\boldsymbol{s})}{\partial\,\mathrm{Re}[\theta_j]} + i\frac{\partial\psi_{\boldsymbol{\theta}}(\boldsymbol{s})}{\partial\,\mathrm{Im}[\theta_j]} = 0 \qquad \Leftrightarrow \qquad \frac{\partial\psi_{\boldsymbol{\theta}}(\boldsymbol{s})}{\partial\theta_j^*} = 0 \tag{4.3}$$

hold. Here, we use the Wirtinger derivative operators [207]

$$\frac{\partial}{\partial\theta_j} = \frac{\partial}{\partial\,\mathrm{Re}[\theta_j]} - i\frac{\partial}{\partial\,\mathrm{Im}[\theta_j]}, \qquad \frac{\partial}{\partial\theta_j^*} = \frac{\partial}{\partial\,\mathrm{Re}[\theta_j]} + i\frac{\partial}{\partial\,\mathrm{Im}[\theta_j]}. \tag{4.4}$$

In this sense, holomorphic wave functions depend on $\boldsymbol{\theta}$ but not its conjugate $\boldsymbol{\theta}^*$, which reduces the amount of derivative information that needs to be computed. Many common ansätze satisfy this condition, particularly RBMs, FFNNs, and CNNs, as long as their activation functions are holomorphic. However, separately applying a nonlinear activation to real and imaginary parts of its input prevents complex differentiability for any non-trivial activation, since Re and Im are nonholomorphic functions.

Following, for instance, Refs. [78, 190], the variational equation of motion can be obtained starting from the requirement that, for an infinitesimal time increment δt, the overlap of the exact time evolved state with the state obtained by applying an infinitesimal update $\delta\boldsymbol{\theta}$ to the variational parameters is maximized, i.e.,

$$\delta\boldsymbol{\theta} = \arg\min_{\delta\boldsymbol{\vartheta}} D_{\mathrm{FS}}\left(e^{-i\delta t\hat{H}(t)}|\psi_{\boldsymbol{\theta}(t)}\rangle, |\psi_{\boldsymbol{\theta}+\delta\boldsymbol{\vartheta}}\rangle\right) \tag{4.5}$$

where D_{FS} denotes the Fubini-Study distance

$$D_{\mathrm{FS}}(|\psi\rangle, |\phi\rangle) = \arccos \sqrt{\frac{\langle\phi|\psi\rangle\langle\psi|\phi\rangle}{\langle\psi|\psi\rangle\langle\phi|\phi\rangle}}. \tag{4.6}$$

For $\delta t \to 0$, this yields the equation of motion [190]

$$\mathbf{S}(\boldsymbol{\theta}(t))\,\dot{\boldsymbol{\theta}} = -i\boldsymbol{F}(\boldsymbol{\theta}(t), t) \qquad \text{(holomorphic TDVP).} \tag{4.7}$$

Two quantities determine this equation of motion: On the left-hand side, the quantum Fisher matrix (QFM)[14] $\mathbf{S}(\boldsymbol{\theta})$, named after the closely related Fisher information matrix associated with classical probability distributions [189, 208], with matrix elements

$$S_{ij}(\boldsymbol{\theta}) = \frac{\langle\partial_i\psi_{\boldsymbol{\theta}}|1 - \hat{P}_{\boldsymbol{\theta}}|\partial_j\psi_{\boldsymbol{\theta}}\rangle}{\langle\psi_{\boldsymbol{\theta}}|\psi_{\boldsymbol{\theta}}\rangle}, \quad \hat{P}_{\boldsymbol{\theta}} = \frac{|\psi_{\boldsymbol{\theta}}\rangle\langle\psi_{\boldsymbol{\theta}}|}{\langle\psi_{\boldsymbol{\theta}}|\psi_{\boldsymbol{\theta}}\rangle}, \tag{4.8}$$

where $|\partial_j\psi_{\boldsymbol{\theta}}\rangle = (\partial/\partial\theta_j)|\psi_{\boldsymbol{\theta}}\rangle$; on the right-hand side, a term proportional to the energy gradient, with elements[15]

$$F_j(\boldsymbol{\theta}, t) = [\nabla_{\boldsymbol{\theta}}\langle\hat{H}(t)\rangle_{|\psi_{\boldsymbol{\theta}}\rangle}]_j \tag{4.9}$$

The structure of the equation of motion can be described as follows: The TDSE induces a flow in parameter space, which is determined by the Hamiltonian as the generator of time evolution that enters into Eq. (4.7) via the force term of Eq. (4.9). However, the mapping between parameter space and quantum Hilbert space is highly nonlinear. A small change in parameters may affect the quantum state to varying degrees, depending on the direction of the change. This geometric structure is incorporated into the equation of motion via the QFM (4.8). Specifically [190],

$$D_{\mathrm{FS}}(|\psi_{\boldsymbol{\theta}}\rangle, |\psi_{\boldsymbol{\theta}+\epsilon\boldsymbol{\vartheta}}\rangle) = \epsilon^2\boldsymbol{\vartheta}^\dagger\,\mathbf{S}(\boldsymbol{\theta})\,\boldsymbol{\vartheta} + o(\epsilon^2), \tag{4.10}$$

where $o(\epsilon^2)$ denotes terms vanishing asymptotically faster than ϵ^2 as $\epsilon \to 0$. Thus, the QFM plays the role of the metric tensor by inducing a position-dependent curvature on the parameter space that represents the geometry of the parametrized quantum state. Due to this structure, $\mathbf{S}$ is not generally invertible. The null space of $\mathbf{S}(\boldsymbol{\theta})$ contains redundant directions that, at a fixed point $\boldsymbol{\theta}$, do not affect the resulting quantum state. Parameter changes in redundant directions may, however, affect gauge degrees of freedom, i.e., norm and global phase of the variational state.

To simplify the presentation, we will temporarily assume $\mathbf{S}$ to be invertible. Given parameter-space functions[16] $O_1, O_2 \colon \mathcal{X} \to \mathbb{C}$, we can define a *Poisson bracket* [205, 210]

$$\{O_1, O_2\}(\boldsymbol{\theta}) = i\sum_{i,j=1}^{M}\left[\frac{\partial O_1(\boldsymbol{\theta})}{\partial\theta_i}[\mathbf{S}(\boldsymbol{\theta})^{-1}]_{ij}\frac{\partial O_2(\boldsymbol{\theta})}{\partial\theta_j^*} - \frac{\partial O_2(\boldsymbol{\theta})}{\partial\theta_i}[\mathbf{S}(\boldsymbol{\theta})^{-1}]_{ij}\frac{\partial O_1(\boldsymbol{\theta})}{\partial\theta_j^*}\right]. \tag{4.11}$$

[14] The matrix $\mathbf{S}$ is also referred to as the quantum geometric tensor (QGT) [P4], due to its geometric interpretation discussed below, or “S matrix”, particularly in VMC literature [70, 72].

[15] For complex parameters, the gradient is defined here as $[\nabla_{\boldsymbol{\theta}}]_j = \partial/\partial\theta_j^*$ (i.e., as the vector of conjugate derivatives) since this corresponds to the direction of steepest ascent for real functions of complex arguments [209].

[16] This includes quantum expectation values, i.e., any function of the form $O(\boldsymbol{\theta}) = \langle\hat{O}\rangle_{|\psi_{\boldsymbol{\theta}}\rangle}$ for a Hermitian operator $\hat{O}$.

which can be used to write the equation of motion (4.7) in the equivalent form

$$\dot{\theta}_j = \{H, \theta_j\} \tag{4.12}$$

where $H(\boldsymbol{\theta}) = \langle \hat{H} \rangle_{|\psi_{\boldsymbol{\theta}}\rangle}$ is the energy expectation value. More generally, for any (time-independent) parameter-space function O,

$$\dot{O} = \{H, O\}. \tag{4.13}$$

This mathematical structure closely resembles classical mechanics in complex coordinates [210]. Hence, in the TDVP formalism, quantum dynamics are mapped to classical dynamics in parameter space, with all quantum effects incorporated into the space's geometry via the QFM. Equation (4.13) implies that the TDVP dynamics respect energy conservation for time-independent Hamiltonians, even when the variational trajectory deviates from the true quantum dynamics. Other constants of motion are not generally conserved unless they satisfy additional compatibility criteria depending on the ansatz [129].

Note that while the invertibility of $\mathbf{S}(\boldsymbol{\theta})$ simplifies the presentation, it is not a necessary assumption. The equation of motion and energy conservation can be derived for a singular QFM (which is the standard case in NQS applications) as well [78, 205].

4.1.2 Variational principles for nonholomorphic mappings

For a nonholomorphic variational mapping of complex parameters, the conjugate derivatives $\partial\psi_{\boldsymbol{\theta}}(\boldsymbol{s})/\partial\theta_j^*$ (and thus the corresponding log-derivatives) are generally nonzero. Therefore, additional derivative information needs to be taken into account. In that case, there are no savings in computational effort compared to treating the ansatz as a real-parameter variational mapping and separately computing derivatives for real and imaginary parts.

For real-parameter mappings $\boldsymbol{\theta} \in \mathbb{R}^M$, two distinct versions of Eq. (4.7) corresponding to different variational principles (VPs) can be obtained, depending on the starting point of the derivations [32, 211–213]. These VPs can be expressed in the form of Eq. (4.7) as

$$\mathrm{Re}[\mathbf{S}(\boldsymbol{\theta})]\,\dot{\boldsymbol{\theta}} = \mathrm{Re}[-i\boldsymbol{F}(\boldsymbol{\theta}, t)] \qquad \text{(McLachlan's VP)} \tag{4.14}$$

$$\mathrm{Im}[\mathbf{S}(\boldsymbol{\theta})]\,\dot{\boldsymbol{\theta}} = \mathrm{Im}[-i\boldsymbol{F}(\boldsymbol{\theta}, t)] \qquad \text{(real-valued TDVP)} \tag{4.15}$$

which are referred to [32, 213] as *McLachlan's VP* [211] and the *real-valued TDVP* [212], respectively. Both variational principles become equivalent if the real-valued parametrization can be expressed in terms of $M/2$ complex parameters $z_j = \theta_{\varpi(2j)} + i\,\theta_{\varpi(2j-1)}$ (where ϖ is a suitable permutation of the indices), so that the resulting variational mapping is holomorphic [213].

Although the real-valued TDVP has some theoretical advantages (in particular, it guarantees energy conservation just as the holomorphic TDVP [32]), McLachlan's VP is predominantly used in practice, since $\mathrm{Re}[\mathbf{S}]$ is a Hermitian, positive-semidefinite matrix while $\mathrm{Im}[\mathbf{S}]$ is skew-Hermitian. This makes solving the equation of motion in McLachlan's VP easier for standard algorithms and usually results in more stable propagation [110].

4.1.3 Imaginary time propagation and stochastic reconfiguration

Variational time-evolution methods are not limited to real-time dynamics. Most notably, using the same approaches, it is possible to approximate the evolution according to the imaginary-time Schrödinger equation

$$\frac{\mathrm{d}}{\mathrm{d}t}|\psi_{\boldsymbol{\theta}(t)}\rangle = -(\hat{H} - \langle\hat{H}\rangle_{|\psi_{\boldsymbol{\theta}(t)}\rangle})|\psi_{\boldsymbol{\theta}(t)}\rangle, \tag{4.16}$$

which is a diffusion equation that drives the state towards lower energies and will eventually project its initial state to the ground state manifold (as long as the initial state has a nonzero overlap with a ground state) [71]. Since the structure of Eq. (4.16) is identical to the TDSE (4.1) except for the right-hand-side prefactor, it can be simulated in an analogous fashion by solving the equation of motion

$$\mathbf{S}(\boldsymbol{\theta}(t))\,\dot{\boldsymbol{\theta}}(t) = -\boldsymbol{F}(\boldsymbol{\theta}(t)) \tag{4.17}$$

(where t now denotes an imaginary-time argument) for a holomorphic ansatz [compare Eq. (4.7)] or the equivalent of McLachlan's VP (4.14),

$$\mathrm{Re}[\mathbf{S}(\boldsymbol{\theta}(t))]\,\dot{\boldsymbol{\theta}}(t) = -\,\mathrm{Re}[\boldsymbol{F}(\boldsymbol{\theta}(t))], \tag{4.18}$$

otherwise [32]. Solving this equation of motion using the first-order Euler scheme

$$\dot{\boldsymbol{\theta}}(t+\delta t) = \boldsymbol{\theta}(t) + \dot{\boldsymbol{\theta}}(t)\delta t \tag{4.19}$$

is equivalent to the stochastic reconfiguration (SR) optimization method commonly used in VMC [72, 186, 187]. Higher-order Runge-Kutta integrators can be used for imaginary time evolution, which sometimes can lead to an improved ground state convergence [214].

In the remainder, we will concentrate on the real-time formulation of variational time propagation, though most arguments also apply to imaginary-time propagation via the correspondence shown here.

4.2 Time-dependent variational Monte Carlo

In order to solve the variational equations of motion, it is necessary to efficiently obtain the parameter derivative $\dot{\boldsymbol{\theta}}$ from the linear equation (4.7), which requires computation of the energy gradient (4.9) and the QFM (4.8). Instead of computing the full QFM, it can also be sufficient to compute its action on a trial vector, $\boldsymbol{v} \mapsto \mathbf{S}\boldsymbol{v}$, which can be used as part of an iterative linear solver [P4].

Similar to expectation values, their gradients can also be rewritten as classical expectation values in the form of the covariance [72][17]

$$F_j(\boldsymbol{\theta}, t) = \mathrm{Cov}[\Theta_j, \tilde{H}] = \mathbb{E}[\Theta_j^*(\tilde{H} - \mathbb{E}[\tilde{H}])] \tag{4.20}$$

[17] As in Section 3.3, all formulas for $\hat{H}$ also apply to other quantum observables, not just the Hamiltonian.

of the local estimator (3.23) of $\hat{H}$ and the log-derivatives of the probability amplitude

$$\Theta_j(\boldsymbol{s}) = \frac{1}{\psi_{\boldsymbol{\theta}}(\boldsymbol{s})} \frac{\partial \psi_{\boldsymbol{\theta}}(\boldsymbol{s})}{\partial \theta_j} = \frac{\partial \ln \psi_{\boldsymbol{\theta}}(\boldsymbol{s})}{\partial \theta_j}. \tag{4.21}$$

This identity is known as the "log-derivative trick" [190] and $\boldsymbol{F}$ is sometimes called the *force vector*. Conveniently, the QFM can also be expressed as a covariance involving the same log-derivatives,

$$S_{ij}(\boldsymbol{\theta}) = \mathrm{Cov}[\Theta_i, \Theta_j] = \mathbb{E}[\Theta_i^*(\Theta_j - \mathbb{E}[\Theta_j])]. \tag{4.22}$$

These quantities can be directly used in the holomorphic TDVP (4.7). For nonholomorphic and real-parameter mappings, see Table 4.1.

Therefore, the equation of motion (4.7) can be approximated by using VMC techniques to estimate both energy gradient and QFM. After obtaining an approximate derivative $\dot{\boldsymbol{\theta}}$, the variational dynamics can again be computed using standard numerical techniques, in particular Runge-Kutta solvers. This propagation method is known as time-dependent variational Monte Carlo (t-VMC) [72, 215].

There are several sources of error that affect t-VMC propagation, some due to the nature of the variational approach, some due to the VMC sampling:

1. *Projection error*, which occurs when the variational manifold does not fully contain the direction of the exact quantum trajectory.
2. *Linear solver error* from solving the linear system of equations (4.7) for $\dot{\boldsymbol{\theta}}$ by an inexact algorithm (in particular iterative linear solvers).
3. *Local truncation error* from the discrete time steps in numerical propagation.
4. *Stochastic error* due to noise affecting the estimate of the equation of motion and, thus, the parameter updates.

Diagnosing and controlling these sources of error is crucial to ensuring stable numerical time propagation. The deterministic sources of error can be estimated using various means. For example, an estimate of the projection error can be obtained from the residual distance of the minimization (4.5) [78] and the local truncation error can be readily estimated in Runge-Kutta schemes by comparing the steps obtained from integrators of two different orders [216], which forms the basis of adaptive-step size schemes. However, such diagnostics do not take into account the stochastic noise affecting the quantities entering into the error estimates. In order to reliably use these error estimates (e.g., as part of an adaptive time propagation algorithm), a large number of Monte Carlo samples is required to suppress the stochastic noise to a sufficient degree. Due to the highly parallelizable nature of VMC sampling, this is possible in practice by distributing the computation over many CPUs or GPUs in an HPC cluster [110, P4].

Nevertheless, understanding the influence of these sources of error on the t-VMC propagation and how it can be reduced at a lower computational cost is still beneficial for optimal utilization of available resources and motivates the analyses performed in Publication [P1], which is included in the following pages.

	Case I: real	Case II: complex, holomorphic
Parameters	$\boldsymbol{x} \in \mathbb{R}^M$	$\boldsymbol{z} \in \mathbb{C}^M$
Variational mapping	$\mathbb{R}^M \to \mathcal{H}, \quad \boldsymbol{x} \mapsto \lvert\psi_{\boldsymbol{x}}\rangle$	$\mathbb{C}^M \to \mathcal{H}, \quad \boldsymbol{z} \mapsto \lvert\psi_{\boldsymbol{z}}\rangle$
Logarithmic gradient	$\Theta_j(\boldsymbol{s}) = \frac{\partial \ln \psi_{\boldsymbol{x}}(\boldsymbol{s})}{\partial x_j}$, $\boldsymbol{\Theta}(\boldsymbol{s}) \in \mathbb{C}^M$	$\Theta_j(\boldsymbol{s}) = \frac{\partial \ln \psi_{\boldsymbol{z}}(\boldsymbol{s})}{\partial z_j}$, $\boldsymbol{\Theta}(\boldsymbol{s}) \in \mathbb{C}^M$
Force vector	$F_j = \mathrm{Cov}[\Theta_j, \tilde{H}]$, $\boldsymbol{F} \in \mathbb{C}^M$	$F_j = \mathrm{Cov}[\Theta_j, \tilde{H}]$, $\boldsymbol{F} \in \mathbb{C}^M$
Energy gradient	$\frac{\partial E}{\partial x_j} = 2\,\mathrm{Re}[F_j]$, $\nabla_{\boldsymbol{x}} E \in \mathbb{R}^M$	$\frac{\partial E}{\partial z_j^*} = F_j$, $\nabla_{\boldsymbol{z}} E \in \mathbb{C}^M$
QFM	$S_{ij} = \mathrm{Cov}[\Theta_i, \Theta_j]$, $\mathbf{S} \in \mathbb{C}^{M\times M}$	$S_{ij} = \mathrm{Cov}[\Theta_i, \Theta_j]$, $\mathbf{S} \in \mathbb{C}^{M\times M}$
Equation of motion	$\sum_j \mathrm{Re}[S_{ij}]\dot{x}_j = \mathrm{Re}[-iF_i]$, $\dot{\boldsymbol{x}} \in \mathbb{R}^N$	$\sum_j S_{ij}\dot{z}_j = -iF_i$, $\dot{\boldsymbol{z}} \in \mathbb{C}^M$

	Case III: complex, nonholomorphic	
	(real-valued representation)	(complex-valued representation)
Parameters	$\boldsymbol{x} = \mathrm{Re}[\boldsymbol{z}] \oplus \mathrm{Im}[\boldsymbol{z}] \in \mathbb{R}^{2M}$	$z_j = x_j + i x_{j+M}$, $\boldsymbol{z} \in \mathbb{C}^M$
Variational mapping	$\mathbb{R}^{2M} \to \mathbb{C}^M \to \mathcal{H}, \quad \boldsymbol{x} \mapsto \boldsymbol{z} \mapsto \lvert\psi_{\boldsymbol{z}}\rangle$	$\mathbb{C}^M \to \mathcal{H}, \quad \boldsymbol{z} \mapsto \lvert\psi_{\boldsymbol{z}}\rangle$
Logarithmic gradient	$\Theta_j(\boldsymbol{s}) = \frac{\partial \ln \psi_{\boldsymbol{x}}(\boldsymbol{s})}{\partial x_j}$, $\boldsymbol{\Theta}(\boldsymbol{s}) \in \mathbb{C}^{2M}$	
Force vector	$F_j = \mathrm{Cov}[\Theta_j, \tilde{H}]$, $\boldsymbol{F} \in \mathbb{C}^{2M}$	
Energy gradient	$\frac{\partial E}{\partial x_j} = 2\,\mathrm{Re}[F_j]$, $\nabla_{\boldsymbol{x}} E \in \mathbb{R}^{2M}$	$\frac{\partial E}{\partial z_j^*} = \frac{1}{2}\left[\frac{\partial E}{\partial\,\mathrm{Re}[z_j]} + i\frac{\partial E}{\partial\,\mathrm{Im}[z_j]}\right]$
QFM	$S_{ij} = \mathrm{Cov}[\Theta_i, \Theta_j]$, $\mathbf{S} \in \mathbb{C}^{2M\times 2M}$	
Equation of motion	$\sum_j \mathrm{Re}[S_{ij}]\dot{x}_j = \mathrm{Re}[-iF_i]$, $\dot{\boldsymbol{x}} \in \mathbb{R}^{2M}$	$\dot{z}_j = \dot{x}_j + i\dot{x}_{j+M}$, $\dot{\boldsymbol{z}} \in \mathbb{C}^M$

Table 4.1 | Comparison of relevant quantities and formulas for the t-VMC implementation of McLachlan's VP between real-valued (Case I), holomorphic (Case II), and complex-valued nonholomorphic (Case III) parametrizations. In all cases, the log-probability amplitudes $\ln \psi_{\boldsymbol{\theta}}(\boldsymbol{s}) \in \mathbb{C}$ and, therefore, their gradients, as well as force vectors, and the QFM, are complex. In Case III, the complex parameters are represented here as the direct sum of real and imaginary parts in the real-valued representation to simplify notation; other orderings (such as interleaving real and imaginary part) are equivalent. See Refs. [32, 190] for details.

Publication P1

[P1] **D.H.,** Giammarco Fabiani, Johan H. Mentink, Giuseppe Carleo, Michael A. Sentef.
Role of stochastic noise and generalization error in the time propagation of neural-network quantum states.
SciPost Physics 12, 165 (2022). DOI 10.21468/SciPostPhys.12.5.165.

Summary

This article discusses the influence of different sources of error on the stability of t-VMC simulation [focusing on the holomorphic TDVP (4.7)] in a two-leg Heisenberg ladder as a model system. It had previously been shown that dynamics in the two-dimensional Heisenberg model on a square lattice can be simulated with t-VMC using an RBM ansatz [76, 79]. In contrast, stability issues had been identified as main challenges for NQS simulations in other works on quenches in the one and two-dimensional transverse-field Ising models [78, 111, 112]. In this work, we compare driven dynamics in the antiferromagnetic Heisenberg model on the square lattice and the two-leg ladder. We find the ladder system to be more sensitive to stochastic noise, requiring significantly more Monte Carlo samples or other stabilization techniques, and provide a systematic analysis of the effects of different sources of error on t-VMC propagation. To this end, we introduce a validation-set-based error diagnostic that can quantify these effects in terms of overfitting of the parameter update to stochastic noise. Using this diagnostic, we determine that, in this model system, the dynamics can be stabilized by fine-tuning the regularization strength without requiring a significant increase in Monte Carlo samples.

Publication status

This work has been published in SciPost Physics.

Declaration of contributions

I have implemented the t-VMC simulation code based on NetKet 2 [P3] (as well as the simulations with artificial noise based on full-summation code provided by G.F.), performed simulations and data analysis (with some t-VMC and full-summation results contributed by G.F.), written the initial manuscript (with input from M.A.S.), and created the figures.

Role of stochastic noise and generalization error in the time propagation of neural-network quantum states

Damian Hofmann[1⋆], Giammarco Fabiani[2†], Johan H. Mentink[2‡], Giuseppe Carleo[3∘] and Michael A. Sentef[1§]

1 Max Planck Institute for the Structure and Dynamics of Matter, Center for Free-Electron Laser Science (CFEL), Luruper Chaussee 149, 22761 Hamburg, Germany
2 Radboud University, Institute for Molecules and Materials, Heyendaalseweg 135, 6525 AJ Nijmegen, The Netherlands
3 Institute of Physics, École Polytechnique Fédérale de Lausanne (EPFL), 1015 Lausanne, Switzerland

⋆ damian.hofmann@mpsd.mpg.de , † g.fabiani@science.ru.nl , ‡ j.mentink@science.ru.nl , ∘ giuseppe.carleo@epfl.ch , § michael.sentef@mpsd.mpg.de

Abstract

Neural-network quantum states (NQS) have been shown to be a suitable variational ansatz to simulate out-of-equilibrium dynamics in two-dimensional systems using time-dependent variational Monte Carlo (t-VMC). In particular, stable and accurate time propagation over long time scales has been observed in the square-lattice Heisenberg model using the Restricted Boltzmann machine architecture. However, achieving similar performance in other systems has proven to be more challenging. In this article, we focus on the two-leg Heisenberg ladder driven out of equilibrium by a pulsed excitation as a benchmark system. We demonstrate that unmitigated noise is strongly amplified by the nonlinear equations of motion for the network parameters, which causes numerical instabilities in the time evolution. As a consequence, the achievable accuracy of the simulated dynamics is a result of the interplay between network expressiveness and measures required to remedy these instabilities. We show that stability can be greatly improved by appropriate choice of regularization. This is particularly useful as tuning of the regularization typically imposes no additional computational cost. Inspired by machine learning practice, we propose a validation-set based diagnostic tool to help determining optimal regularization hyperparameters for t-VMC based propagation schemes. For our benchmark, we show that stable and accurate time propagation can be achieved in regimes of sufficiently regularized variational dynamics.

Received 07-05-2021
Accepted 29-04-2022
Published 18-05-2022
doi:10.21468/SciPostPhys.12.5.165

Contents

1 Introduction

In recent times, the application of machine learning methods to problems in quantum physics has received considerable interest [1]. Examples include the use of neural networks for quantum state reconstruction [2], quantum control [3] and feedback [4], as well as classifying phases of matter [5–7]. Due to their success in approximating high-dimensional nonlinear functions in machine learning applications, neural networks were proposed in 2017 as a variational ansatz for the quantum wave function [8]. These neural-network quantum states (NQS) have been applied to a wide variety of problems in quantum many-body physics, including spin [8–23], bosonic [11, 24] and fermionic [25, 26] systems, as well as quantum computation [27, 28] and dissipative systems [29–32]. One particular research area where NQS could prove important in the near future are non-equilibrium quantum many-body problems, which are of interest across research fields, reaching from quantum simulators with cold atoms and trapped ions [33, 34] via arrays of Rydberg atoms [35], and photonic platforms [36] to laser-driven quantum materials [37]. The theoretical investigation of such scenarios is restricted by a lack of computational methods that allow researchers to reliably simulate driven correlated systems, in particular in two dimensions. NQS provide a promising candidate wave function for the purpose of investigating out-of-equilibrium dynamics, in part due to their ability to capture high-entanglement [38] and topological states of matter [9, 10], which may serve to complement other approaches, in particular those based on tensor network states [39].

Typically, NQS are time propagated using time-dependent variational Monte Carlo (t-VMC) [8, 40, 41]. So far, this has been studied in the literature primarily in the context of quenches in the spin-1/2 Ising and Heisenberg models in both one and two dimensions [42–47]. In many cases, achieving numerical stability has been identified as the key challenge for the reliable simulation of quantum dynamics [42–44] and also for ground state optimization using imaginary time propagation [19]. In contrast, the capability of the NQS ansatz to represent the relevant dynamical quantum states was not found to be a limiting factor. However, the general question which types of states can be represented well by a given network architecture and the scaling of the required network size is still a matter of active research [20, 23].

In this work, we are concerned with understanding and separating different sources of instabilities that can prevent t-VMC time propagation to reach dynamical states even when they can in principle be captured by the variational ansatz. To this end, we take a closer look at dynamics in the antiferromagnetic 2D Heisenberg model, which has previously been studied with t-VMC on a 2D square lattice geometry. There, stable time propagation has been demonstrated using the well-established restricted Boltzmann machine (RBM) architecture [45–47]. We compare this setting to the same Hamiltonian on a two-leg ($L \times 2$) ladder geometry, which features significantly more complex quantum dynamics and which we find to be much more sensitive to numerical instabilities. This is true already for very small systems which are still accessible by exact numerical time evolution (exact diagonalization, ED) which provides us with reliable data to benchmark the RBM dynamics.

We demonstrate that the observed instabilities arise primarily as a result of stochastic error, which is amplified through the generally ill-conditioned variational equations of motion. The stability of the propagation can be improved by reducing noise through means such as increasing the number of Monte Carlo samples or reducing the simulation time step, but this comes at the cost of increasing the required computational resources. However, we show that regularization of the equation of motion also helps to mitigate the effects of noise without significant additional computational cost and highlight the strong effect of regularization parameters on the quality of the resulting trajectory. Taking inspiration from machine learning terminology, this effect can be described as overfitting to stochastic noise, which leads to poor reliability of the time stepping procedure. As in machine learning, this type of overfitting can be detected and quantified by validating the optimized time step on independently sampled data, leading us to introduce a validation-set variational error and show that it can help identify unstable regimes and thus optimize regularization hyperparameters.

The main contributions of this work are therefore (i) presenting the two-leg Heisenberg model as a particularly challenging system to simulate using NQS with t-VMC, making it a useful benchmark case; (ii) the analysis of different sources of error and the effect of regularization on t-VMC propagation in this model; and (iii) the introduction of a validation-set error for quantifying error due to overfitting to stochastic noise.

This article is structured as follows: In Section 2 we define the driven Heisenberg model used as reference in the rest of this paper, in Section 3 we show the influence of regularization on stability and accuracy of the NQS dynamics, and in Section 4 we discuss how this can be quantified using a validation-set approach. Finally, in Section 5, we conclude and provide an outlook on future work.

2 Model and methods

We study excitations in the two-dimensional antiferromagnetic (AFM) Heisenberg model (working in units with $\hbar = 1$),

$$\hat{H}(t) = J_0 \sum_{\{i,j\}\in\mathcal{N}} \hat{h}_{ij} + J_0\,\Delta_x(t) \sum_{\{i,j\}\in\mathcal{N}_x} \hat{h}_{ij}\,, \tag{1}$$

where

$$\hat{h}_{ij} = \sum_{\mu=1}^{3} \hat{\sigma}_i^{\mu}\hat{\sigma}_j^{\mu} \tag{2}$$

denotes the local Heisenberg coupling acting on each bond with the Pauli matrices $\hat{\sigma}_i^{\mu}$, $\mu \in \{1,2,3\}$, and exchange coupling strength $J_0 > 0$. Here, the outer sum runs over

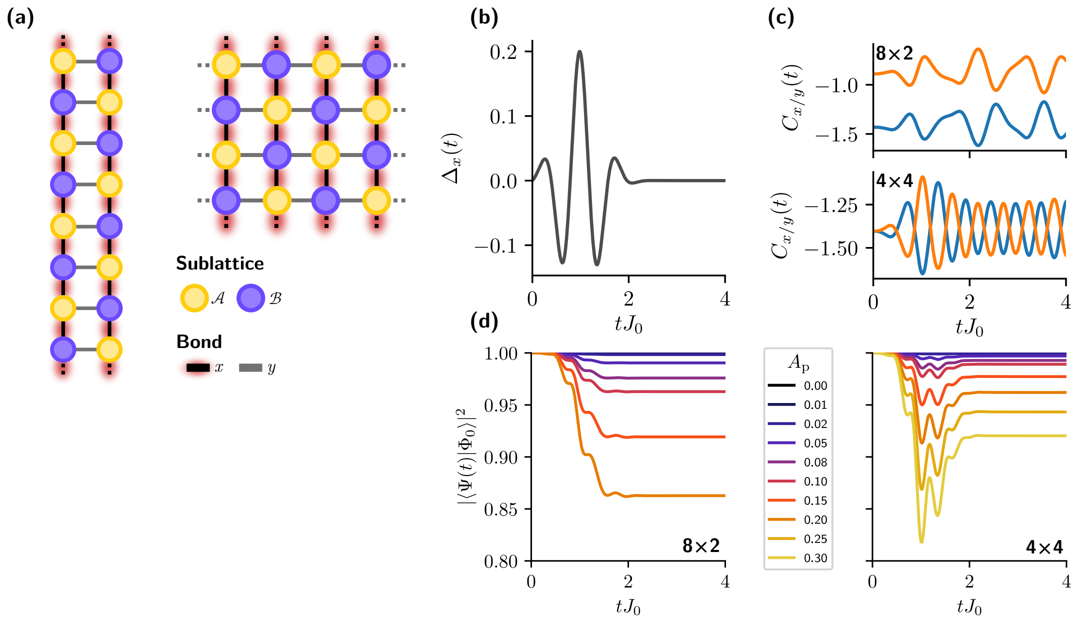


Figure 1: **(a)** Ladder and square lattice geometry for the 2D AFM Heisenberg model. The colors indicate the $\mathcal{A}$ and $\mathcal{B}$ sublattices of the bipartite model. In the ladder geometry, periodic boundary conditions (PBC) in the x direction are imposed. In the square geometry, PBC are imposed in both directions. **(b)** The Heisenberg system is driven out of its ground state by modulating the x-bond coupling in a single pulse with shape $\Delta_x(t)$ [Eq. (3)], here displayed for $A_p = 0.20$, $t_p = 0.987J_0^{-1}$, $\omega_p = 8.0J_0$, and $\sigma_p = 0.4J_0^{-2}$. **(c)** Oscillations of the x and y bond spin-spin correlations $C_{x/y}(t)$ [Eq. (4)] caused by the pulse for both geometries as computed from the exact time-evolved state. **(d)** Overlap of the initial state $|\Phi_0\rangle$ obtained from ED with the exact time-evolved state $|\Psi(t)\rangle = \hat{U}(t)|\Phi_0\rangle$ for varying pulse amplitude A_p and both geometries. The other pulse parameters are the same as in panel (b).

the nearest-neighbor bonds $\mathcal{N}$ of a finite-dimensional rectangular lattice $\mathfrak{L}$ of size $N = L_x \times L_y$ [Fig. 1(a)] and $\mathcal{N}_{x/y} \subseteq \mathcal{N}$ denotes subset of x/y bonds. We will consider two different lattice geometries: the square lattice with side length $L := L_x = L_y$ and the ladder geometry with $L := L_x$, $L_y = 2$ sites. In both cases, periodic boundary conditions in x direction are assumed. For the square lattice, we also impose periodic boundary conditions in the y direction.

Starting from the ground state at $t = 0$, we study the time evolution of the system under an excitation created by a pulsed modulation of the exchange coupling along the x direction of the lattice (which is the long direction in the ladder system), which has the form [Fig. 1(b)]

$$\Delta_x(t) = A_p \sin(\omega_p t) \exp\left(-\frac{(t-t_p)^2}{2\sigma_p}\right), \qquad \text{for } t \geq 0 . \tag{3}$$

This driving is physically motivated and can be viewed as a single-cycle THz pulse polarized along x which drives the exchange coupling through a Raman process [45,48]. The y direction coupling is kept constant. In addition to the energy, we compute the average nearest-neighbor bond correlation

$$C_\nu(t) = \frac{1}{N} \sum_{\{i,j\}\in\mathcal{N}_\nu} \sum_{\mu=1}^{3} \langle \hat{\sigma}_i^\mu \hat{\sigma}_j^\mu \rangle , \tag{4}$$

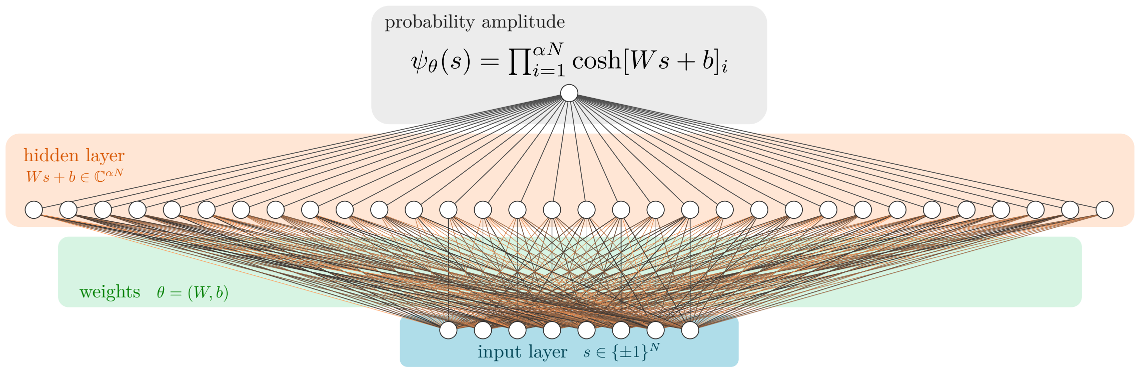

probability amplitude
$\psi_\theta(s) = \prod_{i=1}^{\alpha N} \cosh[Ws+b]_i$
hidden layer
$Ws + b \in \mathbb{C}^{\alpha N}$
weights $\theta = (W, b)$
input layer $s \in \{\pm 1\}^N$

Figure 2: Restricted Boltzmann machine architecture used as a variational quantum state with N visible units corresponding to the lattice size and a hidden unit density α. A detailed description of the ansatz is given in Appendix A.

along the $\nu = x, y$ direction as an observable. To obtain reference data, we have simulated the time evolution under this pulse through exact (ED) propagation. The resulting dynamics are shown in [Fig. 1(c)]. In both systems, the pulse causes oscillations that persist after the pulse. However, while our driving protocol causes only singlet excitations on the square lattice, the ladder model exhibits both singlet and triplet excitations [49–51] and we indeed observe more complex and irregular dynamics in the time-dependent bond correlations. For equal amplitude A_p of the x-bond modulation, the ladder system is more strongly affected as evidenced by the higher distance to the initial state [Fig. 1(d)].

As a variational ansatz, we employ the restricted Boltzmann machine (RBM) with complex-valued weights $\theta \in \mathbb{C}^M$ (Fig. 2) as a parametrization of the quantum wave function $\ln \psi_\theta(s)$ mapping basis spin configurations to the corresponding log-probability amplitudes. The translation symmetries of the lattice are enforced in the manner described in Refs. [8, 45], which reduce the number of variational parameters to $M = \alpha(N+1)$, where α is the hidden unit density. The translation group of an $N = L \times L$ lattice with periodic boundary conditions contains N distinct operations. In the ladder geometry, the notion of periodic boundary conditions only applies to the long direction; however, we include the reflection symmetry along the short direction, so the total order of the translation symmetries is still N. The network architecture is fully described in Appendix A.

The time propagation is done using time-dependent variational Monte Carlo (t-VMC) [8, 40], which corresponds to numerically solving the equation of motion of the time-dependent variational principle (TDVP)

$$S(\theta(t))\,\dot{\theta} = -\mathrm{i}F(\theta(t), t), \tag{5}$$

where $\dot{\theta} = \mathrm{d}\theta(t)/\mathrm{d}t$, using a stochastic estimate of the quantum Fisher matrix (QFM)

$$S_{ij}(\theta) = \mathbb{E}[\Theta_i^* \Theta_j] - \mathbb{E}[\Theta_i^*]\mathbb{E}[\Theta_j], \tag{6}$$

and energy gradient

$$F_i(\theta, t) = \mathbb{E}[\Theta_i^* \mathfrak{H}(t)] - \mathbb{E}[\Theta_i^*]\mathbb{E}[\mathfrak{H}(t)], \tag{7}$$

with log-probability derivatives $\Theta_i(s) = \partial_i \ln \psi_\theta(s)$ and local energy $\mathfrak{H}(t)(s) = \frac{\langle s|\hat{H}(t)|\psi_\theta\rangle}{\langle s|\psi_\theta\rangle}$. The expectation values $\mathbb{E}[\,\cdot\,]$ are taken with respect to the Born probability distribution $\sim |\psi_\theta(\,\cdot\,)|^2$. Further details on the t-VMC propagation scheme are provided in Appendix B. The initial ground state is prepared by minimizing the energy of a randomly initialized RBM using stochastic reconfiguration [41].

3 Stability and regularization

In this section, we will highlight jump-like numerical instabilities that we find to arise primarily due to stochastic noise from VMC sampling that enters into the nonlinear equation of motion (5), leading to missteps where the simulation diverges from the physical trajectory in an irrecoverable fashion. We will then show how regularization of the equation of motion can stabilize the dynamics without requiring a change in time step or an increase in Monte Carlo samples.

3.1 Numerical instabilities from unmitigated noise

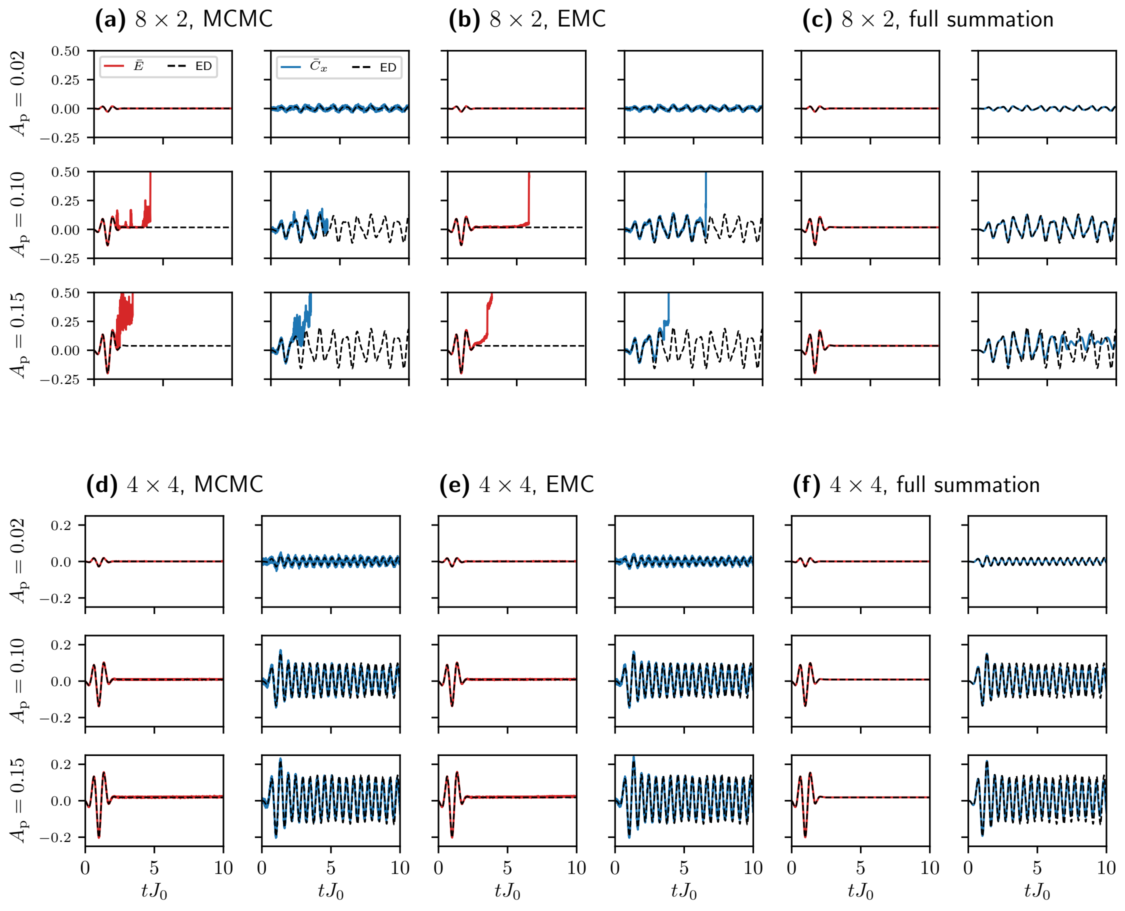


Figure 3: Time-dependent per-site energy change $\bar{E}(t) = [\langle \hat{H}(t) \rangle - E(0)]/N$ and average x-bond correlation $\bar{C}_x(t) = C_x(t) - C_x(0)$ for the 8×2 ladder [panels (a)–(c)] and 4×4 square geometry [panels (d)–(f)] and varying pulse strengths A_p. The trajectories have been obtained from t-VMC evolution using MCMC [panels (a),(d)] and EMC [panels (b),(e)] sampling as well as results based on full summation of the equations of motion [panels (c),(f)]; see Sect. 3.1 for details. The dashed lines show ED results for reference. In all cases, a symmetric RBM with hidden unit density of $\alpha = 10$ has been used. The initial state is the approximate ground state of the respective system obtained by stochastic reconfiguration and is the same for panels (a)–(c) and (d)–(f), respectively. In all cases, the equation of motion is evaluated by singular-value decomposition of S and applying a diagonal shift of $\epsilon = 10^{-3}$ (see Sect. 3.2 and Appendix D) for regularization.

We first highlight the practical challenge posed by the highly nonlinear and stochastic t-VMC equation of motion, by demonstrating how a change in lattice geometry of an otherwise unaltered physical model can affect the stability of the NQS propagation.

In previous works [45–47], it has been shown that the dynamics of the Heisenberg model on a square lattice can indeed be successfully simulated using t-VMC with RBM quantum states. We obtain equivalent results for our pulsed driving [Fig. 3(d)–(f)]. The main manifestation of the error as compared to the exact dynamics is a continuous decay of amplitude of the resulting oscillations, which is visible from the averaged nearest-neighbor correlation $C_x(t)$. Increasing the width of the network, i.e., the hidden unit density α, both improves the accuracy of the initial (ground) state and reduces the loss of accuracy over the course of the time evolution (data not shown). This shows that the decay is the result of accumulated TDVP error and can be reduced by an increase in network size, which is in agreement with results for a square pulse excitation presented in Ref. [45]. However, this behavior is markedly different for the ladder geometry (with otherwise unchanged system parameters), where instabilities quickly occur during t-VMC evolution already for weak pulse strengths $A_\mathrm{p} \geqslant 0.02$ [Fig. 3(a),(b)]. Notably, the observed instabilities violate energy conservation, a property that is inherent to the TDVP equation of motion for a static Hamiltonian. Therefore, their origin has to be numerical. In order to better understand which sources of error in the t-VMC method contribute to these instabilities, we compare three different propagation schemes:

1. t-VMC propagation where the components of the equation of motion are estimated stochastically using Markov chain Monte Carlo (MCMC) with the Metropolis algorithm, which is the standard propagation method for NQS [8, 41] [Fig. 3(a),(d)];

2. autocorrelation-free "exact" Monte Carlo (EMC), where samples are directly drawn according to the Born distribution $|\psi(\cdot)|^2$ without using the Metropolis algorithm [Fig. 3(b),(e)]; and

3. time propagation based on full summation of the t-VMC equation of motion over all spin configurations, which provides a reference free of stochastic noise [Fig. 3(c),(f)].

In the Metropolis MCMC scheme, updates to the spin configuration are proposed based on exchanges of spin pairs which preserve the total magnetization and thus the restriction of the ansatz to the zero-magnetization sector. The EMC scheme can only be applied to small systems accessible to ED, because it relies on the knowledge of the full Born distribution. Here, it is used strictly as a benchmark to uncover the influence of noise on the dynamics while ruling out errors due to non-convergence of the Metropolis sampling[1]. The full summation scheme is similarly limited to small systems with tractable Hilbert space. We have used a second-order Runge-Kutta method (Heun's method) for time propagation in all cases, using two evaluations of the equation of motion (5) per time step, a fixed step size of $\delta t = 0.002$ and $N_\mathrm{s} = 7000$ Monte Carlo samples for EMC and t-VMC. Here and in all other MCMC simulations presented in this work, a number of Monte Carlo steps equal to the system size N is performed between each of the N_s samples included in the chain in order to reduce the autocorrelation between successive samples. While there is a visibly increased level of noise with the MCMC sampling, divergences occur at similar time points of the evolution for both approaches. In the full summation results at the same driving strengths, the energy jumps are absent and the dynamics are more accurately reproduced on the ladder with pulses $A_\mathrm{p} \leq 0.10$ [Fig. 3(c),(f)]. We note that even in the absence of stochastic noise, the time evolution shown here fails to accurately capture the ladder dynamics for stronger excitations, as can be seen from the

[1]Note that autocorrelation-free Monte Carlo sampling is practically possible beyond the ED regime for NQS architectures based on autoregressive networks [52], though this is quite different from the benchmark implementation considered here.

$A_\mathrm{p} = 0.15$ trajectory. Furthermore, we note that the likelihood of instabilities can be reduced by lowering the integrator time step, which, however, increases the computational cost of the simulation.

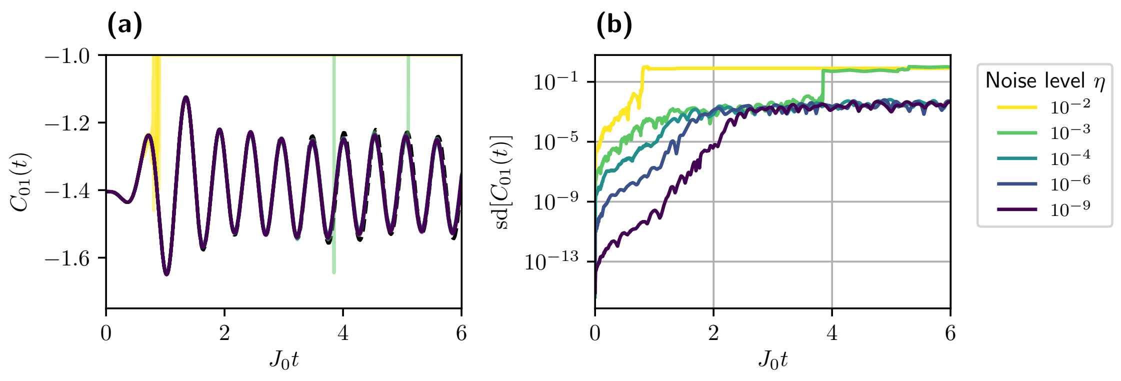


Figure 4: **(a)** Time evolution of the nearest-neighbor coupling term $C_{01} = \sum_{\mu=1}^{3} \langle \hat{\sigma}_0^\mu \hat{\sigma}_1^\mu \rangle$ using the TDVP equation of motion evaluated by full summation with added artificial noise [Eq. 8]. We show results for the dynamics on the 4×4 lattice with pulse strength $A_\mathrm{p} = 0.20$ for an RBM with hidden unit density $\alpha = 10$. For each value of the noise strength η, five independent trajectories are shown as faint lines. The opaque lines indicate the median of the respective curves. **(b)** Empirical standard deviation between the set of curves in panel (a) grouped by noise strength η.

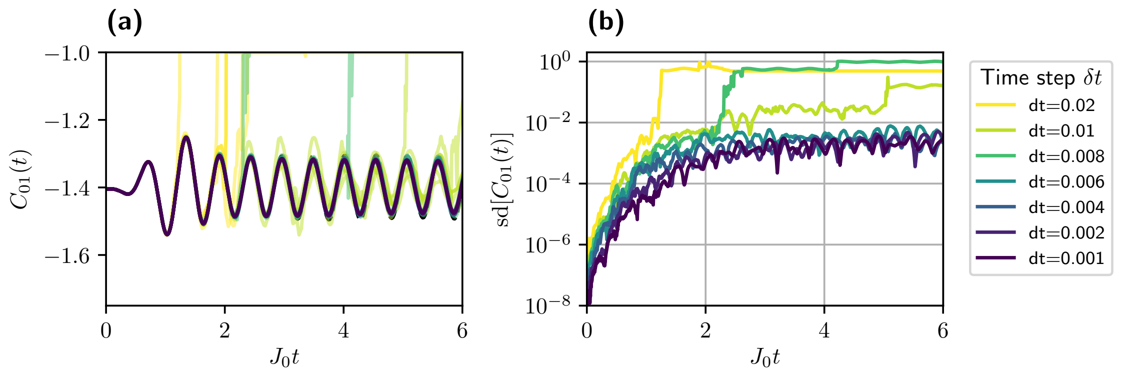


Figure 5: **(a),(b)** Same trajectories as in Fig. 4[(a),(b)] for a driving strength of $A_\mathrm{p} = 0.10$ and with varying time step δt at a fixed noise level of $\eta = 10^{-3}$.

In any case, the occurrence of jump-type instabilities is significantly more likely in the presence of Monte Carlo noise. This is true both for MCMC and EMC, showing that the instabilities are not just a result of failed convergence of the Markov chain sampling. Indeed, a noisy energy gradient alone is sufficient to cause the observed divergences when combined with the t-VMC equation of motion for NQS. We show this in an idealized picture as follows: Consider Eq. (5) without any stochastic sampling but with an artificial term of proportional Gaussian white noise added to the energy gradient[2]. This provides a direct way to control the noise

[2]We note that the proportional Gaussian noise model of Eq. (8) is indeed a simplification. An analysis in Ref. [44] shows that the actual noise level varies between components of the energy gradient and depends on the quantum geometry of the ansatz (through the QFM spectrum) as well as the energy fluctuations. However, already

level without affecting other steps in the propagation. Specifically, we solve

$$S\dot{\theta} = -\mathrm{i}(F + \xi)\,, \tag{8}$$

where ξ is a random vector with components drawn from a complex normal distribution with mean $\mathbb{E}[\xi_i] = 0$ and variance $\mathrm{Var}[\xi_i] = |\eta F_i|^2$. The parameter η determines the relative noise strength and the standard error in each component is proportional to $|F_i|$. The equation of motion is solved using the same second-order Heun scheme used for the t-VMC simulation and at a fixed time step of $\delta t = 0.002$. In Fig. 4(a) we show resulting trajectories for varying noise levels η. For each value of η, five independent trajectories are shown. In the absence of instabilities, the standard deviation of the trajectories at first grows with increasing noise [Fig. 4(b)]. After a time of the order of the pulse length, the spread of the trajectories stabilizes at a value that is independent of η. For $\eta \geq 10^{-3}$, jump instabilities occur within the simulation time. Whereas for $\eta = 10^{-2}$ all trajectories show this instability already around $t = 1J_0^{-1}$, the jumps happen more sporadically and at later times for $\eta = 10^{-3}$, with only two of the trajectories exhibiting a jump before $t = 6J_0^{-1}$. Reducing the integrator time step decreases the frequency of instabilities in a similar fashion, as is shown in Fig. 5.

This idealized experiment shows that random noise in the gradient can be amplified through Eq. 5. Together with the highly non-linear nature of the phase space of the NQS ansatz (compare Ref. [19]), this can cause jump-type instabilities like we have observed in the t-VMC propagation, instead of a gradually increasing spread of the trajectories that would be expected for a more regular ansatz and equation of motion. Both a reduction in noise level and time step reduce the likelihood of instabilities.

In agreement with observations made for other systems [19,44], we find that the expressive capabilities of the network are not a limiting factor. For individual time points, the exact quantum state shown in Fig. 3 can indeed be represented to good accuracy by an RBM of width $\alpha = 10$. See Appendix C for detailed results.

Beyond the data shown here, we have observed RBM states with fewer parameters to be generally more stable. This, however, comes at the cost of decreased accuracy over time, as representational error accumulates (cf. Ref. [45]). The numerical instability present in larger network thus counteracts the benefits of increased expressiveness, making it particularly important to find ways of alleviating this effect without significant increase in computational cost. We further note that while the ladder system is particularly sensitive to the types of instabilities discussed here, they can also occur in the square lattice geometry. We have observed this both in the artificial noise model (Fig. 4) and for a driving strength increased beyond $A_\mathrm{p} = 0.30$ (data not shown). Therefore, while the Heisenberg ladder is a very suitable benchmark system for these types of numerical issues, the observations made here can be expected to apply to a broader class of systems, especially when they are driven far out of equilibrium on short time scales.

In the next section, we will discuss how the choice of regularization scheme affects the stability of the dynamics.

3.2 Influence of regularization

The formal solution of the TDVP equation (5) is given by

$$\dot{\theta} = -\mathrm{i}S^{+}F(t)\,, \tag{9}$$

where S^{+} denotes the Moore-Penrose pseudoinverse of the QFM [8, 42, 44, 46]. Computing the pseudoinverse can be done by singular-value decomposition (SVD), which is equivalent

the simple proportional model used here does exhibit the jump-like instabilities when subjected to noise amplified through the equation of motion.

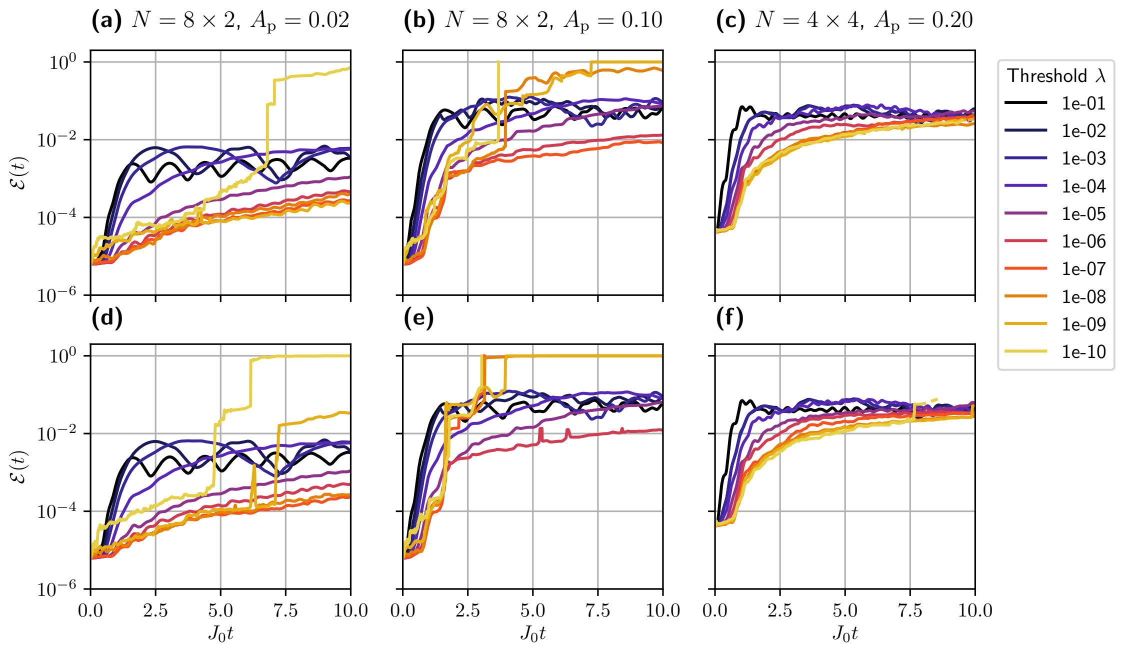


Figure 6: Infidelity of the time-evolved variational state compared to the exact trajectory for varying regularization (in the form of the SVD threshold λ). The trajectories have been computed using t-VMC with EMC [panels (a)–(c)] and Metropolis [panels (d)–(f)] sampling. The columns correspond to a weak excitation $A_\mathrm{p} = 0.02$ [panels (a),(d)] and a moderate excitation $A_\mathrm{p} = 0.10$ [panels (b),(e)] in the 8×2 ladder, as well as a stronger excitation $A_\mathrm{p} = 0.20$ in the 4×4 square lattice [panels (c),(f)]. The initial $\mathcal{E}(t = 0)$ of order 10^{-5} is the approximation error of the variational ground state. The time propagation has been computed using t-VMC with EMC sampling with $N_\mathrm{s} = 24000$ samples for the ladder [panels (a),(b)] and $N_\mathrm{s} = 11200$ for the square lattice [panel (c)]. We have used the symmetrized RBM ansatz with a hidden unit density of $\alpha = 10$.

to the eigendecomposition in this case. This is because S is a covariance matrix and therefore positive semi-definite, i.e., all eigenvalues are nonnegative. Then, in the eigenbasis of $S = V\,\mathrm{diag}(\{\zeta_j\}_{j=1}^M)V^\dagger$, the TDVP equation reduces to

$$\zeta_j [V^\dagger \dot{\theta}]_j = -\mathrm{i}[V^\dagger F]_j\,. \tag{10}$$

We order the eigenvalues of S by magnitude $\zeta_1 \geq \zeta_2 \ldots \geq \zeta_M$ in the following and denote the smallest nonzero eigenvalue by ζ_r. Then, for all nonzero eigenvalues corresponding to $j \leq r$, we have $[V^\dagger \dot{\theta}]_j = -\mathrm{i}\zeta_j^{-1}[V^\dagger F]_j$. The directions in the null-space of S do not contribute to the physical dynamics; changes of θ in those directions only affect gauge degrees of freedom of the quantum state. In order to obtain the minimum-norm solution, these are set to zero, i.e., $[V^\dagger \dot{\theta}]_j = 0$ for $j > r$.

In practice, the numerical solution of this equation is complicated by the fact that NQS typically possess a non-exponential but still large number of variational parameters compared to more traditional variational wave functions and further allow for redundancy in the parametrization of a specific quantum state. As a consequence, the QFM is singular, and the nonzero part of its spectrum typically spans many orders of magnitude [43,44,53]. Therefore, the linear system (5) has a high condition number $\kappa(S) = \zeta_1/\zeta_r$ which, in particular, means that small perturbations in the right-hand side $F(t)$ can be strongly amplified in the solution $\dot{\theta}$

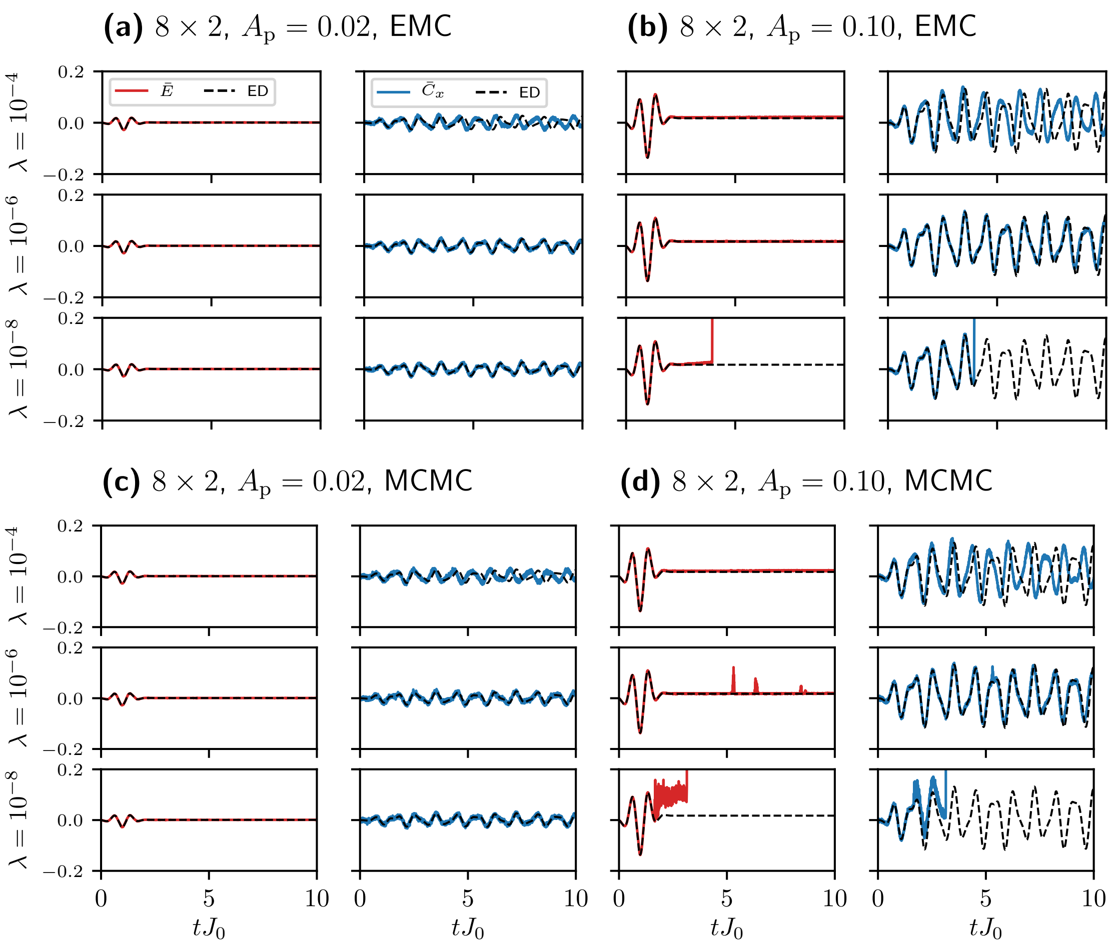


Figure 7: Relative change in energy and x-bond correlation as defined in Fig. 3 for a selection of trajectories shown in Fig. 6 with different regularization strengths λ.

of the equation of motion (EOM), causing the jump instabilities we have empirically observed above. For this reason, it is necessary to regularize the EOM in order to stabilize the dynamics while preserving the physical accuracy of the resulting trajectory. Typically, this is done by truncating eigenvalues below a threshold λ in Eq. (9). Specifically, ζ_i is treated as zero when $\zeta_i \leq \lambda\zeta_1$. The effective condition number of S is then bounded by $\kappa \leq \lambda^{-1}$. While we focus on this regularization scheme in the remainder, our analysis also applies to a broader class of regularization schemes. In particular, the application of a diagonal shift to S and a signal-to-noise ratio based regularization scheme proposed in Ref. [44] are briefly discussed in Appendix D.

Naturally, there is a trade-off between stability and accuracy: Too much regularization will suppress crucial parts of the physical dynamics, while the system is susceptible to instabilities without or with only weak regularization. This can be seen in Fig. 6, where we show the infidelity of the time-dependent quantum state relative to the ED time evolution of the system,

$$\mathcal{E}(t) = 1 - \frac{|\langle\Psi_{\mathrm{ED}}(t)|\psi_\theta(t)\rangle|^2}{\langle\Psi_{\mathrm{ED}}(t)|\Psi_{\mathrm{ED}}\rangle\langle\psi_\theta(t)|\psi_\theta(t)\rangle}, \tag{11}$$

for varying regularization strength λ. Specifically for the weak pulse [Fig. 6(a)], we can clearly see a separation of three regimes: an over-regularized regime (for $\lambda \geq 10^{-4}$), where the dynamics are stable but inaccurate; an intermediate stable regime where the physical observables are accurate and the regularization still sufficient to stabilize the dynamics; and an unstable regime ($\lambda \leq 10^{-8}$) where jump instabilities occur within the simulation time frame. While for the weak pulse the stable regime spans several orders of magnitude, it becomes smaller with increasing strength [Fig. 6(b)]. By contrast, on the square lattice the dynamics remain stable and largely unaffected by the regularization strength over a wide range of λ even at a pulse strength of $A_\text{p} = 0.20$ [Fig. 6(c)]. These results have been obtained with EMC sampling, but the same behavior can be observed for the practically relevant case of Metropolis sampling, which is shown in Fig. 6[(d)–(f)]. In this case, the stable regime of regularization becomes smaller for the ladder system. Figure 7 shows the expectation values of energy and bond correlations for several trajectories. These observables show how in the over-regularized regime, numerical stability comes at the cost of physical errors which manifest here in an incorrect reproduction of the oscillation frequencies. At the same time, the square lattice system is almost unaffected by Metropolis sampling, except at very low thresholds. Notably, for both EMC and MCMC sampling strategies, the dynamics converge to a stable trajectory at a much lower number of Monte Carlo samples than in the the ladder system. This hints at an increased sampling complexity of the low-lying ladder excitations in accordance with the higher physical complexity of the ladder excitations, an observation which is corroborated by our results in the next section.

Altogether, our results highlight the delicate balance between stability and accuracy of the dynamics in the presence of stochastic noise and the resulting necessity to fine-tune regularization hyperparameters to reach the optimal regime. From the comparison of ladder and square geometry, we have further seen that the extent of this behavior depends strongly on the details of the system.

4 Overfitting to noise and validation error

In order to choose an optimal regularization for a given system and excitation scheme, it is important to have access to appropriate diagnostics. While the error relative to ED as shown in the previous section provides a straightforward way to assess the quality of the solution, this option is restricted to small benchmark systems. Here, we therefore propose an alternative diagnostic which is more generally applicable.

The local truncation error resulting from a single time step in the variational approximation is quantified by the TDVP error [8, 44]

$$r^2(t) = \left[\frac{D(\psi[\theta(t) + \dot{\theta}\,\delta t], \hat{U}_{t+\delta t,t}\psi[\theta(t)])}{D(\psi[\theta(t)], \hat{U}_{t+\delta t,t}\psi[\theta(t)])} \right]^2 . \tag{12}$$

Here, $D(\cdot,\cdot)$ denotes the Fubini-Study distance and $\hat{U}_{t',t}$ is the unitary time evolution operator from t to t'. The equation of motion (5) can be derived by locally minimizing the numerator of Eq. (12). The denominator provides a rescaling of the error to account for the varying exact distance between points along the trajectory. This quantity can be estimated to second order in δt as [44]

$$r^2(\dot{\theta}; S, F, \delta E) = 1 + \frac{\dot{\theta}^\dagger(S\dot{\theta} + \mathrm{i}F) - \mathrm{i}F^\dagger\dot{\theta}}{(\delta E)^2} , \tag{13}$$

where $(\delta E)^2 = \mathrm{Var}[\hat{H}(t)]$ and the other quantities are defined as in Eq. (5).

While capturing loss of accuracy due to the variational approximation, the TDVP error does not account for effects caused by the stochastic noise affecting the equation of motion and thus its solution. In order to account for this additional source of errors, we take inspiration from a standard practice of machine learning: the use of a so called validation error to detect failure to

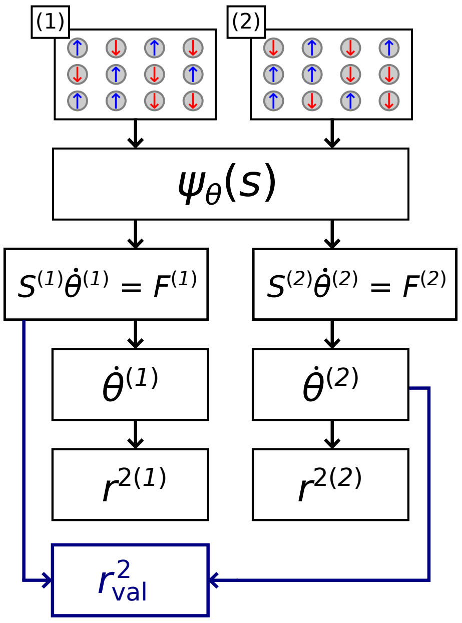


Figure 8: Illustration of the process for computing the validation TDVP error as described in Sect. 4. Two sets of spin configurations are independently generated via Monte Carlo sampling and used to obtain two independent derivatives through solving the equation of motion. The validation error (15) is then computed as the error of the second update with respect to the first equation of motion.

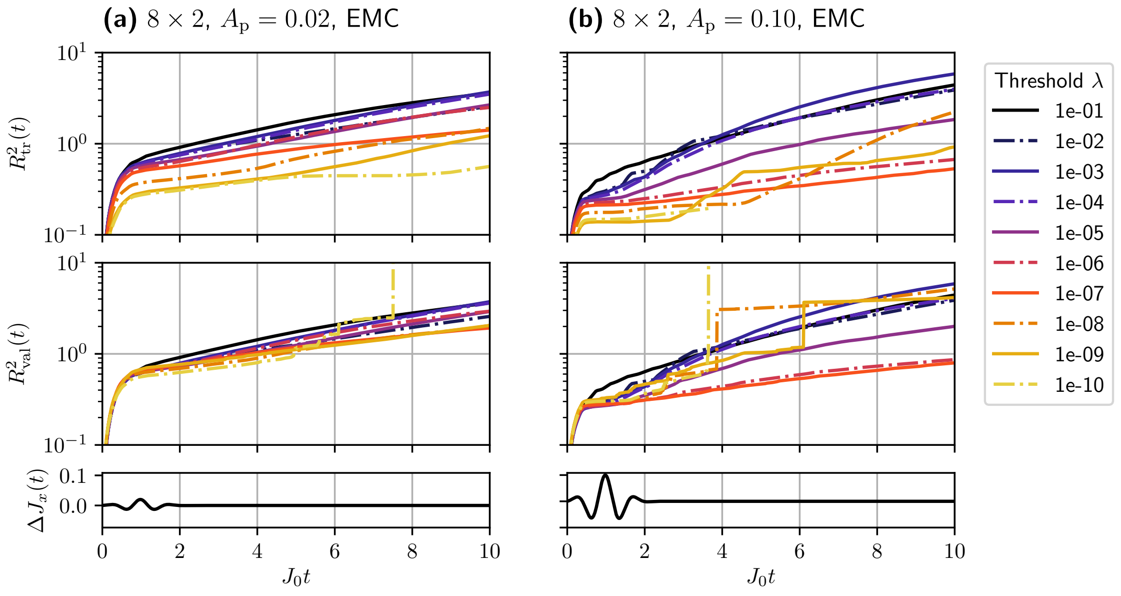


Figure 9: Integrated bare and validation TDVP error (16) over time with varying regularization (SVD threshold λ) for the same trajectories as shown in Fig. 6(a),(b), i.e., using EMC sampling and with pulse strengths of **(a)** $A_p = 0.02$ and **(b)** $A_p = 0.10$. The bottom panels show the time-dependent modulation $\Delta J_x(t)$ for reference.

generalize beyond the training data caused by overfitting to a specific sample [54]. Adapted to our present purpose, we consider the specific realization of spin configurations used to estimate the EOM as the training set. Solely optimizing the parameter update for this realization bears the risk of overfitting, in which case the solution may be optimal only on the training set but performs badly on independent estimates of the same EOM. In order to detect this, we can compute two updates $\dot\theta^{(i)}$, $i = 1, 2$, from independently drawn samples (separately estimating $S^{(i)}$, $F^{(i)}$ for both). While the resulting $\dot\theta^{(i)}$ and corresponding error estimates

$$r^2_{\mathrm{tr},i} = r^2(\dot\theta^{(i)}; S^{(i)}, F^{(i)}, \delta E^{(i)}) \tag{14}$$

are identically distributed, the error of the update $\dot\theta^{(2)}$ with respect to the independently estimated equation of motion $S^{(1)}\dot\theta = -\mathrm{i}F^{(1)}$ can be used to quantify the generalization properties of the parameter derivative. This procedure is illustrated in Fig. 8. Specifically, we define the validation TDVP error as

$$r^2_{\mathrm{val}} = r^2(\dot\theta^{(2)}; S^{(1)}, F^{(1)}, \delta E^{(1)}) . \tag{15}$$

Crucially, r^2_{val} can be estimated using only quantities that are accessible as part of the t-VMC computation. This makes it feasible to use the validation error as a diagnostic for the degree of overfitting and thus reliability of the TDVP solution in systems where a comparison to ED data is no longer possible. If the solution of the EOM is deterministic, we have $\dot\theta^{(1)} = \dot\theta^{(2)}$ and consequently $r^2_{\mathrm{val}} = r^2_{\mathrm{tr},i}$. Otherwise, the validation error will be larger, indicating the amount of "overfitting" of the update $\dot\theta^{(1)}$ to noise present in the sample. We note that the error estimates are themselves affected by noise in both EOM and energy variance. This is alleviated by considering the integral

$$R^2_{\mathrm{tr/val}}(t) = \int_0^t r^2_{\mathrm{tr/val}}(t')\,\mathrm{d}t' , \tag{16}$$

or, if the local quantity is needed, by averaging over additional realizations of r^2_{val}. Despite its ad-hoc nature, we find that this definition of a validation error provides a useful way of quantifying how the regularization scheme affects the solution of the EOM in the presence of noise. Figure 9 shows the integrated TDVP and validation error for weak and moderate driving. While the bare TDVP error is insensitive to the Monte Carlo error and corresponding instabilities[3], a clearly discernible effect is present in the validation error which therefore shows a much better qualitative agreement with the reference ED error (compare Fig. 6). Note that we show here the error for EMC sampling because, while the utility of the local validation error is not limited to this case, the integrated curves are strongly affected by local perturbances present in the MCMC data.

Figure 10 shows the local TDVP and validation error over a range of thresholds at various times during the duration of the pulse. Here we can see that the unstable regimes of regularization indeed correspond to an increased validation error r^2_{val} compared to r^2_{tr}, which is consistent with their interpretation as being a consequence of overfitting to noise in the Monte Carlo update, while in the stable regions almost no overfitting error is observed, indicating a high degree of consistency between updates. Furthermore, this behavior is not uniform over time: For $A_{\mathrm{p}} = 0.10$, overfitting occurs particularly strongly at the waning edge of the pulse. The degree of overfitting and its sensitivity to the regularization strength is significantly lower for the weaker excitation. Note that the absolute magnitude of the TDVP error is not directly comparable between different pulse strengths. This is because the denominator of Eq. (12) depends on the distance between $|\psi(t)\rangle$ and $|\psi(t+\delta t)\rangle$ which decreases with decreasing driving

[3]While R^2_{tr} does increase for small λ eventually in Fig. 9, this only happens after the jumps in the corresponding trajectories have already occurred, in contrast to R^2_{val} which detects hints of the instability already before that point.

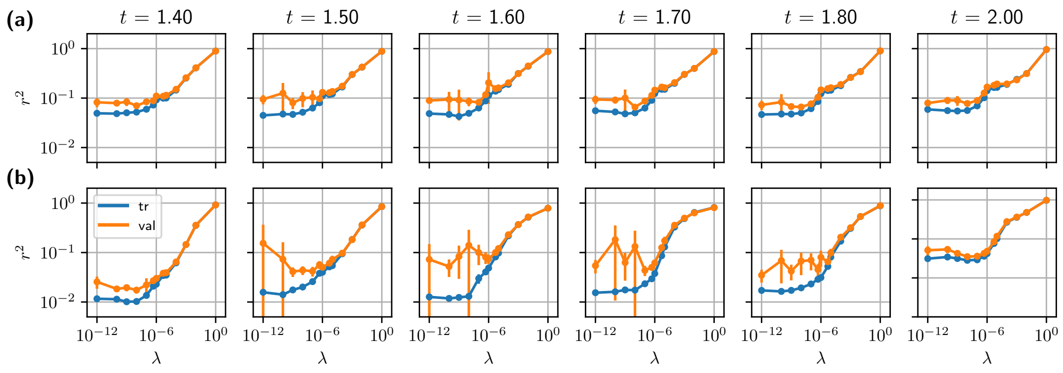


Figure 10: Comparison of the TDVP error r^2_{tr} and the validation error r^2_{val} on a logarithmic scale for varying regularization strength in the form of the SVD threshold λ at different points in time for **(a)** a weak $A_{\text{p}} = 0.02$ and **(b)** stronger driving amplitude $A_{\text{p}} = 0.10$ on the 8×2 ladder system. The TDVP error has been computed here by taking variational states $|\psi_{\theta(t)}\rangle$ from the stable $\lambda = 10^{-6}$ trajectory [Fig. 6(d),(e)] and then performing a single step $\delta t = 0.002$ at each displayed time and for each λ using Metropolis sampling with $N_s = 28 \cdot 10^3$ samples. We show here the average error over five independent realizations of the validation error, with error bars indicating the standard deviation, in order to account for variance in the error estimate itself.

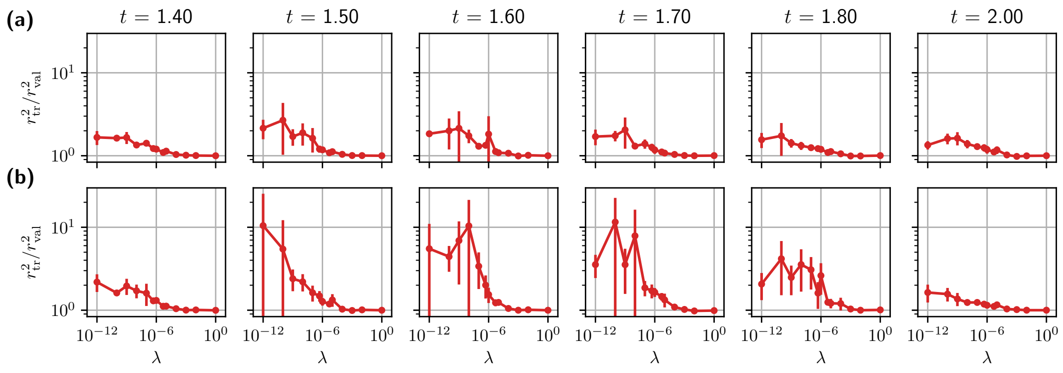


Figure 11: Validation error relative to the TDVP error for the data shown in Fig. 10.

strength and goes to zero for vanishing dynamics. Therefore, r^2 measures the error relative to the magnitude of the physical dynamics which puts the observed higher values of $r^2_{\text{tr/val}}$ for the weaker excitation strengths into perspective. Data for the validation error relative to the baseline TDVP error can be found in Fig. 11. The validation error also provides some insight into the behavior of the propagation depending on the number of Monte Carlo samples, which we briefly show in Appendix E.

We note that the region of increased overfitting coincides with a region where the exact quantum states, while being representable to a fidelity below 10^{-3}, appear to be harder to learn using a supervised scheme than states at other times (see Appendix C). However, we would like to stress it is still possible to achieve stable and accurate propagation in this region by suitable tuning of regularization and sample size, showing that an absolute inability of the RBM ansatz to represent those states is not the issue. The precise relationship between the difficulty of supervised optimization, sampling complexity, and generalization error remains an important question for further research, also in comparison with other works [23].

In summary, we have demonstrated that the sensitivity of the ladder system to the regularization scheme as well as the need for a high number of Monte Carlo samples to accurately estimate dynamics especially around the end of the THz pulse is indeed captured by the proposed TDVP validation error. These effects can thus be seen as a consequence of a lack of generalization of the derivative estimate $\dot{\theta}$ or overfitting to an insufficiently representative sample of spin configurations.

5 Conclusions and outlook

We have presented the time propagation of the Heisenberg model on the two-leg ladder as a key benchmark for neural-network-based methods to simulate quantum many-body dynamics. In line with other studies, we have found that RBM quantum states are in principle capable of representing the relevant quantum states during the simulated time evolutions, although important open questions remain regarding the relationship between learnability and sampling complexity of an NQS. However, the combination of (i) numerical instabilities already in small systems and (ii) tunability between relatively well-behaved dynamics on the square lattice and the much more challenging dynamics on the ladder make this model system a suitable case study for t-NQS. Moreover, larger-scale ladders can also be simulated with tensor network states, which makes Heisenberg ladders an ideal *drosophila* for more detailed comparisons between different systematically improvable variational ansätze and propagation schemes beyond system sizes accessible to exact diagonalization.

We have shed light on the delicate balance between stabilizing regularization and physical accuracy of the variational time evolution in the presence of stochastic noise inherent in the t-VMC approach. In particular, motivated by the interpretation of these instabilities as a consequence of overfitting to Monte Carlo noise in the equation of motion, we have introduced a validation-set approach as a quantitative diagnostic of the noise-based error. We have demonstrated that this validation error can be used to aid in the optimization of relevant hyperparameters and can help identifying critical regions where the propagation becomes particularly sensitive to noise. While this is particularly relevant for NQS dynamics, the validation-set approach can be applied to t-VMC simulations using other variational states as well as ground state optimization based on imaginary-time propagation.

The specific validation error introduced here is based on a second-order approximation of the TDVP error, which can be computed from quantities directly available during standard t-VMC runs. However, it is itself susceptible to noise and numerical instabilities. Therefore, while we have shown its capability of quantifiying the influence of regularization and highlighting regions of particularly unstable dynamics, finding a more robust measure of the error may be a useful line of future research.

The ability of quantifying the generalization error in t-VMC propagation also opens up the possibility of devising an adaptive scheme to control regularization hyperparameters and Monte Carlo sampling in order to achieve stable dynamics without the need for manual fine tuning. This is particularly relevant for general NQS software frameworks such as NETKET [55] which strive to be usable in a wide range of physical settings.

Acknowledgements

We acknowledge helpful discussions with Filippo Vicentini. Simulations have been performed using a t-VMC implementation based on NETKET 2.1b1 [55] as well as code based on ULTRAFAST [45]. The exact Schrödinger dynamics have been computed using QUTIP 4.5.0 [56].

Author contributions

D.H. implemented the t-VMC simulation code based on NETKET, performed simulations and data analysis, wrote the initial manuscript, with contributions from M.A.S., and created the figures. G.F. contributed t-VMC and full summation results using an independent implementation (ULTRAFAST). J.H.M., G.C., and M.A.S. conceived the project. All authors contributed to discussions throughout the project and the preparation of the final manuscript.

Funding information

D.H. and M.A.S. acknowledge support from the Max Planck-New York City Center for Nonequilibrium Quantum Phenomena. Computational resources have been provided by Flatiron Institute, a division of the Simons Foundation. This work is part of the Shell-NWO/FOM-initiative "Computational sciences for energy research" of Shell and Chemical Sciences, Earth and Life Sciences, Physical Sciences, FOM and STW, and received funding from the European Research Council ERC grant agreement No. 856538 (3D-MAGiC) (G.F., J.H.M.). M.A.S. acknowledges funding by Deutsche Forschungsgemeinschaft (German Research Foundation, DFG) under the Emmy Noether program (SE 2558/2).

A Variational ansatz

In our simulations, we employ the translation-invariant RBM ansatz as introduced in Ref. [8]. Explicitly,

$$\ln \psi_\theta(s) = \sum_{j=1}^{N_\mathrm{h}} \ln\cosh\,[\tilde{W}s + \tilde{b}]_j\,. \tag{17}$$

Here, the full weight matrix $\tilde{W} \in \mathbb{C}^{N_\mathrm{h}\times N}$ and hidden bias $\tilde{b} \in \mathbb{C}^{N_\mathrm{h}}$ are defined in terms of a smaller number of independent parameters $W \in \mathbb{C}^{\alpha\times N}$ and $b \in \mathbb{C}^{\alpha}$, ensuring that $\psi(\tau(s)) = \psi(s)$ for all N lattice translations τ. We refer to the independent parameters collectively as $\theta = (W, b) \in \mathbb{C}^M$. This ansatz reduces the number of variational parameters to $M = \alpha(N+1)$ where $\alpha = N_\mathrm{h}/N$ is the hidden unit density. Thus, the dimension of the parameter space grows only linearly in the system size for the symmetric RBM ansatz. Note that RBM wave functions can include another term, the visible bias $\tilde{a} \in \mathbb{C}^N$, as $\psi_{a,\theta}(s) = e^{\tilde{a}^\top s}\,\psi_\theta(s)$. When enforcing translation invariance in the manner described above, only one component of the visible bias $\tilde{a}_i = a \in \mathbb{C}$ remains independent which is redundant in the zero magnetization sector and therefore not included in our variational ansatz.

In addition to enforcing translation symmetry, we restrict the state space of the model to the zero magnetization subspace of the full Hilbert space. Thus, the ansatz wave function is only evaluated for spin configurations satisfying $\sum_j s_j = 0$ and $\psi_\theta(s) = 0$ is assumed otherwise. For N sites, the dimension of the full Hilbert space is 2^N, the zero-magnetization subspace has dimension $\binom{N}{N/2}$. Note that the driving is compatible with both the translation symmetry and zero magnetization constraints. Furthermore, all of our calculations were performed in a computational basis taking into account the sign structure of the AFM ground state, which is a standard approach for the Heisenberg model [57–59] and helps circumvent the difficulty of learning states with a nontrivial sign structure, which is a more challenging task for NQS [17,18]. Specifically, this is done as follows: Let $\{|s\rangle \mid s \in \{\pm 1\}^N\}$ denote the $\hat{\sigma}^z$ eigenbasis, so that $\hat{\sigma}^z_i|s\rangle = s_i|s\rangle$. In the AFM phase, the Heisenberg model has a nondegenerate ground state

$$|\Phi_0\rangle = \sum_{s\in\{\pm1\}^N} \Phi_0(s)|s\rangle\,, \tag{18}$$

which is part of the zero eigenspace of the magnetization $\hat{M}^z = \sum_{i\in\mathfrak{L}} \hat{\sigma}_i^z$. On a bipartite lattice where the sites are partitioned into disjoint subsets $\mathcal{A}$ and $\mathcal{B}$ [compare Fig. 1(a),(b)], the ground state coefficients have the form

$$\Phi_0(s) = (-1)^{\varpi(s)} A_0(s)\,, \tag{19}$$

where $A_0(s) \in \mathbb{R}_{>0}$ is real and nonnegative. The parity $\varpi(s) = \sum_{i\in\mathcal{A}} s_i$ is determined by the magnetization on the $\mathcal{A}$ sublattice. This property is known as Marshall's sign rule [60] and makes it possible to represent the ground state by a real nonnegative wave function in the computational basis $|s^{\mathrm{c}}\rangle = \prod_{i\in\mathcal{A}} \hat{\sigma}_i^z |s\rangle$, which significantly improves convergence of the NQS ground state optimization as the network only needs to learn a trivial sign structure.

B Time propagation

In t-VMC [40, 41], the time propagation of the variational ansatz is based on the time-dependent variational principle (TDVP). The equation of motion for the vector of variational parameters $\theta \in \mathbb{C}^M$ is given by $\mathrm{d}\theta(t)/\mathrm{d}t = \dot{\theta}$, where $\dot{\theta}$ is the solution of the linear system Eq. (5). The quantities involved in this equation are the quantum Fisher matrix (QFM)

$$S_{ij}(\theta) = \frac{\langle \partial_i \psi_\theta | \partial_j \psi_\theta \rangle}{\langle \psi_\theta | \psi_\theta \rangle} - \frac{\langle \partial_i \psi_\theta | \psi_\theta \rangle \langle \psi_\theta | \partial_j \psi_\theta \rangle}{\langle \psi_\theta | \psi_\theta \rangle^2}\,, \tag{20}$$

and the energy gradient

$$F_i(\theta, t) = \frac{\partial \langle \hat{H}(t) \rangle}{\partial \theta_i^*} = \frac{\langle \partial_i \psi_\theta | \hat{H}(t) - \langle \hat{H}(t) \rangle | \psi_\theta \rangle}{\langle \psi_\theta | \psi_\theta \rangle}\,. \tag{21}$$

Here, $\partial_i = \partial/\partial\theta_i$ denotes the complex partial derivative with respect to θ_i. Geometrically, the QFM accounts for the local curvature around $|\psi_\theta\rangle$ on the manifold of variational states. This is analogous to the role of the Fisher information matrix for classical probability distributions [28], which is used in natural gradient descent [61]. The QFM only depends on the form of the variational ansatz and the location in parameter space but not on the Hamiltonian. Still, analysis of its structure and, in particular, its spectrum can give insight into the quantum properties and phase diagram of the system for the RBM ansatz [53].

In t-VMC, both QFM and gradient are estimated as stochastic expectations values $\mathbb{E}[\,\cdot\,]$ with respect to the Born probability distribution $\sim |\psi_\theta(\,\cdot\,)|^2$ as written in Eqs. (6) and (7). The resulting equations of motion are valid for a complex differentiable (holomorphic) mapping $\theta \mapsto \psi_\theta$ between the parameters and the quantum wave function. This is indeed satisfied by the symmetric RBM ansatz.

In order to obtain the ground states used as initial states for the time propagation, we have used stochastic reconfiguration [8, 41], which is based on an approximation of the imaginary-time Schrödinger equation in the manner of Eq. (5).

C Representability of the trajectory

Even though wider RBMs are necessary in order to better follow the true dynamics via TDVP-based propagation, the states along the trajectories considered here are in general not significantly harder to learn by an RBM than the ground state. In order to test this statement, we fit RBMs with $\alpha = 2$ and $\alpha = 10$ to the exact states along the ED trajectory $|\Psi(t)\rangle$ in the ladder

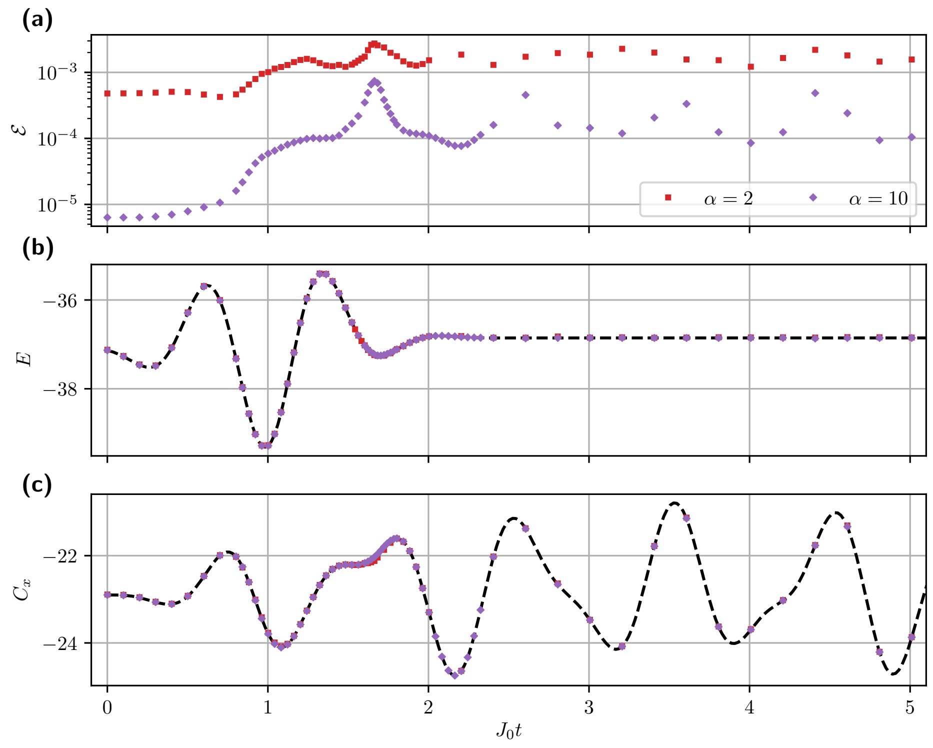


Figure 12: **(a)** Infidelity [Eq. (11)], **(b)** energy, and **(c)** spin-spin correlation C_x for $\alpha = 2$ and $\alpha = 10$ RBM states obtained from supervised learning of the amplitudes along the exact trajectory at an excitation strength of $A_\mathrm{p} = 0.10$. For each time point, the best-fidelity results among five independently optimized states are displayed. See Appendix C for further details.

system using a supervised learning approach. The results presented here have been computed using the supervised learning implementation available in NETKET [55]. Specifically, for each given time t, we minimize the negative log-overlap loss

$$L_t(\theta) = -\ln \frac{\langle \psi_\theta | \Psi(t) \rangle \langle \Psi(t) | \psi_\theta \rangle}{\langle \psi_\theta | \psi_\theta \rangle \langle \Psi(t) | \Psi(t) \rangle}, \tag{22}$$

using the known probability amplitudes of the exact states that we have obtained from ED. Starting from an approximate ground state, we have run a natural-gradient based optimization [61] targeting this loss function for $n = 1000$ steps at a constant learning rate of $\gamma = 0.01$. This corresponds to a parameter update

$$\theta^{(i+1)} \leftarrow \theta^{(i)} - \gamma\, g_t(\theta^{(i)}), \tag{23}$$

with the loss gradient $g_t(\theta)$ which is the least-squares solution of the linear equation

$$S(\theta)\, g_t(\theta) = \nabla_\theta L_t(\theta). \tag{24}$$

The QFM $S(\theta)$ is defined in the same way as in the main text. In practise, we evaluate the loss $L_t(\theta)$ on batches of spin configurations $\{s^{(i)}\}_{i=1}^{B}$ of size $B = 1000$ per step, which are randomly drawn from a uniform distribution over all zero-magnetization configurations on the lattice.

Note that this update equation is of the same form as in stochastic reconfiguration (SR) [41]. This is because SR is a special case of the natural gradient descent approach applied to the energy expectation value as opposed to a general loss function [28]. The optimization is performed independently for each time t. Results for the final infidelity, energy, and spin-spin correlation are shown in Fig. 12. For each time point, the best-fidelity state has been selected from five independent optimizations from the same initial state and with the same parameters[4]. Already at a small hidden unit density of $\alpha = 2$, the trajectory can be represented with an infidelity of the order of 10^{-3} with good accuracy in energy and spin correlations. For a wider network at $\alpha = 10$ and starting from a well-converged initial state at $\mathcal{E} = 10^{-5}$, the trajectory can be captured with infidelity below 10^{-3} throughout, although a peak of infidelity is clearly visible around $t_* \approx 1.65$. The location of this peak matches the region of increased overfitting and thus instability observed in Sect. 4 (compare Fig. 10). However, even though the peak region is more prone to instabilities, it can still be passed by t-VMC if the time propagation is sufficiently stabilized (compare Fig. 7). Thus, the increased final infidelity of the learned excited states is not necessarily an indication of an absolute inability of the RBM ansatz to capture them more accurately but may also be attributed to an increased difficulty of the optimization.

Altogether, these results provide an upper bound on the minimal infidelity achievable by an optimal RBM representation of the dynamic states at a given size and thus indicate that the representability of the time-evolved states is not the key limitation here. Therefore, an improved time propagation scheme should be expected to be able to reach the accuracy of the supervised learned states.

D Alternative regularization schemes

There are many ways to regularize the linear equation of motion (5) to reduce its susceptibility to noise. We have focused here on the conceptually simple method of truncating the QFM spectrum at a relative threshold as described in Sect. 3.2. Alternatively, the EOM can be regularized by adding a diagonal shift $\tilde{S} = S + \epsilon I$, with $\epsilon > 0$. This is typically done for ground state optimization (see, e.g., Refs. [8,42,46]), but can in principle also be applied to the time-dependent case. Since S is Hermitian and positive semi-definite, the spectrum of the shifted $\tilde{S}$ is bounded from below by ϵ, making the matrix invertible and bounding the condition number by $\kappa(\tilde{S}) \leq (\zeta_1 + \epsilon)/\epsilon$. Therefore, a shift significantly larger than machine precision also serves to improve the condition number and stabilize the propagation in a fashion similar to the SVD cutoff. This can be seen in Fig. 13(a) which shows a similar behavior of the shift regularization when compared to the threshold in Fig. 6. A more sophisticated regularization strategy has recently been proposed in Ref. [44]. This approach truncates parts of the equations of motion akin to the singular value threshold above but taking into account the strength of VMC noise in different components of the energy gradient. Specifically, directions in the S eigenbasis are discarded based on a softened cutoff λ_{SNR} of the signal-to-noise of the corresponding component of the energy gradient which can be estimated from the t-VMC data. When applying this approach to the ladder geometry, we have found a behavior similar to the other methods discussed above as a function of varying λ_{SNR} [Fig. 13(b)]. This highlights that the trade-off between stability and physical accuracy we have discussed and the need for reliable

[4]In the region of peak infidelity around t_* defined below, the optimization of the $\alpha = 2$ RBM frequently became unstable after reaching the minimal energy, leading to an increased energy after 1000 steps compared to the actual achievable minimum. Therefore, for the results presented here, the iteration has been stopped early after 10 successive steps without reduction of the loss in this region. The optimization of the $\alpha = 10$ RBM did not have this issue.

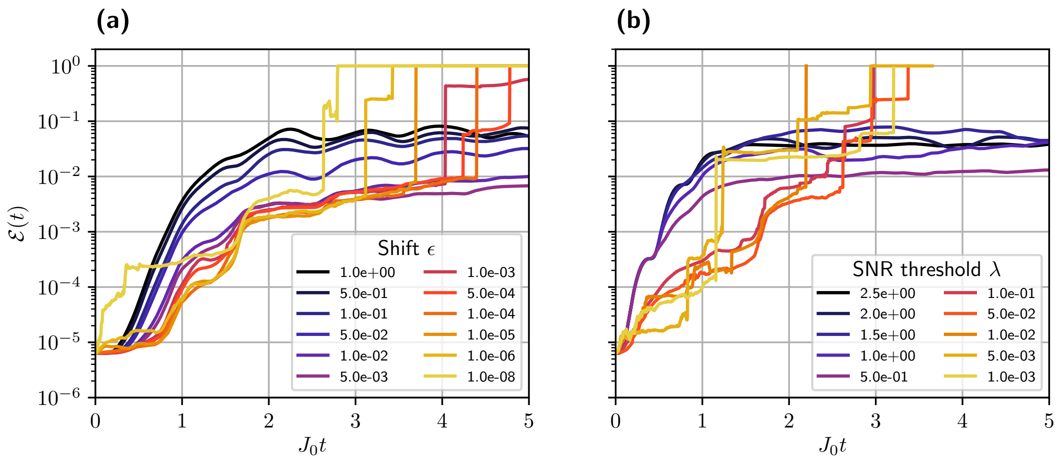


Figure 13: Infidelity of the time-evolved variational state compared to the exact trajectory for **(a)** the diagonal shift regularization with varying strength ϵ and **(b)** the signal-to-noise (SNR) ratio based regularization of Ref. [44] with varying threshold λ_{SNR}. The trajectories have been computed using t-VMC with EMC at a moderate excitation strength of $A_\text{p} = 0.10$ in the 8×2 ladder geometry using the symmetrized RBM ansatz with a hidden unit density of $\alpha = 10$.

diagnostics is relevant beyond the simple regularization scheme used in the main text.

E Validation error and Monte Carlo sample size

While sensitive to the regularization, the Monte Carlo error in the update $\dot{\theta}$ is of course also dependent on the number of samples used in the estimate of the equation of motion. The fewer samples are used, the higher the generalization error with respect to the full Hilbert space will be. As with the regularization, this behavior is strongly system and excitation dependent. For the square lattice geometry, convergence with respect to the sample size occurs quickly compared to the ladder system, where a much higher number of samples is needed to obtain reliable estimates. This indicates a higher sampling complexity of the ladder states compared to the well-behaved singlet magnon excitation in the square lattice, as is qualitatively captured by the validation error. This is demonstrated in Fig. 14, which shows the TDVP and validation error, and Fig. 15, which shows the relative validation error, as a function of sample size for both geometries and different driving strengths. We see a clear overfitting behavior which is especially strong around the waning edge of the pulse, as is similarly observed in the regularization dependence in Section 4, and which is only suppressed by increasing the number of samples up to $\sim 28 \cdot 10^3$ for the ladder[5]. In contrast, convergence of the validation error in the square lattice system occurs much faster, even for the stronger excitation shown here.

[5]We note that simulations with larger sample sizes are computationally feasible (compare, e.g., Ref. [44]). However, $\sim 28 \cdot 10^3$ already exceeds the Hilbert space dimension of our benchmark systems by more than a factor of two. Given this already large relative size and the fact that MCMC convergence is proportional to the square root of the (effective) sample size, we consider it important to be able to stabilize the dynamics through regularization in this regime.

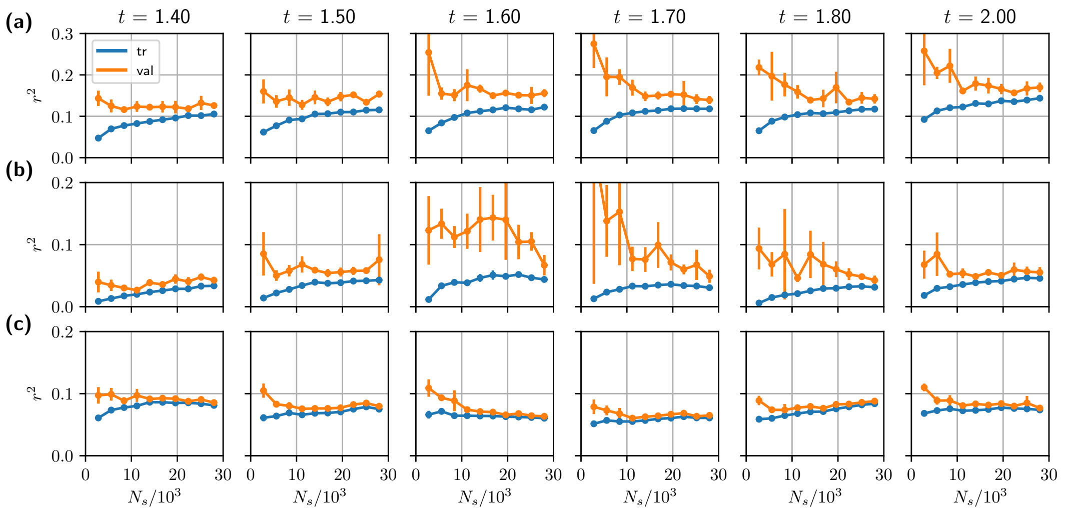


Figure 14: **(a),(b)** Comparison of TDVP and validation error for varying number of Monte Carlo samples on a linear scale for the same trajectory and parameters as in Fig. 10, with $\lambda = 10^{-6}$ and at driving amplitudes $A_\text{p} = 0.02$ [panel (a)] and $A_\text{p} = 0.10$ [panel (b)] in the 8×2 ladder system. **(c)** The same data for an $A_\text{p} = 0.20$ pulse in the 4×4 square geometry. In all panels, the error bars indicate the standard deviation over five independent realizations of the validation error as in Fig. 10.

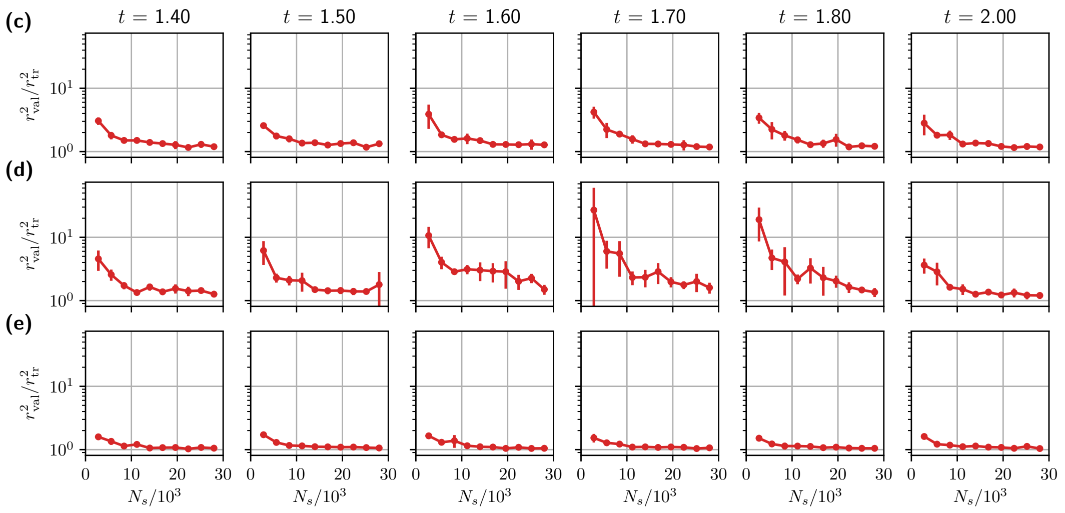


Figure 15: Validation error relative to the TDVP error for the data shown in Fig. 14.

References

[1] G. Carleo, I. Cirac, K. Cranmer, L. Daudet, M. Schuld, N. Tishby, L. Vogt-Maranto and L. Zdeborová, *Machine learning and the physical sciences*, Rev. Mod. Phys. **91**, 045002 (2019), doi:10.1103/RevModPhys.91.045002.

[2] G. Torlai, G. Mazzola, J. Carrasquilla, M. Troyer, R. Melko and G. Carleo, *Neural-network quantum state tomography*, Nat. Phys. **14**, 447 (2018), doi:10.1038/s41567-018-0048-5.

[3] M. Bukov, A. G. R. Day, D. Sels, P. Weinberg, A. Polkovnikov and P. Mehta, *Reinforcement learning in different phases of quantum control*, Phys. Rev. X **8**, 031086 (2018),

doi:10.1103/PhysRevX.8.031086.

[4] T. Fösel, P. Tighineanu, T. Weiss and F. Marquardt, *Reinforcement learning with neural networks for quantum feedback*, Phys. Rev. X **8**, 031084 (2018), doi:10.1103/PhysRevX.8.031084.

[5] J. Carrasquilla and R. G. Melko, *Machine learning phases of matter*, Nat. Phys. **13**, 431 (2017), doi:10.1038/nphys4035.

[6] B. S. Rem, N. Käming, M. Tarnowski, L. Asteria, N. Fläschner, C. Becker, K. Sengstock and C. Weitenberg, *Identifying quantum phase transitions using artificial neural networks on experimental data*, Nat. Phys. **15**, 917 (2019), doi:10.1038/s41567-019-0554-0.

[7] A. Bohrdt et al., *Classifying snapshots of the doped Hubbard model with machine learning*, Nat. Phys. **15**, 921 (2019), doi:10.1038/s41567-019-0565-x.

[8] G. Carleo and M. Troyer, *Solving the quantum many-body problem with artificial neural networks*, Science **355**, 602 (2017), doi:10.1126/science.aag2302.

[9] I. Glasser, N. Pancotti, M. August, I. D. Rodriguez and J. I. Cirac, *Neural-network quantum states, string-bond states, and chiral topological states*, Phys. Rev. X **8**, 011006 (2018), doi:10.1103/PhysRevX.8.011006.

[10] R. Kaubruegger, L. Pastori and J. C. Budich, *Chiral topological phases from artificial neural networks*, Phys. Rev. B **97**, 195136 (2018), doi:10.1103/PhysRevB.97.195136.

[11] K. Choo, G. Carleo, N. Regnault and T. Neupert, *Symmetries and many-body excitations with neural-network quantum states*, Phys. Rev. Lett. **121**, 167204 (2018), doi:10.1103/PhysRevLett.121.167204.

[12] X. Liang, W.-Y. Liu, P.-Z. Lin, G.-C. Guo, Y.-S. Zhang and L. He, *Solving frustrated quantum many-particle models with convolutional neural networks*, Phys. Rev. B **98**, 104426 (2018), doi:10.1103/PhysRevB.98.104426.

[13] L. Pastori, R. Kaubruegger and J. Carl Budich, *Generalized transfer matrix states from artificial neural networks*, Phys. Rev. B **99**, 165123 (2019), doi:10.1103/PhysRevB.99.165123.

[14] K. Choo, T. Neupert and G. Carleo, *Two-dimensional frustrated J_1-J_2 model studied with neural network quantum states*, Phys. Rev. B **100**, 125124 (2019), doi:10.1103/PhysRevB.100.125124.

[15] D. Hendry and A. E. Feiguin, *Machine learning approach to dynamical properties of quantum many-body systems*, Phys. Rev. B **100**, 245123 (2019), doi:10.1103/PhysRevB.100.245123.

[16] T. Vieijra, C. Casert, J. Nys, W. De Neve, J. Haegeman, J. Ryckebusch and F. Verstraete, *Restricted Boltzmann machines for quantum states with non-Abelian or anyonic symmetries*, Phys. Rev. Lett. **124**, 097201 (2020), doi:10.1103/PhysRevLett.124.097201.

[17] T. Westerhout, N. Astrakhantsev, K. S. Tikhonov, M. I. Katsnelson and A. A. Bagrov, *Generalization properties of neural network approximations to frustrated magnet ground states*, Nat. Commun. **11**, 1593 (2020), doi:10.1038/s41467-020-15402-w.

[18] A. Szabó and C. Castelnovo, *Neural network wave functions and the sign problem*, Phys. Rev. Research **2**, 033075 (2020), doi:10.1103/PhysRevResearch.2.033075.

[19] M. Bukov, M. Schmitt and M. Dupont, *Learning the ground state of a non-stoquastic quantum Hamiltonian in a rugged neural network landscape*, SciPost Phys. **10**, 147 (2021), doi:10.21468/SciPostPhys.10.6.147.

[20] C.-Y. Park and M. J. Kastoryano, *Expressive power of complex-valued restricted Boltzmann machines for solving non-stoquastic hamiltonians*, arXiv:2012.08889v2.

[21] N. Astrakhantsev, T. Westerhout, A. Tiwari, K. Choo, A. Chen, M. H. Fischer, G. Carleo and T. Neupert, *Broken-symmetry ground states of the Heisenberg model on the pyrochlore lattice*, Phys. Rev. X **11**, 041021 (2021), doi:10.1103/PhysRevX.11.041021.

[22] A. Valenti, E. Greplova, N. H. Lindner and S. D. Huber, *Correlation-enhanced neural networks as interpretable variational quantum states*, Phys. Rev. Research **4**, L012010 (2022), doi:10.1103/PhysRevResearch.4.L012010.

[23] S.-H. Lin and F. Pollmann, *Scaling of neural-network quantum states for time evolution*, Phys. Status Solidi (b) 2100172 (2022), doi:10.1002/pssb.202100172.

[24] K. McBrian, G. Carleo and E. Khatami, *Ground state phase diagram of the one-dimensional Bose-Hubbard model from restricted Boltzmann machines*, J. Phys.: Conf. Ser. **1290**, 012005 (2019), doi:10.1088/1742-6596/1290/1/012005.

[25] Y. Nomura, A. S. Darmawan, Y. Yamaji and M. Imada, *Restricted Boltzmann machine learning for solving strongly correlated quantum systems*, Phys. Rev. B **96**, 205152 (2017), doi:10.1103/PhysRevB.96.205152.

[26] K. Choo, A. Mezzacapo and G. Carleo, *Fermionic neural-network states for ab-initio electronic structure*, Nat. Commun. **11**, 2368 (2020), doi:10.1038/s41467-020-15724-9.

[27] B. Jónsson, B. Bauer and G. Carleo, *Neural-network states for the classical simulation of quantum computing*, arXiv:1808.05232.

[28] J. Stokes, J. Izaac, N. Killoran and G. Carleo, *Quantum natural gradient*, Quantum **4**, 269 (2020), doi:10.22331/q-2020-05-25-269.

[29] G. Torlai and R. G. Melko, *Latent space purification via neural density operators*, Phys. Rev. Lett. **120**, 240503 (2018), doi:10.1103/PhysRevLett.120.240503.

[30] M. J. Hartmann and G. Carleo, *Neural-network approach to dissipative quantum many-body dynamics*, Phys. Rev. Lett. **122**, 250502 (2019), doi:10.1103/PhysRevLett.122.250502.

[31] N. Yoshioka and R. Hamazaki, *Constructing neural stationary states for open quantum many-body systems*, Phys. Rev. B **99**, 214306 (2019), doi:10.1103/PhysRevB.99.214306.

[32] F. Vicentini, A. Biella, N. Regnault and C. Ciuti, *Variational neural-network ansatz for steady states in open quantum systems*, Phys. Rev. Lett. **122**, 250503 (2019), doi:10.1103/PhysRevLett.122.250503.

[33] J. Eisert, M. Friesdorf and C. Gogolin, *Quantum many-body systems out of equilibrium*, Nat. Phys. **11**, 124 (2015), doi:10.1038/nphys3215.

[34] R. J. Lewis-Swan, A. Safavi-Naini, A. M. Kaufman and A. M. Rey, *Dynamics of quantum information*, Nat. Rev. Phys. **1**, 627 (2019), doi:10.1038/s42254-019-0090-y.

[35] D. Bluvstein et al., *Controlling quantum many-body dynamics in driven Rydberg atom arrays*, Science **371**, 1355 (2021), doi:10.1126/science.abg2530.

[36] I. Carusotto and C. Ciuti, *Quantum fluids of light*, Rev. Mod. Phys. **85**, 299 (2013), doi:10.1103/RevModPhys.85.299.

[37] A. de la Torre, D. M. Kennes, M. Claassen, S. Gerber, J. W. McIver and M. A. Sentef, *Colloquium: Nonthermal pathways to ultrafast control in quantum materials*, Rev. Mod. Phys. **93**, 041002 (2021), doi:10.1103/RevModPhys.93.041002.

[38] D.-L. Deng, X. Li and S. Das Sarma, *Quantum entanglement in neural network states*, Phys. Rev. X **7**, 021021 (2017), doi:10.1103/PhysRevX.7.021021.

[39] J. Eisert, M. Cramer and M. B. Plenio, *Colloquium: Area laws for the entanglement entropy*, Rev. Mod. Phys. **82**, 277 (2010), doi:10.1103/RevModPhys.82.277.

[40] G. Carleo, F. Becca, M. Schiró and M. Fabrizio, *Localization and glassy dynamics of many-body quantum systems*, Sci. Rep. **2**, 243 (2012), doi:10.1038/srep00243.

[41] F. Becca and S. Sorella, *Quantum Monte Carlo approaches for correlated systems*, Cambridge University Press, Cambridge, UK, ISBN 9781107129931 (2017), doi:10.1017/9781316417041.

[42] S. Czischek, M. Gärttner and T. Gasenzer, *Quenches near Ising quantum criticality as a challenge for artificial neural networks*, Phys. Rev. B **98**, 024311 (2018), doi:10.1103/PhysRevB.98.024311.

[43] I. L. Gutiérrez and C. B. Mendl, *Real time evolution with neural-network quantum states*, Quantum **6**, 627 (2022), doi:10.22331/q-2022-01-20-627.

[44] M. Schmitt and M. Heyl, *Quantum many-body dynamics in two dimensions with artificial neural networks*, Phys. Rev. Lett. **125**, 100503 (2020), doi:10.1103/PhysRevLett.125.100503.

[45] G. Fabiani and J. Mentink, *Investigating ultrafast quantum magnetism with machine learning*, SciPost Phys. **7**, 004 (2019), doi:10.21468/SciPostPhys.7.1.004.

[46] G. Fabiani and J. H. Mentink, *Ultrafast entanglement dynamics and spreading of quantum correlations in two-dimensional antiferromagnets*, arXiv:1912.10845.

[47] G. Fabiani, M. D. Bouman and J. H. Mentink, *Supermagnonic propagation in two-dimensional antiferromagnets*, Phys. Rev. Lett. **127**, 097202 (2021), doi:10.1103/PhysRevLett.127.097202.

[48] D. Bossini et al., *Laser-driven quantum magnonics and terahertz dynamics of the order parameter in antiferromagnets*, Phys. Rev. B **100**, 024428 (2019), doi:10.1103/physrevb.100.024428.

[49] T. Barnes, E. Dagotto, J. Riera and E. S. Swanson, *Excitation spectrum of Heisenberg spin ladders*, Phys. Rev. B **47**, 3196 (1993), doi:10.1103/PhysRevB.47.3196.

[50] S. Gopalan, T. M. Rice and M. Sigrist, *Spin ladders with spin gaps: A description of a class of cuprates*, Phys. Rev. B **49**, 8901 (1994), doi:10.1103/PhysRevB.49.8901.

[51] E. Dagotto and T. M. Rice, *Surprises on the way from one- to two-dimensional quantum magnets: The ladder materials*, Science **271**, 618 (1996), doi:10.1126/science.271.5249.618.

[52] O. Sharir, Y. Levine, N. Wies, G. Carleo and A. Shashua, *Deep autoregressive models for the efficient variational simulation of many-body quantum systems*, Phys. Rev. Lett. **124**, 020503 (2020), doi:10.1103/PhysRevLett.124.020503.

[53] C.-Y. Park and M. J. Kastoryano, *Geometry of learning neural quantum states*, Phys. Rev. Research **2**, 023232 (2020), doi:10.1103/PhysRevResearch.2.023232.

[54] I. Goodfellow, Y. Bengio and A. Courville, *Deep learning*, MIT Press, Cambridge, Massachusetts, US, ISBN 9780262035613 (2016).

[55] G. Carleo et al., *NetKet: A machine learning toolkit for many-body quantum systems*, SoftwareX **10**, 100311 (2019), doi:10.1016/j.softx.2019.100311.

[56] J. R. Johansson, P. D. Nation and F. Nori, *QuTiP 2: A Python framework for the dynamics of open quantum systems*, Comput. Phys. Commun. **184**, 1234 (2013), doi:10.1016/j.cpc.2012.11.019.

[57] D. J. Klein, *Ground-state features for Heisenberg models*, J. Chem. Phys. **77**, 3098 (1982), doi:10.1063/1.444232.

[58] U. Schollwoeck, *Marshall's sign rule and DMRG acceleration*, arXiv:cond-mat/9804231.

[59] R. F. Bishop, D. J. J. Farnell and J. B. Parkinson, *Sign rules for anisotropic quantum spin systems*, Phys. Rev. B **61**, 6775 (2000), doi:10.1103/PhysRevB.61.6775.

[60] W. Marshall, *Antiferromagnetism*, Proc. R. Soc. Lond. A **232**, 48 (1955), doi:10.1098/rspa.1955.0200.

[61] S.-i. Amari, *Natural gradient works efficiently in learning*, Neural Comput. **10**, 251 (1998), doi:10.1162/089976698300017746.

5 Symmetries in neural quantum states

Symmetries are operations under which a physical system, represented by its Hamiltonian in quantum many-body physics, is invariant. Knowing the symmetry sector of a target state can be utilized to either improve convergence by constraining the ansatz to the desired sector or evaluate the accuracy of an optimized unconstrained trial state.

In a quantum system with Hamiltonian $\hat{H}$, a symmetry corresponds to an invertible operator $\hat{T}$ that commutes with the Hamiltonian, $[\hat{H}, \hat{T}] = 0$. As a consequence, the two operators are simultaneously diagonalizable. Every non-degenerate eigenstate of $\hat{H}$ is, therefore, also an eigenstate of $\hat{T}$. Degenerate energy eigenstates can still be superpositions of states in different symmetry sectors. Thus, while the simultaneous diagonalizability guarantees the existence of a complete eigenbasis of joint eigenstates of $\hat{H}$ and $\hat{T}$, arbitrary energy eigenstates are not necessarily eigenstates of $\hat{T}$. If the ground state of a system is degenerate, variational energy minimization will generally yield an arbitrary vector within the ground state manifold (if it can be represented and learned). In this case, symmetry projections can be used for targeting specific ground-state subspaces. Symmetry projections can also restrict the ansatz to specific non-ground-state sectors to target excited states by energy minimization [88, 92].

Generally, symmetries of a quantum system can be described in terms of symmetry groups and their representations. Therefore, some concepts from group theory will appear in the following sections. A more comprehensive overview of group-theoretic concepts can be found in the literature, e.g., in Refs. [217, 218]. We only consider Abelian symmetry groups[18]; approaches to enforce non-Abelian symmetries are discussed in Refs. [200, 219].

In the following, Section 5.1 will review several approaches from the literature to implement symmetries in NQS. Then, Section 5.2 will discuss a specific application of symmetry projections in the context of the honeycomb Kitaev model (HKM).

5.1 Implementing symmetries in NQS

This section discusses several approaches for constructing symmetry-adapted variational states.

5.1.1 Diagonal symmetries and basis restrictions

The simplest forms of symmetries are those that can be represented by an operator $\hat{D}$ that is diagonal in the computational basis, i.e., $\langle \boldsymbol{s}'|\hat{D}|\boldsymbol{s}\rangle = D(\boldsymbol{s})\delta_{\boldsymbol{s},\boldsymbol{s}'}$. Restricting the trial state to the eigenspace of $\hat{D}$

[18]A group G is Abelian if all its elements commute, i.e., if for all $g, g' \in G$: $gg' = g'g$ [217].

corresponding to a specific eigenvalue $\lambda \in \mathbb{R}$ then requires the eigenvalue condition

$$(\hat{D} - \lambda)|\psi_{\boldsymbol{\theta}}\rangle = 0 \quad \Leftrightarrow \quad [D(\boldsymbol{s}) - \lambda]\psi_{\boldsymbol{\theta}}(\boldsymbol{s}) = 0 \tag{5.1}$$

to hold for all $\boldsymbol{s} \in \mathcal{S}$. Thus, whenever $D(\boldsymbol{s}) \neq \lambda$, the corresponding probability amplitude needs to vanish or, equivalently, $\operatorname{supp}\psi_{\boldsymbol{\theta}} \subseteq \mathcal{D}_\lambda = \{\boldsymbol{s} \in \mathcal{S} \mid D(\boldsymbol{s}) = \lambda\}$, while amplitudes corresponding to configurations $\boldsymbol{s} \in \mathcal{D}_\lambda$ within the desired eigenspace are unconstrained.

To enforce this condition in VMC, it is sufficient to restrict the Monte Carlo sampling process to the subset $\mathcal{D}_\lambda$, so that the wave function is only evaluated in the correct symmetry sector. The commutation relation $[\hat{H}, \hat{D}] = 0$ implies that the Hamiltonian only has non-zero matrix elements within each symmetry sector, i.e.,

$$\boldsymbol{s} \in \mathcal{D}_\lambda \Rightarrow \operatorname{Conn}(\boldsymbol{s}; \hat{H}) \subseteq \mathcal{D}_\lambda \tag{5.2}$$

[where the connected configurations $\operatorname{Conn}(\boldsymbol{s}; \hat{H})$ are defined as in Eq. (3.27)]. Thus, the local energy (3.23) is well-defined for the restricted set of configurations, and Eq. (3.24) can still be applied.

Since configurations outside $\mathcal{D}_\lambda$ have zero probability, they do not need not be explored during MCMC sampling (Section 3.3.3). Thus, the process needs to be implemented in such a way that configuration updates leading outside the symmetry sector are always rejected and it is necessary to adapt the proposal distribution so that ergodicity is preserved and, ideally, no symmetry-breaking updates are proposed.

As a specific example, consider a spin-1/2 system and the total z-direction magnetization

$$\hat{M}_z = \frac{1}{2}\sum_{i=1}^{N} \hat{\sigma}_i^z, \qquad M_z(\boldsymbol{s}) = \langle \hat{M}_z \rangle_{|\boldsymbol{s}\rangle} = \frac{1}{2}\sum_{i=1}^{N} s_i. \tag{5.3}$$

The operator $\hat{M}_z$ is diagonal in the $\hat{\sigma}^z$ eigenbasis $\hat{\sigma}_i^z|\boldsymbol{s}\rangle = s_i|\boldsymbol{s}\rangle$. This global magnetization is conserved, e.g., in the antiferromagnetic Heisenberg model, and the ground state is part of the zero-magnetization eigenspace. This symmetry has been regularly exploited in the NQS literature [70, 76, 79, P1] by restricting the simulation to that subspace, which is spanned by the configurations $\mathcal{D}_0 = \{\boldsymbol{s} \in \mathcal{S} \mid M_z(\boldsymbol{s}) = 0\}$. The standard proposal scheme for spin-1/2 systems is to propose single spin flips at randomly chosen indices. However, in $\mathcal{D}_0$, such proposals would always be rejected as they necessarily change the magnetization, leading to a stuck sampling process. Therefore, the proposal scheme is typically altered so that exchanges of spins at two randomly chosen sites are proposed instead, which conserves the total magnetization and retains ergodicity in $\mathcal{D}_0$, since any permutation of spins can be decomposed into a sequence of exchanges. Another example of a diagonal symmetry, the parity of the magnetization, will be discussed in Section 5.2.

5.1.2 Symmetry-projection by superposition

Non-diagonal symmetries can no longer be imposed simply by restricting the computational basis. However, given an unconstrained variational wave function $\tilde{\psi}_{\boldsymbol{\theta}}$, with no assumptions regarding its internal structure, it is possible to project the trial state to the desired symmetry sector [82]. The symmetrized amplitudes

$$\psi_{\boldsymbol{\theta}}(\boldsymbol{s}) = \langle \boldsymbol{s}|\hat{P}|\tilde{\psi}_{\boldsymbol{\theta}}\rangle \tag{5.4}$$

can then be used in VMC algorithms. This approach requires the action of the symmetrization operator $\hat{P}$ on the trial state to be efficiently computable.

We consider an Abelian symmetry group G with $N_{\text{sym}} = |G|$ elements acting on the configurations $\boldsymbol{s}$ by permutation of the indices. This action is denoted by $g.\boldsymbol{s}$ with $(g.\boldsymbol{s})_i = s_{g(i)}$ and the operator $\hat{T}_g$ is defined by $\hat{T}_g|\boldsymbol{s}\rangle = |g.\boldsymbol{s}\rangle$. Prominent examples of such groups are space groups of discrete lattices, which are generated by elementary translations, rotations, and reflections. An *invariant* trial state then has the property that, for all $g \in G$,

$$\psi_{\boldsymbol{\theta}}(\boldsymbol{s}) = \psi_{\boldsymbol{\theta}}(g.\boldsymbol{s}) \tag{5.5}$$

or, equivalently, $\hat{T}_g|\psi_{\boldsymbol{\theta}}\rangle = |\psi_{\boldsymbol{\theta}}\rangle$. Such a state can be obtained from any ansatz wave function $\tilde{\psi}_{\boldsymbol{\theta}}(\boldsymbol{s})$ by applying the symmetrization operator $\hat{P} = \frac{1}{N_{\text{sym}}} \sum_{g\in G} \hat{T}_g$, giving the amplitudes

$$\psi_{\boldsymbol{\theta}}(\boldsymbol{s}) = \frac{1}{N_{\text{sym}}} \sum_{g\in G} \tilde{\psi}_{\boldsymbol{\theta}}(g.\boldsymbol{s}). \tag{5.6}$$

Thus, the probability amplitudes of the symmetry-invariant state are obtained as a superposition of the unconstrained state's amplitudes for all configurations in its orbit $\boldsymbol{s}^G = \{g.\boldsymbol{s} \mid g \in G\}$.

This approach can be generalized beyond invariance. Different symmetry sectors (i.e., subspaces of states with specific symmetry quantum numbers) can be labeled by different irreducible representations (irreps) of the group[19] and, since G is Abelian by assumption, all its irreps are one-dimensional [217]. Consequently, each irrep ρ is determined by a mapping $\chi_\rho \colon G \to \mathbb{C} \setminus \{0\}$ which is called the *character* of the representation in this case. A general symmetry-projected state is constructed as [82, 218, P4]

$$\psi^{\rho}_{\boldsymbol{\theta}}(\boldsymbol{s}) = \frac{1}{N_{\text{sym}}} \sum_{g\in G} \chi^*_\rho(g)\, \tilde{\psi}_{\boldsymbol{\theta}}(g.\boldsymbol{s}), \tag{5.7}$$

with symmetrization operator $\hat{P}_\rho = \frac{1}{N_{\text{sym}}} \sum_{g\in G} \chi^*_\rho(g) \hat{T}_g$, and transforms according to the irrep ρ as

$$\psi^{\rho}_{\boldsymbol{\theta}}(g.\boldsymbol{s}) = \chi_\rho(g)\, \psi^{\rho}_{\boldsymbol{\theta}}(\boldsymbol{s}). \tag{5.8}$$

There always exists a trivial irrep ρ_0, the *unit representation*, mapping all elements to the identity and therefore satisfying $\chi_{\rho_0}(g) = 1$ for all $g \in G$. For this irrep, Eq. (5.7) reduces to the symmetry invariant Eq. (5.6).

To illustrate this abstract picture using a concrete symmetry, consider the example of a translation-invariant Hamiltonian on a one-dimensional N-site chain with periodic boundaries (adapted from an example in Ref. [218]). The translation group of this lattice is isomorphic to the cyclic group $\mathbb{Z}_N \simeq \{\tau^n\}_{n=0}^{N-1}$ and is generated by the shift operation τ, which acts on a configuration as

$$\tau.\boldsymbol{s} = (s_N, s_1, \dots, s_{N-1}). \tag{5.9}$$

The irreps of $\mathbb{Z}_N$ are the N-th roots of unity which can be labeled by the lattice momenta $k_n = 2\pi n/N$.

[19]Formally, an n-dimensional *representation* of a group G is a map from G to the space of invertible linear operators $\mathrm{GL}(V)$ on an n-dimensional vector space V. The representation associates an operator $\rho(g)$ with each group element g so that the group structure is respected, i.e., $\rho(g)\rho(g') = \rho(gg')$. An *irrep* is a representation with no nontrivial subrepresentations. For the purpose of this chapter, it is only relevant that the set of irreps of a symmetry group can be used to classify different symmetry sectors. See Ref. [217] for a detailed explanation.

The character is then given by

$$\chi_{k_n}(\tau^m) = e^{ik_n m}. \tag{5.10}$$

In physical terms, each irrep corresponds to a permissible value of the lattice momentum, and the symmetry-projected states

$$\psi_{\boldsymbol{\theta}}^n(\boldsymbol{s}) = \frac{1}{N} \sum_{m=1}^{N} e^{-ik_n m} \tilde{\psi}_{\boldsymbol{\theta}}(\tau^m.\boldsymbol{s}) \tag{5.11}$$

transform according to this momentum as

$$\psi_{\boldsymbol{\theta}}^n(\tau^m.\boldsymbol{s}) = e^{ik_n m} \psi_{\boldsymbol{\theta}}(\boldsymbol{s}). \tag{5.12}$$

As expected, translation-invariant states correspond to the trivial irrep with momentum $k_0 = 0$.

Adopting the symmetry-projection-based approach can significantly improve the accuracy of VMC results, as shown by Nomura [82]. It has also been observed that the construction of the symmetrized wave function (5.7) as a sum of many independent copies of the unconstrained trial wave function helps to improve convergence for target states with complicated phase structure [88, 89].

5.1.3 Symmetry-projection using representatives

The approach for symmetry projection discussed in the previous section requires N_{sym} evaluations of the variational ansatz. While this has its benefits (as outlined above), it also increases the computational cost of evaluating the symmetrized state and, therefore, any VMC optimization procedure linearly in the size of the symmetry group. An alternative approach first described for NQS by Choo et al. [92] is reviewed here.

As above, we consider an Abelian symmetry group G acting on the configurations in the computational basis, which associates the orbit $\boldsymbol{s}^G = \{g.\boldsymbol{s} \mid g \in G\}$ with each configuration. These orbits are equivalence classes, i.e., each configuration is contained in exactly one orbit, and the set of orbits partitions the set of all configurations. Instead of summing over all probability amplitudes contained in an orbit as in Eq. (5.7), it is possible to instead enforce the symmetry condition by selecting a *canonical representative* $\text{can}(\boldsymbol{s})$ that is constant on each orbit, i.e.,

$$\text{can}(\boldsymbol{s}) \in \boldsymbol{s}^G \qquad \text{and} \qquad \text{can}(g.\boldsymbol{s}) = \text{can}(\boldsymbol{s}) \; \forall g \in G. \tag{5.13}$$

This representative can be selected arbitrarily. For practical implementations, it is possible to, e.g., select the smallest configuration by lexicographic order. Then, the ansatz defined by

$$\psi_{\boldsymbol{\theta}}(\boldsymbol{s}) = \tilde{\psi}_{\boldsymbol{\theta}}(\text{can}(\boldsymbol{s})) \tag{5.14}$$

is symmetry invariant (5.5) as an immediate consequence of Eq. (5.13).

To construct an ansatz that transforms according to a specific irrep ρ [Eq. (5.8)], Eq. (5.14) needs to be extended so that

$$\psi_{\boldsymbol{\theta}}^{\rho}(\boldsymbol{s}) = r_{\rho}(\boldsymbol{s}) \tilde{\psi}_{\boldsymbol{\theta}}(\text{can}(\boldsymbol{s})). \tag{5.15}$$

Thus, the problem of enforcing the correct transformation properties is reduced to appropriately constructing $r_\rho(\boldsymbol{s})$. See Ref. [92] for a description of this construction for a finitely generated Abelian group[20]. A closely related approach specifically for the flux symmetries of the HKM is described in Section 5.2.3.

5.1.4 Group-convolutional neural networks

Group-convolutional neural networks (GCNNs) can be seen as an extension of the symmetry-projection scheme discussed in Section 5.1.2 to deep networks. GCNNs were introduced by Cohen and Welling [170] as a generalization of CNNs beyond translation symmetries and first applied to NQS by Roth and MacDonald [171]. The main idea of a GCNN is to implement network layers that respect the group structure of a symmetry group G in such a way that transformed inputs are mapped to equivalently transformed outputs; this property is known as *equivariance*.

Specifically, an equivariant NQS ansatz can be constructed as follows (we follow the presentation of Refs. [88, P4]). As before, we consider a group acting on the configurations $\boldsymbol{s} \in \mathcal{S}$ by permutation of the indices, which we also denote as $(g.\boldsymbol{s})_i = s_{g(i)}$.

First, a linear *embedding layer* is applied to input configuration $\boldsymbol{s}$ to generate the features $\boldsymbol{y} = \{\boldsymbol{y}_g\}_{g\in G}$, indexed by group elements and with each $\boldsymbol{y}_g \in \mathbb{C}^M$, as

$$\boldsymbol{y}_g(\boldsymbol{s}) = \Phi(\boldsymbol{v}_g(\boldsymbol{s})), \qquad v_{g,i}(\boldsymbol{s}) = \sum_{j=1}^{N} K_{i,g^{-1}(j)} s_j + b_i. \tag{5.16}$$

Here, Φ is an activation function, $\mathbf{K} \in \mathbb{C}^{M\times N}$ a weight matrix, and $\boldsymbol{b} \in \mathbb{C}^M$ is a bias vector in analogy to a standard linear layer (Section 3.2.1). By construction, the output of the embedding layer satisfies the equivariance property

$$\boldsymbol{y}_g(h.\boldsymbol{s}) = \boldsymbol{y}_{hg}(\boldsymbol{s}) \tag{5.17}$$

for all $g, h \in G$. In a deep GCNN, these group-indexed feature vectors are then propagated through multiple *equivariant layers* that preserve this property. An equivariant mapping between features $\boldsymbol{y}$ and $\{\boldsymbol{z}_g\}_{g\in G}$, $\boldsymbol{z}_g \in \mathbb{C}^{M'}$, has the form

$$\boldsymbol{z}_g(\boldsymbol{y}) = \Phi(\boldsymbol{v}'_g(\boldsymbol{y})), \qquad v'_{g,i}(\boldsymbol{y}) = \sum_{h\in G}\sum_{j=1}^{M} W_{i,j}^{(g^{-1}h)} y_{h,j} + b'_i \tag{5.18}$$

where, again, Φ is the activation function, each $\mathbf{W}^{(g)} \in \mathbb{C}^{M'\times M}$ an independent weight matrix, and $\boldsymbol{b}' \in \mathbb{C}^{M'}$ a bias vector. Writing $h.\boldsymbol{y} = \{\boldsymbol{y}_{hg}\}_{g\in G}$, it is apparent that Eq. (5.18) indeed retains the equivariance property

$$\boldsymbol{z}_g(h.\boldsymbol{y}) = \boldsymbol{z}_{hg}(\boldsymbol{y}) \tag{5.19}$$

as required. The output of an equivariant layer (or a composition of multiple such layers) can finally be summed analogously to Eq. (5.7) to obtain a network output and, thus, a quantum state that transforms

[20] An Abelian group G is said to be finitely generated by a set of generators $\{g_i\}_{i=1}^{n_G}$ if any group element can be decomposed as a product of generators $g_{\boldsymbol{n}} = \prod_{i=1}^{n_G} g_i^{n_i}$. In this case, the integer powers $\boldsymbol{n} \in \mathbb{N}^{n_G}$ determine the group element.

according to a desired irrep ρ. Specifically, if $\boldsymbol{f} = \{f_g\}_{g \in G}$ denotes the output of the final equivariant layer (with feature dimension one, i.e., $f_g \in \mathbb{C}$),

$$\ln \psi_{\boldsymbol{\theta}}^{\rho} = \sum_{g \in G} \chi_{\rho}^{*}(g) \exp(f_g), \tag{5.20}$$

yields the desired log-probability amplitude.

5.2 Symmetry-projected NQS for the honeycomb Kitaev model

This section is based on a project I have worked on in collaboration with Martin Claassen (Department of Physics and Astronomy, University of Pennsylvania) and Michael A. Sentef which has not yet been independently published, and discusses its motivation, approach, and presents a selection of preliminary results.

Declaration of contributions. I have selected the variational ansatz, implemented the model and symmetry-projection scheme for the flux constraints based on NetKet 3, performed simulations and data analysis, created the figures, and written the text of this section.

The honeycomb Kitaev model (HKM) is an important setting for studying quantum spin liquid (QSL) physics [220]. It was originally proposed by Kitaev [24] as a simple two-dimensional spin model which hosts anyonic excitations (i.e., particles with neither Bose nor Fermi statistics [221, 222]). The physics of the HKM play a vital role in the study of topological quantum computing [220, 223].

Kitaev-like interactions are present in materials with strong spin-orbit coupling, such as the transition metal compounds Na_2IrO_3, Li_2IrO_3, and α-$RuCl_3$ [220]. While many ground state properties of the HKM, including the ground state energy for a given flux sector (Section 5.2.1), can be analytically predicted using the Majorana fermion representation introduced by Kitaev [24], this is no longer true once the Hamiltonian is extended to include Heisenberg or off-diagonal exchange interactions which are relevant for the physics of real materials [224–226]. Recent progress in learning QSL phases using neural quantum states in other spin models, particularly in frustrated quantum magnets [86, 88], makes it a natural question whether an NQS ansatz can learn the QSL phase of the HKM ground state in the spin representation as well. If this is the case, one might hope to apply such an NQS ansatz to an extended Kitaev-Heisenberg model and, particularly, study whether it is possible to reach the Kitaev QSL phase by nonequilibrium processes [227, 228].

Here, we discuss the approach and preliminary results of a VMC study of the HKM using a GCNN ansatz. We find that the QSL ground state of the HKM is challenging to learn with this ansatz, but convergence can be improved by enforcing additional symmetries of the model. Specifically, we apply a scheme to project the variational trial states to eigenspaces of the plaquette flux operators (Section 5.2.3), which are integrals of motion of the Kitaev Hamiltonian, and find that projecting the trial state to the ground state flux eigenspace for a large fraction of plaquettes does improve convergence in small systems. However, because the computational complexity of our symmetry projection scheme scales exponentially with the number of plaquette operators considered, the question of the scalability of this approach is still open. We discuss potential remedies and alternative variational ansätze in the discussion section (Section 5.2.5).

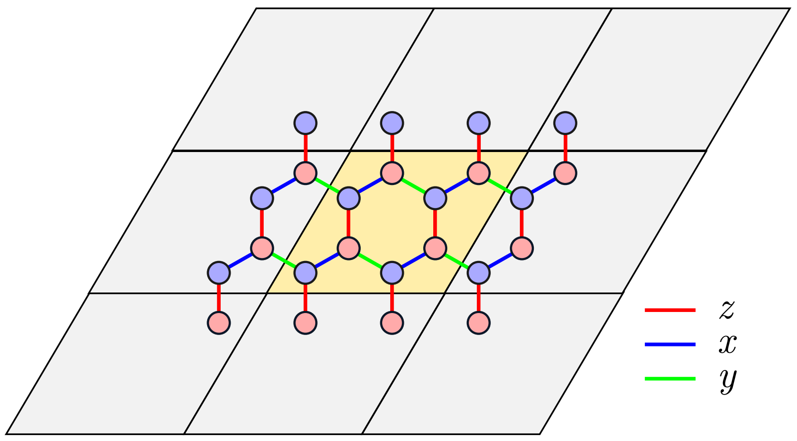


Figure 5.1 | Lattice structure of the Kitaev honeycomb model. The lattice is bipartite, with each unit cell containing two sites connected by a z bond. The x and y bonds connect sites in adjacent unit cells. This figure depicts a simulation cell consisting of 2×2 unit cells (highlighted) with periodic boundary conditions in both lattice directions.

5.2.1 Honeycomb Kitaev model

The HKM is defined on the honeycomb lattice with a partition of the nearest-neighbor bonds into three classes called x, y, and z bonds (Fig. 5.1). We denote these bonds by $\mathcal{E}$ and the assigned class of each edge $(i, j) \in \mathcal{E}$ by $\gamma(i, j) \in \{x, y, z\}$. The Hamiltonian is then defined by assigning an Ising coupling term to each bond in either the x, y, or z basis according to the class of the bond. Explicitly, the Hamiltonian is [24]

$$\hat{H}_K = \sum_{(i,j)\in\mathcal{E}} J_{\gamma(i,j)} \hat{K}_{(i,j)}, \qquad \text{where} \qquad \hat{K}_{(i,j)} = \boldsymbol{\sigma}_i^{\gamma(i,j)} \boldsymbol{\sigma}_j^{\gamma(i,j)}. \tag{5.21}$$

In the following, we only consider the case of uniform couplings $J_{\gamma(i,j)} = -1$, corresponding to the gapless QSL phase of the HKM. An important property of the HKM is the existence of a large set of integrals of motion, i.e., operators that commute with the Hamiltonian and thus partition the Hilbert space into subspaces simultaneously diagonalizable with $\hat{H}$. Specifically, let p denote a hexagonal plaquette, ordered in such a way that the edges are of class z, x, y, z, x, y (in that order) along the path, and define the *flux operator*

$$\hat{W}_p = \prod_{(i,j)\in p} \hat{K}_{(i,j)}. \tag{5.22}$$

The flux operators commute with the Hamiltonian and among themselves, $[\hat{H}_K, \hat{W}_p] = [\hat{W}_p, \hat{W}_q] = 0$, which implies the existence of a simultaneous eigenbasis in which all operators are diagonalized. Additionally, the flux operators square to the identity, $\hat{W}_p^2 = 1$, and have vanishing trace, $\operatorname{tr} \hat{W}_p = 0$. Thus, their possible eigenvalues are ± 1, with equal multiplicity. If $\langle \hat{W}_p \rangle = -1$, the plaquette p is said to host a vortex; conversely, if $\langle \hat{W}_p \rangle = +1$, p is called vortex-free [24].

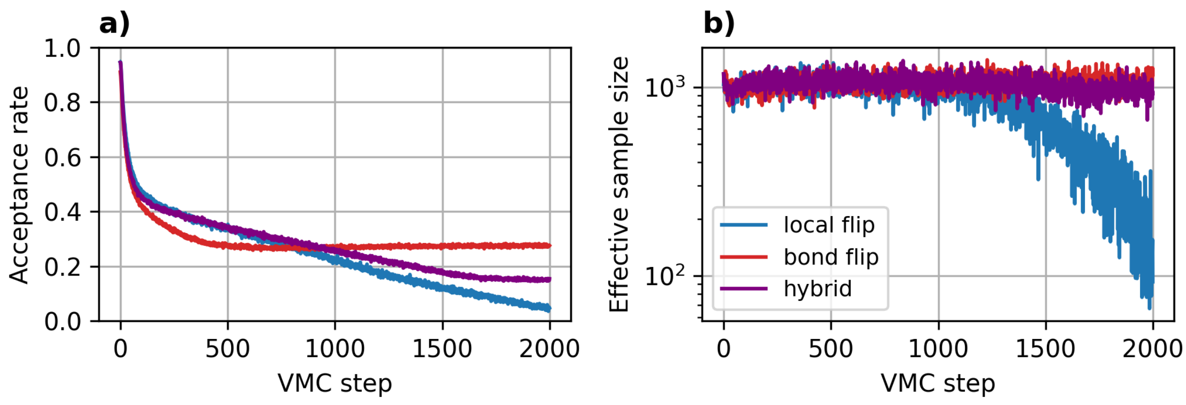


Figure 5.2 | a) Acceptance rate of Metropolis-Hastings updates and **b)** estimated effective sample size (Section 3.3.3) over the course of a VMC optimization run for $\hat{H}_K$ using a GCNN ansatz for different MCMC proposal rules. The blue curves correspond to local spin flip proposals, the red curves to proposals of simultaneous flips of two spins connected by an HKM bond, and the purple curves to a hybrid rule which proposes one of the previous updates randomly with equal probability. For the bond-flip rule, configurations are restricted to the even-magnetization sector $\{\boldsymbol{s} \in \mathcal{S} \mid \Pi(\boldsymbol{s}) = +1\}$. In all cases, VMC estimates were performed using $N_{\mathrm{MC}} = 10^3$ samples.

As observed by Kitaev, the ground state of the model is unique and part of the simultaneous flux eigenspaces corresponding to $\langle \hat{W}_p \rangle = +1$ for all plaquettes p (in other words, the ground state is contained in the vortex-free sector) based on a theorem by Lieb [24, 229]. However, this argument does not generally apply to finite-size systems. While in those cases, the Hamiltonian is still simultaneously diagonalizable with all flux operators (since this is a consequence of the commutation relations), the ground state is not restricted to the vortex-free sector and can be degenerate, allowing the flux expectation values $\langle \hat{W}_p \rangle$ to potentially vary within the ground state manifold. The flux phase and degeneracy of the ground state can depend on the size and boundary conditions of a finite honeycomb lattice [230, 231]. For example, the ground state of a finite lattice with 2×2 unit cells with periodic boundaries is non-degenerate but part of the $\langle \hat{W}_p \rangle = -1$ sector. Therefore, when the flux quantum numbers are used to assess convergence to the ground state in finite systems, care must be taken to distinguish an unconverged state from one in a superposition of different flux sectors.

5.2.2 Magnetization parity

In the Heisenberg model, the ground state is part of the zero-magnetization eigenspace, which simplifies VMC ground state optimization. This property does not apply to the HKM, since $[\hat{H}_K, \hat{M}_z] \neq 0$ [where $\hat{M}_z$ is defined by Eq. (5.3)]. However, the *magnetization parity*

$$\hat{\Pi} = (-1)^{\hat{M}_z} = (-1)^{N/2} \prod_{i=1}^{N} \hat{\sigma}_i^z, \tag{5.23}$$

which is diagonal in the computational basis with $\Pi(\boldsymbol{s}) = \langle \boldsymbol{s} | \hat{\Pi} | \boldsymbol{s} \rangle = (-1)^{\sum_{i=1}^{N} \frac{s_i}{2}}$, is still a conserved quantity, $[\hat{H}_K, \hat{\Pi}] = 0$, and, furthermore, $[\hat{W}_p, \hat{\Pi}] = 0$ holds for all plaquettes p. Thus, $\hat{\Pi}$ can be used as a diagonal constraint to restrict the computational basis and improve the efficiency of Monte Carlo sampling (Section 5.1.1). The eigenvalues of Π are ± 1 and, therefore, only two sectors (corresponding to even and odd magnetization) exist.

For illustration, Fig. 5.2 shows the dependency of the acceptance rate and effective sample size (Section 3.3.3) for a VMC energy estimate for $\hat{H}_K$ during a ground-state optimization. These diagnostics show that, for single spin-flip updates in the unconstrained configuration space, the efficiency of the MCMC estimation decreases significantly over the course of the simulation. At the same time, sampling within a single Π symmetry sector (which is conserved by a proposal rule that only suggests simultaneous flips of two spins at adjacent sites) retains a reasonable acceptance rate and a high effective sample size throughout the optimization. Using a hybrid proposal rule that at each step proposes one of the previous updates randomly with equal probability still results in a higher acceptance rate and effective sample size in the unconstrained configuration space.

The magnetization-parity symmetry is particularly useful because it applies to both the HKM and the Heisenberg model on the same lattice and can, thus, be used in simulations of the combined Kitaev-Heisenberg model. Further extensions, such as the off-diagonal exchange interaction required for approaching realistic materials [224–226] break this symmetry. However, the application of a hybrid sample rule can still improve MCMC performance in this case, as demonstrated above.

5.2.3 Imposing flux constraints

This section discusses the implementation of a constrained NQS ansatz for specific flux symmetry sectors, which is based on the same ideas as the symmetry-projection approach proposed in Ref. [92] and described in Section 5.1.3.

Single plaquette

Consider a single flux operator $\hat{W}_p$ for an arbitrary plaquette p with sites denoted by $p_1, p_2, \dots, p_6$. The operator $\hat{W}_p$ acts on a spin-1/2 basis state $|\boldsymbol{s}\rangle = |s_1 \dots s_N\rangle$, $\boldsymbol{s} \in \{\pm 1\}^N$, as

$$\hat{W}_p |\boldsymbol{s}\rangle = \sigma_p(\boldsymbol{s}) |w_p(\boldsymbol{s})\rangle. \tag{5.24}$$

where the sign

$$\sigma_p(\boldsymbol{s}) = -s_{p_2} s_{p_3} s_{p_5} s_{p_6} \tag{5.25}$$

depends on the product of all spins connected by x bonds and where the *exchange mapping*

$$w_p(\boldsymbol{s})_i = \begin{cases} -s_i, & i \in \{p_1, p_2, p_4, p_5\}, \\ s_i, & \text{otherwise.} \end{cases} \tag{5.26}$$

flips the spins of all sites connected by z bonds (Fig. 5.3). The exchange mapping is an involution, i.e., $w_p(w_p(\boldsymbol{s})) = \boldsymbol{s}$. Therefore, the action of the flux operator $\hat{W}_p$ partitions the set of basis configurations into 2^{N-1} equivalence classes $\mathcal{C}_j = \{\boldsymbol{s}^j, \bar{\boldsymbol{s}}^j\}$, each with two elements connected by exchange mapping so that $w_p(\boldsymbol{s}^j) = \bar{\boldsymbol{s}}^j$ and, conversely, $\boldsymbol{s}^j = w_p(\bar{\boldsymbol{s}}^j)$.

For a quantum state $|\psi\rangle$ to be contained in the $\nu \in \{\pm 1\}$ eigenspace of $\hat{W}_p$, it must satisfy

$$\psi^\nu(\boldsymbol{s}) = \nu\, \sigma_p(\boldsymbol{s})\, \psi^\nu(w_p(\boldsymbol{s})) \tag{5.27}$$

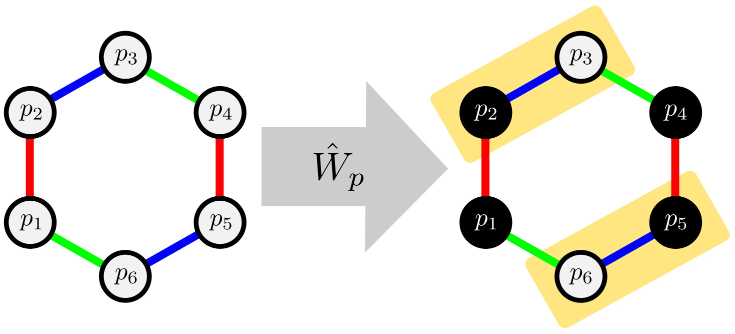


Figure 5.3 | Action of $\hat{W}_p$ on the spin configuration of plaquette p. The spins on sites connected by z bonds (red lines) are swapped, and the state acquires a sign determined by the product of sites connected by x bonds (blue lines, highlighted).

and, hence, any state satisfying Eq. (5.27) is an eigenstate of $\hat{W}_p$ with eigenvalue ν. This condition can be enforced analogously to the symmetry condition of Eq. (5.8). In particular, Kurita et al. [232] have applied the symmetry-projection approach based on superposition (Section 5.1.2) to the HKM, obtaining the symmetrized ansatz

$$|\psi^{\nu}_{\boldsymbol{\theta}}\rangle = \frac{1}{2}(1 + \nu \hat{W}_p)|\tilde{\psi}_{\boldsymbol{\theta}}\rangle \tag{5.28}$$

from an unconstrained trial state $|\tilde{\psi}_{\boldsymbol{\theta}}\rangle$. This ansatz satisfies Eq. (5.27) by construction. Alternatively, in analogy with the approach discussed in Section 5.1.3, a symmetrized trial state can be constructed by first selecting a representative spin $\mathrm{can}(\boldsymbol{s})$ from each class $\mathcal{C}_j$ so that $\mathrm{can}(\boldsymbol{s}) = \mathrm{can}(\boldsymbol{w}_p(\boldsymbol{s}))$ and, then, using this state as the canonical input to the ansatz wave function,

$$\psi^{\nu}_{\boldsymbol{\theta}}(\boldsymbol{s}) = \nu\, \sigma_p(\boldsymbol{s})\, \tilde{\psi}_{\boldsymbol{\theta}}(\mathrm{can}(\boldsymbol{s})). \tag{5.29}$$

Compared to Eq. (5.28), this saves one evaluation of the ansatz. The canonical representative can be selected arbitrarily. Since $\boldsymbol{s}$ and $w_p(\boldsymbol{s})$ always differ by the sign of s_{p_0}, it is, e.g., possible to select the configuration with $s_{p_0} = +1$ as canonical for a single plaquette.

Multiple plaquettes

For multiple plaquettes, a symmetrized trial state can be constructed by superposition using a product of the operators (5.28). To fix the flux quantum numbers of the plaquettes in the set P to the sectors $\boldsymbol{\nu} = \{\nu_p\}_{p\in P}$, the projection operator is [232]

$$|\psi^{\boldsymbol{\nu}}_{\boldsymbol{\theta}}\rangle = \frac{1}{2^{|P|}} \prod_{p\in P} (1 + \nu_p \hat{W}_p)|\tilde{\psi}_{\boldsymbol{\theta}}\rangle. \tag{5.30}$$

This product can be expanded into a sum of $2^{|P|}$ terms, and the ansatz thus requires the corresponding number of evaluations of the unconstrained trial state to obtain a symmetrized probability amplitude, limiting the amount of flux quantum numbers that can be fixed at once.

While this exponential complexity cannot be circumvented simply by using an alternative symmetry-projection scheme, the amount of required wave function evaluations can be reduced, which is beneficial in practical VMC calculations. Each flux operator $\hat{W}_p$ has an associated sign function σ_p and an exchange mapping w_p defined as in Eqs. (5.25) and (5.26). Consider the group G generated by the exchange mappings $\{w_p\}_{p\in P}$ acting on spin configurations. Since $w_p w_p = \mathrm{id}$, any element of G can be represented as $g_{\boldsymbol{n}} = \prod_{p\in P} w_p^{n_p}$ where $n_p \in \{0,1\}$. Note that in the presence of periodic boundary conditions, not all distinct $\boldsymbol{n}$ determine distinct operations. However, the order of G is still exponential in $|P|$ [233]. As before, the orbit of a configuration $\boldsymbol{s}$ under G is denoted by $\boldsymbol{s}^G$, and $\mathrm{can}(\boldsymbol{s})$ denotes a canonical configuration selected from each orbit (5.13). Then, a symmetrized state can be constructed from an unconstrained ansatz $\tilde{\psi}_{\boldsymbol{\theta}}$ by

$$\psi_{\boldsymbol{\theta}}^{\boldsymbol{\nu}}(\boldsymbol{s}) = r^{\boldsymbol{\nu}}(\boldsymbol{s})\,\tilde{\psi}_{\boldsymbol{\theta}}(\mathrm{can}(\boldsymbol{s})). \tag{5.31}$$

which corresponds to the symmetry sector $\hat{W}_p|\psi_{\boldsymbol{\theta}}^{\boldsymbol{\nu}}\rangle = \nu_p|\psi_{\boldsymbol{\theta}}^{\boldsymbol{\nu}}\rangle$. The factor $r^{\boldsymbol{\nu}}(\boldsymbol{s}) \in \{\pm 1\}$ needs to adjust the sign of the symmetrized ansatz, accounting for the sign flips σ_p and quantum numbers ν_p. It can be determined by identifying the exponents $k_p(\boldsymbol{s}) \in \{0,1\}$ so that $g_{\boldsymbol{k}(\boldsymbol{s})}(\boldsymbol{s}) = \mathrm{can}(\boldsymbol{s})$ [i.e., identifying the group element that connects $\boldsymbol{s}$ and $\mathrm{can}(\boldsymbol{s})$] and tracking the sign changes induced by applying this operation. In a numerical implementation, the sign changes $r^{\boldsymbol{\nu}}$ corresponding to each group element can be computed once and then cached before the start of the VMC simulation. While this approach allows for an efficient implementation of Eq. (5.31) [requiring only one evaluation of $\tilde{\psi}_{\boldsymbol{\theta}}$, compared to the $|G|$ evaluations of Eq. (5.30)], the exponential growth of the group order still affects startup and memory costs and, thus, limits the number of flux quantum numbers that can be fixed.

5.2.4 Results

Here we present some preliminary results of applying our symmetrized NQS ansatz based on Eq. (5.31) to the HKM, starting with a single hexagonal plaquette followed by results for extended honeycomb lattices.

Single plaquette

As a simple proof of concept, we first consider a single, isolated hexagonal plaquette of the HKM with $N = 6$ sites. On this geometry, the HKM has a four-fold degenerate ground state subspace, and all ground states are part of the $\langle\hat{W}\rangle = +1$ eigenspace, as verified by ED (Fig. 5.4). We optimize an RBM ansatz with complex weights and hidden unit density $\alpha = N_h/N = 1$ (i.e., the dimension of the hidden layer is equal to the system size) of the form of Eq. (3.20) to minimize the energy using VMC with SR.

The results are shown in Fig. 5.5. Both the unconstrained and the symmetrized RBM can learn the Kitaev ground state for a single hexagon. However, convergence is slower for the unconstrained ansatz, and the final energy (averaged over the last 100 samples and five independent runs) is $E = -3.999(2)$ for the RBM versus $E = -3.9999(2)$ for the symmetrized RBM at otherwise equal optimization parameters. This shows that the symmetry constraints, though not crucial in this six-site toy system, can indeed improve ground-state convergence. Both networks are initialized with random weights from a complex normal distribution. For the unconstrained ansatz, this corresponds to a trial state in a superposition of both symmetry sectors with $\langle\hat{W}\rangle \approx 0$, and it approaches the correct symmetry sector over the course of the simulation (Fig. 5.5b). While the system under consideration is small, the RBM is still able to learn the ground state with compression compared to the full ED state[21].

[21]The variational state with weights and hidden bias contains $M = \alpha N(N+1) = 42$ parameters, corresponding to a

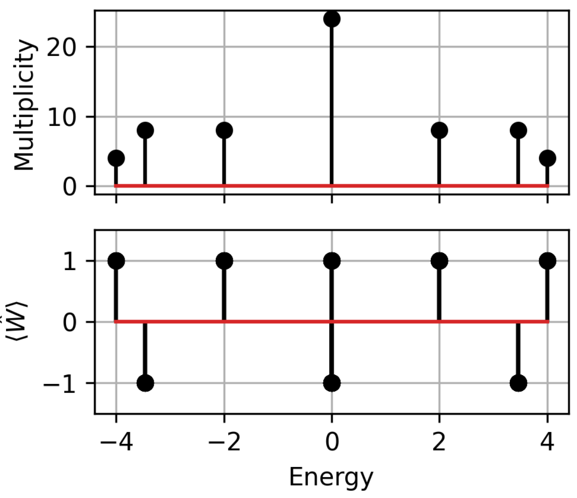


Figure 5.4 | Multiplicity of energy eigenvalues (upper panel) and corresponding values of $\langle \hat{W} \rangle$ (lower panel) obtained from an exact simultaneous diagonalization of the HKM Hamiltonian (5.21) and flux operator $\hat{W}$ (5.22) for a single six-site hexagonal plaquette. In this case, the HKM ground state subspace is four-fold degenerate and fully part of the $\langle \hat{W} \rangle = +1$ sector.

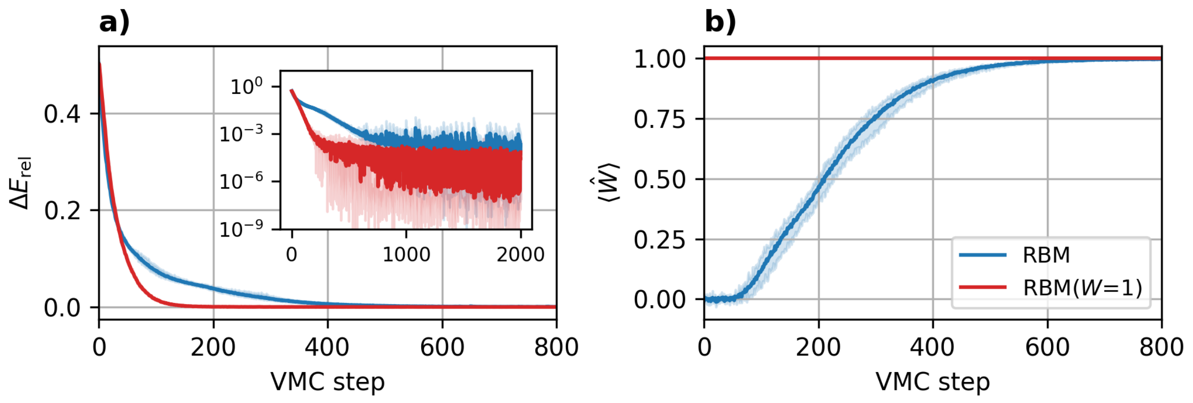


Figure 5.5 | **a)** Relative error $\Delta E_{\text{rel}} = |\langle \hat{H}_K \rangle - E_0|/E_0$ of the variational energy estimate compared to the ED result E_0 and **b)** flux expectation value $\langle \hat{W} \rangle$ over the course of a VMC ground state optimization of the HKM Hamiltonian on a single plaquette. Simulations have been performed for an unconstrained RBM (blue), and a RBM projected to the $\langle \hat{W} \rangle = +1$ sector via the ansatz described in Section 5.2.3 (red). Five independent VMC runs have been performed for each ansatz; the solid line indicates the mean, and the shaded background indicates the spread between maximum and minimum values over these runs. Both panels show the first 800 of the 2000 optimization steps performed in total; the inset of Panel a) displays the same data on a logarithmic scale over the full range of steps.

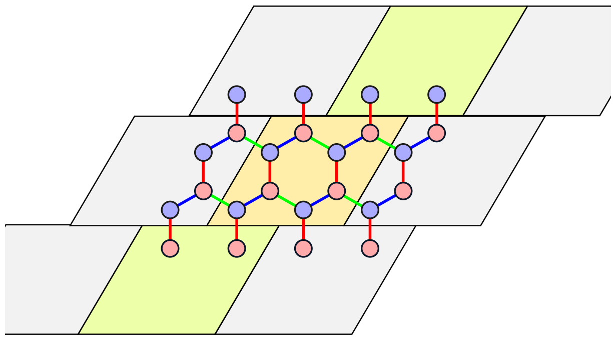


Figure 5.6 | Lattice structure of the HKM with 2×2 unit cells and twisted periodic boundary conditions. The simulation cells are shifted relative to each other by half their extent at the y direction boundary.

Honeycomb lattice

We now turn to the HKM on the extended honeycomb lattice (Fig. 5.1). Motivated by the positive effects of enforcing lattice symmetries on ground state convergence [82, 88, 171], we choose a translation-invariant GCNN as the basis of our ansatz[22] [171] (Section 5.1.4). The flux constraints are imposed after evaluation of the GCNN using the symmetry-projection approach of Section 5.2.3. Here, we include results for honeycomb lattices of 3×3 and 4×4 unit cells.

In order to reduce the influence of finite-size effects, we impose twisted periodic boundary conditions [24, 230, 231]. Specifically, our lattice vectors are shifted by $\lfloor L_y/2 \rfloor$ when crossing the periodic boundary in y direction (Fig. 5.6). Empirically, we have found this to improve the stability of our VMC optimization runs. Furthermore, for small square systems where we could verify the ground state sector via ED, we have found that this shift lifts the ground-state degeneracy and selects the ground state in the $\langle \hat{W} \rangle = +1$ sector, which is consistent with the analytical arguments of Ref. [231].

Figure 5.7 shows VMC results for the HKM on a lattice with 3×3 unit cells ($N = 18$ sites). We chose a GCNN ansatz with four hidden layers, each with four features per group element. We further restrict Monte Carlo sampling to the even-magnetization sector (Section 5.2.2). The ansatz is then projected to the vortex-free flux sector for a varying number of plaquettes using the approach of Section 5.2.3. In the following, let P denote the set of all plaquettes for which the flux quantum number is constrained by the variational ansatz, and $\bar{P}$ its complement for which the quantum numbers are unconstrained. We further denote by $W_{\bar{P}} = |\bar{P}|^{-1} \sum_{\bar{p} \in \bar{P}} \langle \hat{W}_{\bar{p}} \rangle$ the average flux expectation over the unconstrained plaquettes.

As for the single plaquette case above, the initial state (which corresponds to a GCNN ansatz with weights randomly drawn from a Gaussian distribution) is in a superposition of flux sectors with $W_{\bar{P}} \approx 0$. While the fluxes trend towards the $+1$ sector over the course of the VMC optimization already for the fully unconstrained GCNN (i.e., with $|P| = 0$), the correct sector is not reached. Instead, the simulation converges to a state with $W_{\bar{P}} \approx 0.68$. Imposing flux constraints for one plaquette leads to a faster increase in $W_{\bar{P}}$. However, while the flux expectation value for an unconstrained plaquette increases systematically with $|P|$ (Fig. 5.7d), the variational energy of the state with several constrained plaquettes increases at first, compared to the fully unconstrained state (Fig. 5.7c). This changes once the majority of quantum numbers are fixed. Specifically, $|P| = 6$ achieves a lower variational energy than the

compression ratio of $M/2^6 \approx 0.66$.

[22] Note that we only enforce translation invariance in the GCNN, not invariance under the full space group (including rotations) of the lattice.

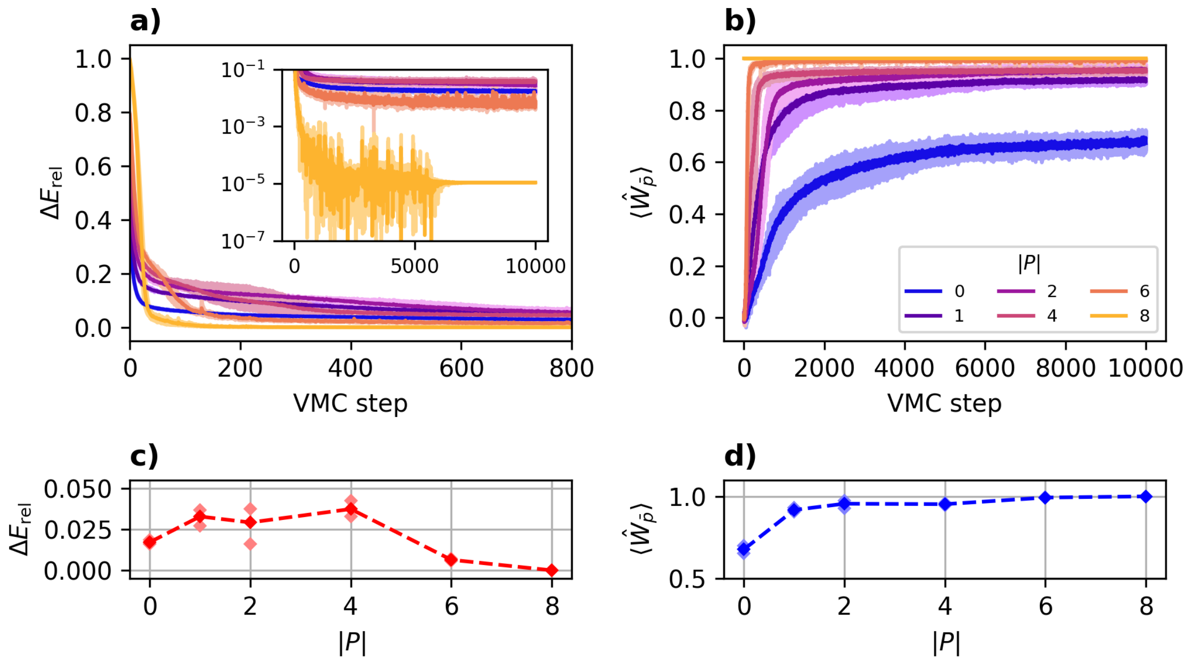


Figure 5.7 | a) Relative error $\Delta E_{\mathrm{rel}} = |\langle \hat{H}_K \rangle - E_0|/E_0$ of the variational energy estimate compared to the ED result E_0 and **b)** flux expectation value $\langle \hat{W}_{\bar{p}} \rangle$ for an unconstrained plaquette $\bar{p} \in \bar{P}$ over the course of a VMC ground state optimization. Simulations have been performed on a 3×3 honeycomb lattice with twisted periodic boundary conditions (Fig. 5.6) using a GCNN ansatz for a varying number of fixed flux sectors $|P|$. The solid curves show the mean values over three independent optimization runs, while the shaded area indicates the span between minimum and maximum value over these runs. Panels a) shows the first 800 of 10 000 optimization steps performed in total; the inset displays the same data on a logarithmic scale over the full range of steps.
c) Final relative energy error and **d)** flux expectation value averaged over the last 500 steps. The curves and solid markers indicate the mean, and the light markers the minimum and maximum values over the simulation runs.

unconstrained state, and for $|P| = 8$, which corresponds to constraining all plaquettes[23], convergence to the true ground state is achieved with a relative error of ΔE_{rel} of the order of 10^{-5} (Fig. 5.7a).

Figure 5.8 shows results for the same approach on a larger system of 4×4 unit cells ($N = 32$ sites). These results indicate a qualitatively similar behavior, where the flux expectation value over the unconstrained plaquettes increases with the number of fixed plaquettes (Fig. 5.8d). Furthermore, the initial increase in energy for $1 \leq |P| \leq 4$ compared to the fully unconstrained state after a fixed number of optimization steps is also observed (Fig. 5.8c), as is the improvement in energy for a larger number of fixed fluxes.

5.2.5 Discussion and outlook

We have presented a symmetry-projection scheme for flux symmetries in the HKM, combining the approaches of Refs. [92, 232] with a GCNN ansatz. Compared to an unconstrained ansatz, we have observed that the symmetry projection approach achieves improved ground state convergence when used with an RBM ansatz for a single-hexagon toy model. Furthermore, imposing the flux symmetries for multiple hexagons on an extended lattice also improves convergence. However, in our simulations so far, this has only occurred once a sufficient number of all possible flux quantum numbers have been fixed. This potentially limits the scalability of the approach because the size of the flux symmetry group and, thus, the computational complexity required to apply the symmetrization schemes, grows exponentially in the number of fixed quantum numbers. Further numerical experiments are required to systematically determine the necessary scaling.

Our current results do not rule out that obtaining more reliable ground-state convergence with an unaltered approach is possible by tuning the optimization hyperparameters. The representation capabilities of the ansatz are likely not the limiting factor since, e.g., Ref. [83] has obtained an accurate ground state using an unconstrained RBM ansatz in the 3×3 HKM[24]. Rather, the main challenge appears to be achieving reliable convergence to the correct symmetry sector in larger systems and the significant dependency of VMC results on the lattice geometry, particularly the choice of boundary conditions.

While the ansatz of Kurita et al. [232] was similarly limited by the exponential growth of the symmetry group, the optimized results they have obtained with the pair-product ansatz exhibit a more systematic improvement of the energy with the number of enforced symmetries, and such pair-product wave functions have been successfully combined with an RBM in previous work [73]. In initial explorations, we have found that a pair-product ansatz constrained using our symmetry-projection approach described here shows improved ground state convergence as well; combining this approach with a (potentially unconstrained) NQS ansatz may yield a more flexible ansatz capable of being applied beyond the pure HKM, especially to the Kitaev-Heisenberg and further extended models relevant to realistic materials that initially motivated our explorations [227, 228].

Acknowledgments. We acknowledge helpful discussions with Johannes Knolle and Attila Szabó. The NQS simulation code is based on NetKet 3 [P4] using JAX [234, 235]. Computational resources have been provided by Flatiron Institute, a division of The Simons Foundation.

[23] In a honeycomb lattice with M unit cells and periodic boundary conditions, only $M - 1$ plaquette operators are independent due to the relation $\prod_{p \in P \cup \bar{P}} \hat{W}_p = 1$ [232]. Thus, constraining $M - 1$ plaquettes already fixes the flux quantum number for the remaining one as well.

[24] The 3×3 calculations of Ref. [83] have been performed by circumventing VMC optimization using full-state evaluation of variational energy and gradients, which is not applicable to system sizes beyond ED. Therefore, whether the RBM results of Ref. [83] scale to larger lattices is not yet clear.

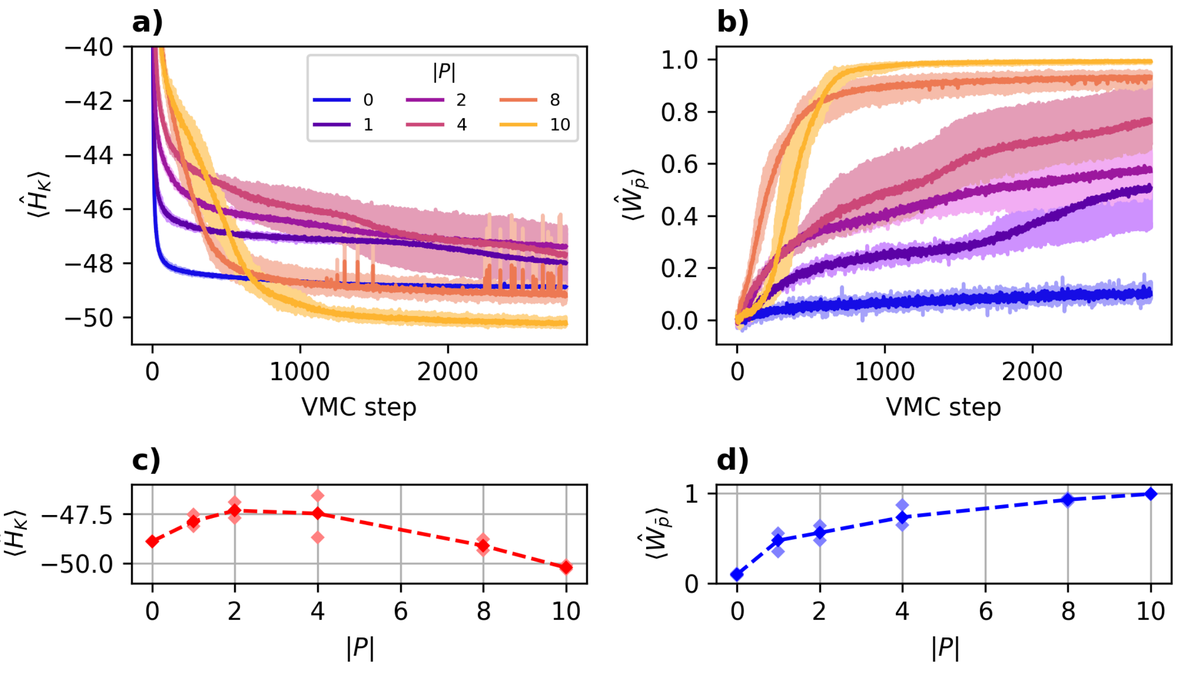


Figure 5.8 | **a)** Variational energy $\langle \hat{H}_K \rangle$ and **b)** flux expectation value $\langle \hat{W}_{\bar{p}} \rangle$ for an unconstrained plaquette $\bar{p} \in \bar{P}$ over the course of a VMC ground state optimization. The simulations are done on a 4×4 honeycomb lattice with twisted periodic boundary conditions. Results are shown for 2800 VMC optimization steps. Otherwise, the setup is the same as in Fig. 5.7.
c) Final variational energy and **d)** flux expectation value averaged over the last 500 steps for three independent runs. The curves and solid markers indicate the mean, and the light markers the minimum and maximum values over the runs.

6 Representability of ground states in the Sachdev-Ye-Kitaev model

Understanding the representation capabilities of NQS is of central importance for the development of the field. While universal approximation theorems [140–143, 165] guarantee that any quantum wave function can be approximated to an arbitrary degree in the limit of infinite network size, this does not necessarily imply efficient representability, i.e., the ability of an ansatz to express relevant quantum states with *exponential* compression compared to the full state vector.

Due to the wide variety of architectures that can be referred to as NQS (Section 3.2.3), it is necessary to focus on specific architectures to provide concrete answers to the question of representability. It has been shown that the RBM can represent some volume-law entangled states due to its nonlocal connectivity between layers and that this scaling is already realized for an RBM with random weights [101, 102]. RBM quantum states have also been shown to be able to efficiently represent chiral topological states in two dimensions [103–105]. Furthermore, Ref. [120] has proven that deep networks can approximate any MPS and a subset of TNS[25] to arbitrary precision with polynomial scaling of the network size.

However, besides the theoretical capability of an ansatz to represent states of interest, it is also essential to consider the difficulty of actually learning a target state. For NQS, the expressivity of the ansatz comes at the cost of a "rugged" energy landscape [214], which can make it difficult to find global energy minima even when they can be represented. As discussed in Section 3.3.2, this is particularly true for target states that exhibit a complicated sign or phase structure [88, 194, 195]. Nonetheless, several works have applied supervised learning (Section 3.2.2) using ED amplitudes for the purpose of testing whether a given state can be represented [106, P1]. Notably, Lin and Pollmann [106] have analyzed the scaling behavior of a NAQS ansatz when learning amplitudes of time-evolved states in a one-dimensional transverse-field Ising chain and found an exponential scaling of the required network size over time. While negative results cannot rule out that a more efficient representation can be found using a different (or more fine-tuned) optimization scheme, direct training on ED amplitudes does remove several potential sources of error present in VMC energy optimization.

In the project presented in this chapter, we further explore the learning capabilities of NQS in a challenging setting. Specifically, we chose the Sachdev-Ye-Kitaev (SYK) model [236, 237]. The SYK model is a disordered model of strongly-correlated fermions with all-to-all interactions between N sites, with a Hamiltonian of the form

$$\hat{H}_{\mathrm{SYK}} \propto \sum_{ijkl} J_{ij;kl} \hat{c}_i^\dagger \hat{c}_j^\dagger \hat{c}_k \hat{c}_l. \tag{6.1}$$

The couplings are random variables drawn from a Gaussian unitary ensemble (see [P2]). The ground

[25] Specifically, the NQS ansatz introduced in Ref. [120] can efficiently approximate those projected entangled pair states [127] for which an efficient approximate or exact contraction algorithm exists.

state of the SYK model is highly entangled and features volume-law entanglement scaling. Despite the apparent complexity of the Hamiltonian for larger system sizes [which depends on $\mathcal{O}(N^4)$ random couplings], ground state properties of the SYK model can be derived analytically in the limit of infinite system size. In this limit, the model exhibits the so-called *self-averaging* property [237], which implies that the physical characteristics of the ground state of Eq. (6.1) become independent of the particular realization of the couplings $J_{ij;kl}$. In this sense, the effect of disorder on the ground state properties vanishes with increasing system size. However, while several variational ansätze for the SYK model have been investigated recently [238, 239], it is not yet clear whether the model's large-N properties can be exploited to efficiently obtain ground state properties at finite sizes.

Motivated by these questions and the known capabilities of RBMs and more general deep networks to capture some forms of volume-law entanglement, we investigate whether a generic FFNN ansatz can be used to find a representation of particular SYK ground state realizations and how the required network size for such representations scales with the system size. A detailed discussion of our approach and results is presented in Publication [P2] included on the following pages.

Publication P2

[P2] Giacomo Passetti, **D.H.,** Pit Neitemeier, Lukas Grunwald, Michael A. Sentef, Dante M. Kennes. Can neural quantum states learn volume-law ground states?
Preprint (2022). DOI 10.48550/arXiv.2212.02204.

Summary

We investigate the practical applicability of an NQS ansatz to the paradigmatic SYK model, which is analytically solvable in the infinite-size limit but challenging to simulate at finite size [237–239]. We choose an FFNN ansatz to connect to existing results on the representability of volume-law states [101, 102, 118]. To avoid possible sources of error from VMC energy optimization, we train the network by optimizing the exact overlap with ground state amplitudes obtained from ED calculations for systems with up to 18 sites. With this approach, we systematically study the accuracy of the optimized ground state energy and its scaling with the network size. We find that our FFNN ansatz is unable to exploit the physical structure of the SYK ground states and does not learn a compressed representation of the exponentially large target states. This is in stark contrast to more structured lattice models, where NQS are known to achieve accurate variational energies with polynomially scaling size [86, 240], which we confirm for our ansatz by comparing the SYK results to the scaling achieved when learning the ground state of a local, one-dimensional spin model. Our results show that efficient compression for structured lattice models and the ability to capture generic volume-law entangled states does not automatically transfer to the more challenging ground state problem in the SYK model and, thus, highlight the need for further investigation into the scaling properties of NQS and the specific physical features that enable quantum states to be efficiently captured by an NQS ansatz.

Publication status

This work has been published as a preprint on arXiv. It has been submitted to *Physical Review Letters* and is currently under review.

The supplementary material is not included in this document and can be obtained from the arXiv version referenced above.

Declaration of contributions

I have assisted with implementing the simulation code based on NetKet 3 [P4] and contributed significantly to the choice of NQS ansatz and optimization scheme as well as to the analysis and interpretation of results. The manuscript was written collaboratively by G.P. and myself, incorporating contributions from the other authors to specific parts of the text.

Can neural quantum states learn volume-law ground states?

Giacomo Passetti,[1] Damian Hofmann,[2] Pit Neitemeier,[1]
Lukas Grunwald,[1, 2] Michael A. Sentef,[3, 2] and Dante M. Kennes[1, 2]

[1] *Institut für Theorie der Statistischen Physik, RWTH Aachen University and JARA-Fundamentals of Future Information Technology, 52056 Aachen, Germany*
[2] *Max Planck Institute for the Structure and Dynamics of Matter, Center for Free-Electron Laser Science (CFEL), Luruper Chaussee 149, 22761 Hamburg, Germany*
[3] *H H Wills Physics Laboratory, University of Bristol, Bristol BS8 1TL, United Kingdom*

(Dated: December 16, 2022)

We study whether neural quantum states based on multi-layer feed-forward networks can find ground states which exhibit volume-law entanglement entropy. As a testbed, we employ the paradigmatic Sachdev-Ye-Kitaev model. We find that both shallow and deep feed-forward networks require an exponential number of parameters in order to represent the ground state of this model. This demonstrates that sufficiently complicated quantum states, although being physical solutions to relevant models and not pathological cases, can still be difficult to learn to the point of intractability at larger system sizes. This highlights the importance of further investigations into the physical properties of quantum states amenable to an efficient neural representation.

Introduction.— The exponential complexity of representing general quantum many-body states is a key challenge in computational quantum physics. To simulate systems beyond small sizes tractable by exact diagonalization methods, it is necessary to find an efficient representation of quantum states of interest. This is made possible by the fact that physically relevant states usually possess a high degree of structure, compared to an arbitrary Hilbert space vector. As a prominent example, ground states of local, gapped Hamiltonians exhibit an area law of the entanglement entropy, i.e., an entanglement entropy that scales like the boundary of the subregion instead of its volume. For systems with a low dimensionality, typically 1D, the area law allows for an efficient representation of the wave function as a matrix product state, which can be simulated by algorithms such as the density matrix renormalization group (DMRG) [1–5].

However, many quantum states of physical interest display a volume law scaling of the entanglement entropy [6], for which generally applicable efficient representations are not known to this date. One class of variational approximations that has been studied to overcome this challenge are neural quantum states (NQS) [7], which are based on an artificial-neural-network representation of the wave function's probability amplitudes [8–10] and have shown promising results for the study of discrete lattice models even beyond one dimension [11–19]. Notably, it has been shown that a shallow NQS ansatz is able to efficiently represent quantum states featuring volume-law entanglement [20, 21], suggesting that this method could complement tensor network techniques for the purpose of uncovering the physics of highly entangled states. Nevertheless, while for matrix product states and more general tensor-network-based approaches it is known how the entanglement scaling limits the representation capabilities of the ansatz [3], there is so far no analogous physical property that directly relates to the ability of an NQS to learn a given quantum state. Universal approximation theorems, which have been proven for several broad classes of neural networks, guarantee that, in the limit of infinite network size, a neural network ansatz can theoretically represent any continuous function to arbitrary precision [22–25]. Still, these results do not provide bounds on the scaling of the required number of parameters with the system size. For practical applications of NQS, it is thus a central question to determine which classes of quantum many-body states can be efficiently represented that are impossible to tackle with other established variational ansätze.

In this Letter, we investigate the capabilities of NQS based on shallow and deep feed-forward neural networks (FFNNs) to represent ground states of the Sachdev-Ye-Kitaev (SYK) model [26–28], which is a paradigmatic model for quantum chaos and non-Fermi liquid behavior [29] and which features a volume-law entanglement in the ground state [30]. We present a systematic study of the representation accuracy achieved by the FFNN in dependence of the network hyperparameters. We find an exponential dependence on the system size for the number of network parameters required to learn the SYK ground state. This demonstrates limitations of fully general NQS to learn complicated quantum ground states of physical interest.

Model.— The SYK model describes strongly correlated fermions on L sites and is defined by the Hamiltonian [26–28]

$$\hat{H}_{\text{SYK}}(J) = \frac{1}{(2L)^{3/2}} \sum_{ijkl} J_{ij;kl}\, \hat{c}_i^\dagger \hat{c}_j^\dagger \hat{c}_k \hat{c}_l, \tag{1}$$

where $\hat{c}_i^{(\dagger)}$, $i \in \{1, \ldots L\}$, are fermionic ladder operators. The vertices $J_{ij;kl}$ have the symmetry $J^*_{ij;kl} = J_{lk;ji}$ and $J_{ij;kl} = -J_{ji;kl}$ and are random, uncorrelated, all-to-all couplings that are drawn from a Gaussian unitary ensemble (GUE) [31] with mean $\mathbb{E}[J_{ij;kl}] = 0$ and variance $\mathbb{E}\left[|J_{ij;kl}|^2\right] = 1$ [29]. Consequently, quantities of physical interest are expectation values over the ensemble of couplings J, which is evaluated after the quantum-

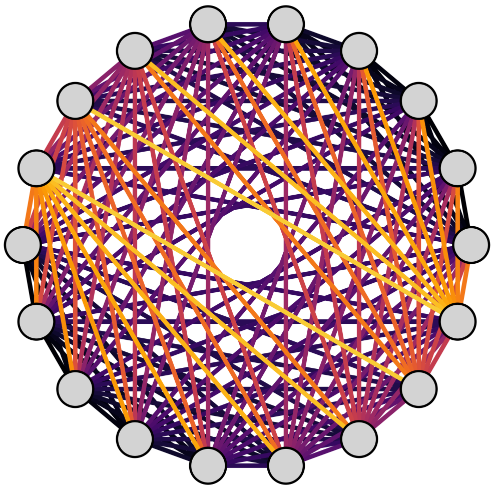


FIG. 1. Cartoon representation of the SYK model. Gray circles represent lattice sites and every different colour shown has two corresponding lines in total connecting four sites. Each color represents one element of the coupling matrix $J_{ij;kl}$ of the SYK model defined by Eq. (1).

expectation value. The ground state of the SYK model describes a strongly correlated non-Fermi liquid without quasi-particle excitations [29], that exhibits volume-law entanglement entropy [32, 33]. In the thermodynamic limit the model becomes self-averaging and exactly solvable, but despite this exact solvability, the ground state is not a Gaussian state, i.e. not a product of single particle wave-functions [34]. At finite sizes, particularly studied in the context of quantum chaos [35, 36] and experimental realizations [37], no exact solutions are known. Different variational ansätze to represent the ground state have been proposed recently [34, 38]. Here the model can be analyzed by employing approximations, or numerically, by drawing a set of couplings $\{J^{(n)}\}_{n=1}^{N}$ from the GUE, constructing the corresponding Hamiltonians $\hat{H}_{\text{SYK}}(J^{(n)})$, and solving for the ground states $|\Psi_{\text{GS}}(J^{(n)})\rangle$. Finally, the properties of interest, such as expectation values, are averaged over this ground state ensemble. Because of the self-averaging property of the SYK model, it suffices to evaluate expectation values for a single realization of J in the thermodynamic limit [29].

Network architecture.— We use a fully-connected FFNN [Figs. 2(a), 3(a)]

$$F(x) = f^{(\mu)} \circ \cdots \circ f^{(1)}(x),$$
$$f^{(l)}(y) = \phi(W^{(l)}y + b^{(l)}) \tag{2}$$

which is a composition of μ layers $f^{(l)}$, each applying an affine transformation and a scaled exponential linear unit (SELU) activation function ϕ [39] as pointwise non-linearity. Each layer has αL neurons, where α is the fixed hidden unit density. The output of the final layer is reduced to a (scalar) log-probability amplitude with respect to the computational basis $\{|x\rangle\}$ by an exponential sum,

$$\log \langle x|\psi_\theta\rangle = \log \sum_{i=1}^{\alpha L} \exp[F_i(x)]. \tag{3}$$

Here, θ denotes the vector of all variational parameters, which contains all entries of the weight matrices $W^{(l)}$ and bias vectors $b^{(l)}$. The variational parameters and therefore network outputs are complex numbers, with the activation function being applied separately to real and imaginary parts. The total number of network parameters scales as $N_{\text{par}} = \mathcal{O}(\mu\,\alpha^2 L^2)$. We choose the occupation number basis (as has been done in previous NQS studies of fermionic molecular Hamiltonians [40–42]) at half filling, which fixes the fermion number to $L/2$. Therefore, the input to the neural network (2) is a vector of occupation numbers $x \in \{0,1\}^L$ such that $\sum_i x_i = L/2$.

We have verified our results for several variations of this network architecture. In particular, we have evaluated using tanh as nonlinear activation function as well as the addition of skip connections, which can be used to counteract the increased training complexity of networks beyond a certain depth [43, 44]. These variations did not achieve better results compared to those presented in the main text. Details can be found in Section III of the supplemental material (SM) [45].

Optimization.— The ground state of the network is obtained by numerically minimizing the overlap difference

$$\delta O(\theta, J) = 1 - \left| \frac{\langle\psi_\theta|\psi_{\text{GS}}(J)\rangle}{\langle\psi_\theta|\psi_\theta\rangle} \right| \tag{4}$$

between the variational state $|\psi_\theta\rangle$ and the ground state $|\psi_{\text{GS}}(J)\rangle$ with respect to the variational parameters θ using Adam [46]. We work with system sizes up to $L = 18$ sites, which are accessible via exact diagonalization (ED) and thus enable training using a supervised learning (SL) protocol targeting the overlap with the ED ground state $|\psi_{\text{GS}}(J)\rangle$ [47]. The system size allows us to evaluate the loss function (4) by summation over the full Hilbert space (preventing any potential errors arising from Monte Carlo sampling) and to assess the quality of our results using the relative energy error

$$\delta E(\theta; J) = \frac{E(\theta; J) - E_{\text{GS}}(J)}{E_{\text{GS}}(J)} \tag{5}$$

compared to the target ground state energy $E_{\text{GS}}(J) = \langle\psi_{\text{GS}}(J)|\hat{H}_{\text{SYK}}(J)|\psi_{\text{GS}}(J)\rangle$. Details on the optimization scheme are reported in Section II of the SM [45].

Results.— To start, we discuss the minimum energy error $\delta E_{\min} = \min|_{t\in[0,t_{\max}]}\delta E(\theta, J)$ reached within a maximum number of iterations $t_{\max}$ of the optimization protocol. Figure 2(b) shows the dependence of $\delta E_{\min}$ on the network width α for a network with a fixed number of $\mu = 2$ layers, while Fig. 3(b) shows the results as a function of network depth μ for deep networks with constant width $\alpha = 4$. We select $\delta E_{\text{threshold}} = 10^{-3}$ as

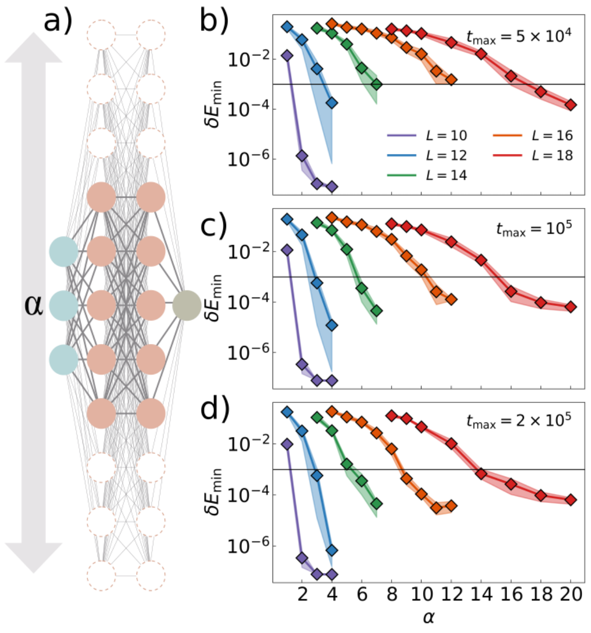


FIG. 2. (a) Shallow fully-connected feed-forward neural network, α denotes the hidden unit density of each layer and thus parametrizes the width of the network. (b), (c), (d) Relative ground state energy error δE as function of the network width α for several system sizes and random initializations after (b) 5×10^4, (c) 10^5, and (d) 2×10^5 simulation steps, respectively. The color of each set of data points corresponds to the average over four independent realizations of the network initial weights, for the system size L as indicated in the legend. The coloured areas give the maximum and minimum values of δE for the four independent runs. Black bars indicate $\delta E_{\text{threshold}} = 10^{-3}$.

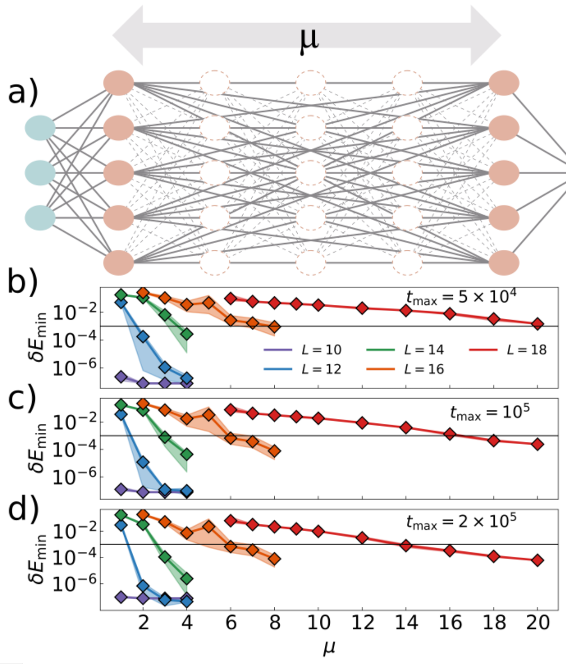


FIG. 3. (a) Deep fully-connected feed forward neural network, μ denotes the number of layers and thus the network depth. (b), (c), (d) Relative energy error δE as function of the network number of layers μ for several system sizes and random initializations after (b) 5×10^4, (c) 10^5, and (d) 2×10^5 simulation steps, respectively. The color of each set of data points corresponds to the average over four independent realizations of the network's initial weights, for the system size L as indicated in the legend. The coloured areas give the maximum and minimum values of δE for the four independent runs. Black bars indicate $\delta E_{\text{threshold}} = 10^{-3}$.

a threshold error to assess successful convergence to the desired ground state. With this threshold, one can see in Figs. 2(b) and 3(b) that at any fixed number of training iterations t_{max} there is a systematic improvement of the accuracy with respect to increasing both α and μ, as one would expect given the increased representation capabilities of the network at larger sizes.

Next, we determine the minimum number of variational parameters at which the network is able to learn the ground state with the desired energy of $\delta E_{\text{threshold}}$. Especially for the smallest system sizes, there is a clear transition between regimes where the network is able or unable to learn the state (in particular as a function of α in the shallow network). For larger system sizes, it is somewhat more difficult to assess convergence. While both very small and very large networks converge to energies above or below the desired threshold within a reasonable optimization time, there is an intermediate regime where the energy gets close to the threshold but only converges at very long time scales. In order to systematically identify a value of α or μ at that boundary, we have developed a criterion used to truncate optimization runs after a reasonable optimization time when those runs are predicted to ultimately converge to a $\delta E(\theta, J)$ higher than $\delta E_{\text{threshold}}$. See Section II B of the SM [45] for details. In Fig. 4 we show the number of network parameters at the critical α_{min} or μ_{min} at which the network is able to reach the target energy accuracy threshold. This allows for a comparison of network expressiveness for both varying width and depth on equal footing. We find that for both the shallow and deep network, an exponentially growing number of parameters is needed to achieve the target energy error. A comparison with the Hilbert space dimension reveals that the network only reaches this threshold once the number of variational parameters exceeds the number of probability amplitudes contained in the respective state vector. Hence, we find that our deep feed-forward NQS ansatz as trained here does not learn a more efficient representation of the SYK ground state than the full state vector representation. It is conceivable, in particular given the fully-connected nature of our ansatz, that there is some redundancy in the learned variational parameters, which could be used to achieve a degree of compression after training. In order to investigate this possibility, we have performed a low-rank approximation based on singular value decom-

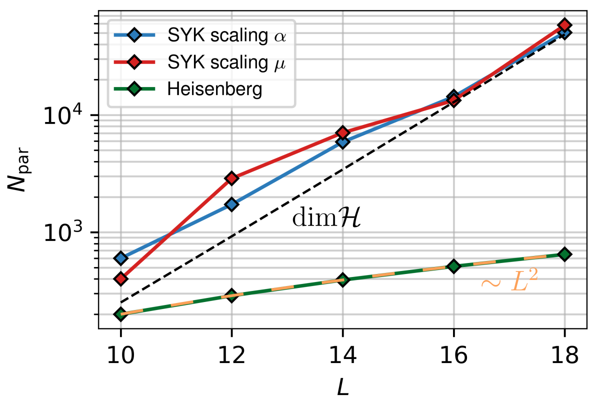


FIG. 4. Minimum number of parameters $N_{\rm par}$ required for the FFNN to learn the ground state of the SYK model as function of the system size L. Results are shown for the scaling with network width in a shallow ($\mu = 2$) network (blue lines) and for the scaling with network depth for fixed $\alpha = 4$ (red line). In both cases, an exponential scaling in the system size is observed, which matches the scaling of the full Hilbert space dimension $\dim\mathcal{H}$ (dashed line). The $N_{\rm par}$ scaling for the ground state of the Heisenberg model (blue) and the associated quadratic polynomial law are reported for comparison.

position of the weight matrices [48], the details of which are reported in Section V of the SM [45]. This analysis, however, has not revealed such an redundancy.

Our scaling results cannot be interpreted as an immediate consequence of the entanglement scaling of the SYK model, as NQS are known to be able to efficiently represent some volume-law quantum states [20], while they seem to fail for others (as shown here). While a particular realization of the SYK Hamiltonian is of significantly higher complexity than a low-dimensional local lattice Hamiltonian (both because of its fully connected structure and the $\propto L^4$ randomly drawn interaction matrix elements), its ground state still exhibits more structure than a random Hilbert space vector. Since it is well known that deep (and, in fact, already two-layer) networks are able to memorize even completely random data once the number of network parameters exceeds the number of data points [49], these results provide evidence that our FFNN ansatz does not learn to utilize any of this structure but only manages to learn it as unstructured random data. This is in stark contrast to more structured lattice Hamiltonians, where it is clear from previous works that neural quantum states can approximate ground state energies with sub-exponential scaling and thus do manage to make use of structure present in the quantum ground state [50, 51], although exponential scaling results as a function of real time have been previously found for time-evolved states in a one-dimensional lattice spin model [52]. We have found comparable sub-exponential behavior when evaluating our training procedure on the ground state of the Heisenberg spin model $\hat{H}_{\rm Heisb} = \sum_{i=1}^{N}\sum_{q=1}^{3}\hat{\sigma}_i^{(q)}\hat{\sigma}_{i+1}^{(q)}$ on a one-dimensional chain with periodic boundary conditions diagonalized in the same zero-magnetization subspace used for the SYK computations. The scaling of the required number of parameters to reach $\delta E_{\rm threshold}$ in this model is also reported in Fig. 4. In this case, a relatively small and fixed $\alpha = 1$ and $\mu = 2$ independent of the system size are sufficient to reach this threshold, implying a polynomial scaling of the required number of parameters $N_{\rm par} = \mathcal{O}(L^2)$. This corresponds to an effective compression of the information contained in the exact state vector and allows to study sizes beyond those tractable by full state simulation [7, 50]. However, the same approach fails to be useful in the more complex SYK model case.

Discussion.— We have tackled the prototypical SYK model using an NQS variational ansatz, presenting a systematic study of the ability of deep FFNNs to learn the volume-law entangled ground states of this model. Focusing on the scaling of the required number of parameters to describe the ground state to a desired and fixed accuracy we find that the size of the FFNN ansatz needs to grow exponentially in the system size. With this we show explicitly that the neural network ansatz is unable to efficiently represent SYK ground states in larger systems in spite of general results raising such hopes. We have performed this analysis using a variety of training techniques (as detailed in the SM [45]), showing that the observed scaling is robust to such implementation choices. While the proven capability of random RBMs to represent volume-law quantum states [20, 21] indicates that NQS methods have the potential to tackle problems out of the reach of established tensor-network based methods, our results demonstrate that the entanglement entropy is not the property that determines whether or not a physical quantum state can be efficiently represented by an NQS. It remains an intriguing open question which other properties of a physical quantum state determine the efficient applicability of NQS-based methods. NQS ansätze more specifically tailored to fermionic systems could potentially achieve better scaling [42, 53]. Studies in this direction would help elucidate to what extent the non-local parity structure inherent to fermionic models [54] affects the learnability of the SYK ground state. Separating this influence from other sources of complexity, such as the lack of spatial structure and the disorder induced by random couplings, and thereby exploring the intermediate region between states that can be learned with compression (such as in the Heisenberg and similar spin models) and states that cannot (such as the SYK results presented here) can provide an improved understanding of the complexity of physical quantum states.

ACKNOWLEDGMENTS

We acknowledge helpful discussions with Giuseppe Carleo, Sebastian Goldt, Javed Lindner, Claudia Merger, Alexandre René, and Attila Szabó. NQS calculations have been performed using NetKet 3 [55, 56] with JAX

[57]. Computations were performed on the HPC system Ada at the Max Planck Computing and Data Facility (MPCDF). The authors also gratefully acknowledge computing time granted by the JARA Vergabegremium and provided on the JARA partition part of the supercomputer JURECA at Forschungszentrum Jülich [58] under the project ID enhancerg. We acknowledge support by the Max Planck-New York City Center for Nonequilibrium Quantum Phenomena.

[1] F. Verstraete and J. I. Cirac, Physical Review B **73** (2006), 10.1103/physrevb.73.094423.
[2] F. Verstraete, V. Murg, and J. Cirac, Advances in Physics **57**, 143 (2008).
[3] J. Eisert, M. Cramer, and M. B. Plenio, Reviews of Modern Physics **82**, 277 (2010).
[4] U. Schollwöck, Annals of Physics **326**, 96 (2011).
[5] J. I. Cirac, D. Pérez-García, N. Schuch, and F. Verstraete, Reviews of Modern Physics **93** (2021), 10.1103/revmodphys.93.045003.
[6] E. Bianchi, L. Hackl, M. Kieburg, M. Rigol, and L. Vidmar, PRX Quantum **3**, 030201 (2022).
[7] G. Carleo and M. Troyer, Science **355**, 602 (2017).
[8] J. Schmidhuber, Neural Networks **61**, 85 (2015).
[9] Y. LeCun, Y. Bengio, and G. Hinton, Nature **521**, 436 (2015).
[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, 2016) `http://www.deeplearningbook.org`.
[11] I. Glasser, N. Pancotti, M. August, I. D. Rodriguez, and J. I. Cirac, Physical Review X **8** (2018), 10.1103/physrevx.8.011006.
[12] S. R. Clark, Journal of Physics A: Mathematical and Theoretical **51**, 135301 (2018).
[13] R. Kaubruegger, L. Pastori, and J. C. Budich, Physical Review B **97** (2018), 10.1103/physrevb.97.195136.
[14] K. Choo, T. Neupert, and G. Carleo, Physical Review B **100** (2019), 10.1103/physrevb.100.125124.
[15] G. Fabiani and J. H. Mentink, SciPost Phys. **7**, 004 (2019).
[16] M. Schmitt and M. Heyl, Physical Review Letters **125** (2020), 10.1103/physrevlett.125.100503.
[17] G. Fabiani, M. Bouman, and J. Mentink, Physical Review Letters **127** (2021), 10.1103/physrevlett.127.097202.
[18] N. Astrakhantsev, T. Westerhout, A. Tiwari, K. Choo, A. Chen, M. H. Fischer, G. Carleo, and T. Neupert, Physical Review X **11** (2021), 10.1103/physrevx.11.041021.
[19] C. Roth, A. Szabó, and A. MacDonald, "High-accuracy variational monte carlo for frustrated magnets with deep neural networks," (2022).
[20] D.-L. Deng, X. Li, and S. Das Sarma, Phys. Rev. X **7**, 021021 (2017).
[21] X.-Q. Sun, T. Nebabu, X. Han, M. O. Flynn, and X.-L. Qi, Phys. Rev. B **106**, 115138 (2022).
[22] G. Cybenko, Mathematics of Control, Signals, and Systems **2**, 303 (1989).
[23] K. Hornik, Neural Networks **4**, 251 (1991).
[24] A. Pinkus, Acta Numerica **8**, 143 (1999).
[25] P. Kidger and T. Lyons, in *Proceedings of Thirty Third Conference on Learning Theory*, Proceedings of Machine Learning Research, Vol. 125, edited by J. Abernethy and S. Agarwal (PMLR, 2020) pp. 2306–2327.
[26] S. Sachdev and J. Ye, Phys. Rev. Lett. **70**, 3339 (1993).
[27] A. Kitaev, "A simple model of quantum holography," (2015).
[28] J. Maldacena and D. Stanford, Phys. Rev. D **94**, 106002 (2016).
[29] D. Chowdhury, A. Georges, O. Parcollet, and S. Sachdev, Reviews of Modern Physics **94**, 035004 (2022).
[30] C. Liu, X. Chen, and L. Balents, Phys. Rev. B **97**, 245126 (2018).
[31] G. Akemann, J. Baik, and P. D. Francesco, eds., *The Oxford Handbook of Random Matrix Theory* (Oxford University Press, 2015).
[32] W. Fu and S. Sachdev, Phys. Rev. B **94**, 035135 (2016).
[33] P. Zhang, (2022).
[34] A. Haldar, O. Tavakol, and T. Scaffidi, Phys. Rev. Research **3**, 023020 (2021).
[35] A. Altland and D. Bagrets, Nuclear Physics B **930**, 45 (2018).
[36] A. M. García-García and J. J. M. Verbaarschot, Phys. Rev. D **94**, 126010 (2016).
[37] M. Brzezinska, Y. Guan, O. V. Yazyev, S. Sachdev, and A. Kruchkov, (2022), arXiv:2208.01032.
[38] J. Kim, J. Kim, and D. Rosa, Phys. Rev. Research **3**, 023203 (2021).
[39] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," (2017), arXiv:1706.02515.
[40] K. Choo, A. Mezzacapo, and G. Carleo, Nature Communications **11** (2020), 10.1038/s41467-020-15724-9.
[41] P.-J. Yang, M. Sugiyama, K. Tsuda, and T. Yanai, Journal of Chemical Theory and Computation **16**, 3513 (2020).
[42] J. Hermann, J. Spencer, K. Choo, A. Mezzacapo, W. M. C. Foulkes, D. Pfau, G. Carleo, and F. Noé, "Ab-initio quantum chemistry with neural-network wavefunctions," (2022).
[43] K. He, X. Zhang, S. Ren, and J. Sun, in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016) pp. 770–778.
[44] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, edited by S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (2018) pp. 6391–6401.
[45] See Supplementary Material at *[link to be inserted]* for details on the network architecture, training protocols, and comparisons with different optimisation schemes.
[46] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," (2014), arXiv:1412.6980.
[47] B. Jónsson, B. Bauer, and G. Carleo, "Neural-network states for the classical simulation of quantum computing," (2018).

[48] J. Xue, J. Li, and Y. Gong, in *Interspeech 2013* (ISCA, 2013).

[49] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, Commun. ACM **64**, 107 (2021).

[50] D. Sehayek, A. Golubeva, M. S. Albergo, B. Kulchytskyy, G. Torlai, and R. G. Melko, Physical Review B **100** (2019), 10.1103/physrevb.100.195125.

[51] L. L. Viteritti, F. Ferrari, and F. Becca, SciPost Physics **12** (2022), 10.21468/scipostphys.12.5.166.

[52] S.-H. Lin and F. Pollmann, physica status solidi (b) **259**, 2100172 (2022).

[53] J. Robledo Moreno, G. Carleo, A. Georges, and J. Stokes, Proceedings of the National Academy of Sciences **119** (2022), 10.1073/pnas.2122059119.

[54] F. Verstraete and J. I. Cirac, Journal of Statistical Mechanics: Theory and Experiment **2005**, P09012 (2005).

[55] F. Vicentini, D. Hofmann, A. Szabó, D. Wu, C. Roth, C. Giuliani, G. Pescia, J. Nys, V. Vargas-Calderón, N. Astrakhantsev, and G. Carleo, SciPost Phys. Codebases , 7 (2022).

[56] G. Carleo, K. Choo, D. Hofmann, J. E. Smith, T. Westerhout, F. Alet, E. J. Davis, S. Efthymiou, I. Glasser, S.-H. Lin, M. Mauri, G. Mazzola, C. B. Mendl, E. van Nieuwenburg, O. O'Reilly, H. Théveniaut, G. Torlai, F. Vicentini, and A. Wietek, SoftwareX **10**, 100311 (2019).

[57] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, "JAX: composable transformations of Python+NumPy programs," (2018).

[58] P. Thörnig, Journal of large-scale research facilities JLSRF **7** (2021), 10.17815/jlsrf-7-182.

7 Simulation software for neural quantum state methods

The availability of free, open-source software for computational tasks plays a vital role in the computational sciences. Accessible code encourages the reproduction of research (which increases confidence in published results) and the exploration of ideas beyond the scope of the work for which an implementation was first developed. Furthermore, open software projects lower the barrier of entry for newcomers, help to establish collaborations, and provide a setting for code review and exchange.

The meaning of the term *open source* goes beyond the mere availability of source code [241]. Following the Open Source Initiative's definition [242], the main requirement of open source software is its availability under licensing terms that allow users free redistribution and the creation of derivative works. Thus, the code of open-source software cannot just be read and reviewed, but also shared, modified, and built upon[26]. This greatly simplifies the exchange and extension of code and stimulates the creation of an ecosystem around open-source solutions [243].

The need for open-source implementations of algorithms in machine learning research has been formulated by Sonnenburg et al. [107] in 2007, lamenting that, in their words, "the true potential of these methods is not used, since existing implementations are not openly shared, resulting in software with low usability, and weak interoperability." Since that time, the machine learning field has progressed tremendously in this regard. Nowadays, the majority of machine learning research is built on open-source software [108] and major frameworks such as TensorFlow [244, 245] and PyTorch [246] provide easy access to common network architectures, training algorithms, and technical features such as GPU computing.

As in machine learning, research in computational sciences also benefits from open-source implementations of central models and algorithms [247–249]. Examples of established open-source software used in computational quantum physics include the ALPS [250, 251] and TRIQS [252], which provide implementations of various algorithms for quantum many-body physics, ITensor [253] and TenPy [134] for MPS/TNS calculations, and mVMC [133] for many-variable VMC calculations using pair-product states. In the NQS community, several open-source frameworks have been published since the introduction of the ansatz in 2017, in particular NetKet [P3, P4], which is described in the remainder of this chapter, and jVMC [110]. More specialized frameworks, such as QuCumber [109] for QST applications, are also available.

In the following sections, we will briefly describe key design aspects for the major releases of NetKet 2 and NetKet 3, respectively.

[26]The redistribution of modified works is typically subject to conditions such as requiring attribution and, in case of less permissive open source licenses, requiring modifications or the whole derived work to retain the open-source license of incorporated code [241].

7.1 NetKet 2

The NetKet project was founded by Giuseppe Carleo, following the publication of the seminal work Ref. [70], to provide the main building blocks and algorithms for studying quantum many-body systems using NQS methods in a collaboratively developed open-source environment [254]. The initial public version, NetKet 1.0, was released in 2018. This was followed by a period of intense development in a growing collaboration of researchers from different institutions, leading to the release of NetKet 2 in 2019.

NetKet 2 provides high-level drivers for VMC ground state search, QST, and supervised learning of amplitudes (Section 3.2.2). These can be customized by using various provided MCMC sampling rules, NQS ansätze, and optimization algorithms. Furthermore, NetKet 2 provides an interface for specifying the (discrete) Hilbert space, graph or lattice structure, and Hamiltonian of the system. Since parallelization is crucial for efficient VMC calculations, NetKet 2 supports distributing calculations over multiple cores and machines in an HPC environment using the message passing interface (MPI) standard.

The framework uses a combination of code in the Python and C++ programming languages (Fig. 7.1). The core functionality is written in C++ and then exposed to end users in Python using binding code based on the `pybind11` library [255]. Such a combination of languages is commonly employed in scientific software frameworks [252, 256] and makes it possible to combine both languages so that their respective strengths complement each other: The primary numerical routines and simulation loops can be implemented in C++, which is compiled to efficient machine code. At the same time, setup and data analysis can be performed in Python, which provides more flexibility for these tasks and allows NetKet to be more comfortably used interactively, e.g., in notebook interfaces [257, 258]. This interactivity aids in quick experimentation and the organization of research code. The significant performance drawbacks of pure Python code can be accepted in this case since the heavy computations are performed within the C++ core. This design, however, generally requires modifying the internal C++ code to add new features such as NQS architectures or optimization schemes.

Further details on NetKet 2 are provided in the included software paper [P3].

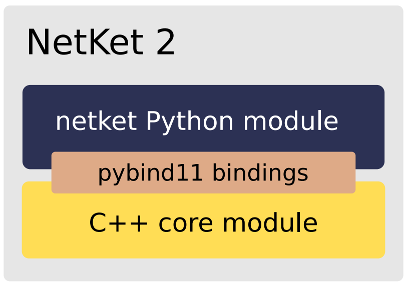


Figure 7.1 | NetKet 2 is built on a C++ core library containing implementations of the main numerical routines. This core functionality is made available as an internal Python module using `pybind11` bindings. The high-level interface of the NetKet 2 Python module is built around these bindings.

7.2 NetKet 3

NetKet 3, released in 2021, represents a significant update to the framework. The most fundamental change is the full replacement of the C++ core underlying NetKet 2 by pure Python code, which has been made possible by the use of JAX, a Python-based framework for accelerated linear algebra [234, 235]. Using JAX, core computational routines can be written as Python functions. These functions are then *traced*, i.e., their execution is recorded as a graph of primitive operations, the *computational graph* [259] (see Fig. 7.2 for an example). This representation can be used to apply transformations to the recorded computations, particularly just-in-time (JIT) compilation and automatic differentiation [235].

JIT compilation refers to the translation of source code to efficient machine code at runtime. JAX relies on the accelerated linear algebra (XLA) compiler [260] for this task, which is developed as part of TensorFlow and allows JAX to target CPU and GPU platforms as compilation targets. These JIT compilation and optimization capabilities have made it possible to implement all core NetKet code in Python and thus remove the need for a C++ core without compromising computational performance. Distributed computations using MPI are still supported using `mpi4py` [261] with the help of the `mpi4jax` library for JAX compatibility [262].

Automatic differentiation refers to the computation of derivatives by accumulating additional gradient information during code execution; as such, it differs from numerical and symbolic differentiation [263]. JAX supports automatic differentiation based on the computational graph of compatible Python functions [235]. This is particularly useful for reducing the effort needed to implement custom NQS models: Whereas NetKet 2 relies on hand-written implementations of gradients, which need to be provided for every model, NetKet 3 makes it possible to write explicit code only for the forward pass $(\boldsymbol{s}, \boldsymbol{\theta}) \mapsto \ln \psi_{\boldsymbol{\theta}}(\boldsymbol{s})$ through the network. The log-derivatives of the ansatz can then be obtained using JAX. Alternatively, automatic differentiation can be performed on the computational graph associated with a higher-level operation, such as the computation of energy expectation values, to obtain relevant gradients without explicitly relying on approaches such as the log-derivative trick (4.20).

Additional features that were introduced in NetKet 3 (over several minor releases) include a JAX-based t-VMC implementation, an improved interface for handling lattices and their symmetry groups, and built-in network architectures such as GCNNs and NAQS. Further details on NetKet 3 are provided in the included software paper [P4].

a)

```
def linear(x, W, b):
    v = W @ x + b
    y = tanh(v)
    return y
```

b)

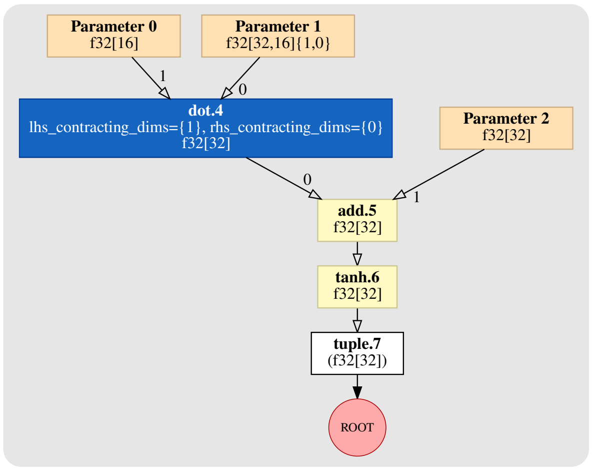


Figure 7.2 | a) Schematic example code snippet implementing a single linear layer (3.9) with $\tanh$ activation and **b)** the corresponding computational graph representation obtained using JAX, which shows the primitive linear algebra operations recorded while tracing the execution of the function `linear`. In this case, the parameters are $\boldsymbol{x} \in \mathbb{R}^{16}$, $\mathbf{W} \in \mathbb{R}^{32\times16}$, and $\boldsymbol{b} \in \mathbb{R}^{32}$, with real numbers represented by 32-bit floats.

Publication P3

[P3] Giuseppe Carleo, Kenny Choo, **D.H.,** James E. T. Smith, Tom Westerhout, Fabien Alet, Emily J. Davis, Stavros Efthymiou, Ivan Glasser, Sheng-Hsuan Lin, Marta Mauri, Guglielmo Mazzola, Christian B. Mendl, Evert van Nieuwenburg, Ossian O'Reilly, Hugo Théveniaut, Giacomo Torlai, Filippo Vicentini, Alexander Wietek.
NetKet: A machine learning toolkit for many-body quantum systems.
SoftwareX 10, 100311 (2019). DOI 10.1016/j.softx.2019.100311.

Summary

This software paper introduces NetKet 2. It provides an overview of the structure and high-level design decisions of the framework, describes the included algorithms for ground state search, quantum state tomography, and supervised learning, and provides code examples demonstrating basic usage.

Publication status

This work has been published in SoftwareX.

Declaration of contributions

For this software paper, I have written several parts of the initial manuscript (primarily in collaboration with G.C. and K.C.), particularly in Section 2, and created Fig. 1. During the preparation of this paper, I have been part of the NetKet project as a core contributor. This has involved contributing code to several parts of the framework, particularly related to time evolution, as well as contributions to design discussions, code reviews, and bug fixes concerning various parts of the codebase.

SoftwareX 10 (2019) 100311

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

NetKet: A machine learning toolkit for many-body quantum systems

Giuseppe Carleo [a,*], Kenny Choo [b], Damian Hofmann [c], James E.T. Smith [d], Tom Westerhout [e], Fabien Alet [f], Emily J. Davis [g], Stavros Efthymiou [h], Ivan Glasser [h], Sheng-Hsuan Lin [i], Marta Mauri [a,j], Guglielmo Mazzola [k], Christian B. Mendl [l], Evert van Nieuwenburg [m], Ossian O'Reilly [n], Hugo Théveniaut [f], Giacomo Torlai [a], Filippo Vicentini [o], Alexander Wietek [a]

[a] *Center for Computational Quantum Physics, Flatiron Institute, 162 5th Avenue, NY 10010, New York, USA*
[b] *Department of Physics, University of Zurich, Winterthurerstrasse 190, 8057 Zürich, Switzerland*
[c] *Max Planck Institute for the Structure and Dynamics of Matter, Luruper Chaussee 149, 22761 Hamburg, Germany*
[d] *Department of Chemistry, University of Colorado Boulder, Boulder, CO 80302, USA*
[e] *Institute for Molecules and Materials, Radboud University, NL-6525 AJ Nijmegen, The Netherlands*
[f] *Laboratoire de Physique Théorique, IRSAMC, Université de Toulouse, CNRS, UPS, 31062 Toulouse, France*
[g] *Department of Physics, Stanford University, Stanford, CA 94305, USA*
[h] *Max-Planck-Institut für Quantenoptik, Hans-Kopfermann-Straße 1, 85748 Garching bei München, Germany*
[i] *Department of Physics, T42, Technische Universität München, James-Franck-Straße 1, 85748 Garching bei München, Germany*
[j] *Dipartimento di Fisica, Università degli Studi di Milano, via Celoria 16, I-20133 Milano, Italy*
[k] *Theoretische Physik, ETH Zürich, 8093 Zürich, Switzerland*
[l] *Technische Universität Dresden, Institute of Scientific Computing, Zellescher Weg 12-14, 01069 Dresden, Germany*
[m] *Institute for Quantum Information and Matter, California Institute of Technology, Pasadena, CA 91125, USA*
[n] *Southern California Earthquake Center, University of Southern California, 3651 Trousdale Pkwy, Los Angeles, CA 90089, USA*
[o] *Université de Paris, Laboratoire Matériaux et Phénomènes Quantiques, CNRS, F-75013, Paris, France*

A R T I C L E   I N F O

Article history:
Received 28 March 2019
Received in revised form 9 August 2019
Accepted 12 August 2019

Keywords:
Neural-network quantum states
Variational Monte Carlo
Quantum state tomography
Machine learning
Supervised learning

A B S T R A C T

We introduce NetKet, a comprehensive open source framework for the study of many-body quantum systems using machine learning techniques. The framework is built around a general and flexible implementation of neural-network quantum states, which are used as a variational ansatz for quantum wavefunctions. NetKet provides algorithms for several key tasks in quantum many-body physics and quantum technology, namely quantum state tomography, supervised learning from wavefunction data, and ground state searches for a wide range of customizable lattice models. Our aim is to provide a common platform for open research and to stimulate the collaborative development of computational methods at the interface of machine learning and many-body physics.

Code metadata

Current code version	2.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX_2019_95
Code Ocean compute capsule	n/a
Legal Code License	Apache 2.0
Code versioning system used	git
Software code languages, tools, and services used	C++, Python, MPI
Compilation requirements, operating environments & dependencies	C++ compiler supporting C++11 (tested with GCC $\geq$ 5, Clang $\geq$ 4, and Xcode $\geq$ 9), MPI, Python 2.7 or 3.6
Developer documentation/manual	https://netket.org/docs
Support email for questions	netket@netket.org

* Corresponding author.
E-mail address: gcarleo@flatironinstitute.org (G. Carleo).

https://doi.org/10.1016/j.softx.2019.100311

1. Motivation and significance

Recent years have seen a tremendous activity around the development of physics-oriented numerical techniques based on machine learning (ML) tools [1]. In the context of many-body quantum physics, one of the main goals of these approaches is to tackle complex quantum problems using compact representations of many-body states based on artificial neural networks. These representations, dubbed neural-network quantum states (NQS) [2], can be used for several applications. In the supervised learning setting, they can be used, e.g., to learn existing quantum states for which a non-NQS representation is available [3]. In the unsupervised setting, they can be used to reconstruct complex quantum states from experimental measurements, a task known as quantum state tomography [4]. Finally, in the context of purely variational applications, NQS can be used to find approximate ground- and excited-state solutions of the Schrödinger equation [2,5–9], as well as to describe unitary [2,10,11] and dissipative [12–15] many-body dynamics. Despite the increasing methodological and theoretical interest in NQS and their applications, a set of comprehensive, easy-to-use tools for research applications is still lacking. This is particularly pressing as the complexity of NQS-related approaches and algorithms is expected to grow rapidly given these first successes, steepening the learning curve.

The goal of NetKet is to provide a set of primitives and flexible tools to ease the development of cutting-edge ML applications for quantum many-body physics. NetKet also wants to help bridge the gap between the latest and technically demanding developments in the field and those scholars and students who approach the subject for the first time. Pedagogical tutorials are provided to this aim. Serving as a common platform for future research, the NetKet project is meant to stimulate the open and easy-to-certify development of new methods and to provide a common set of tools to reproduce published results.

A central philosophy of the NetKet framework is to provide tools that are as simple as possible to use for the end user. Given the huge popularity of the Python programming language and of the many accompanying tools gravitating around the Python ecosystem, we have built NetKet as a full-fledged Python library. This simplicity of use however does not come at the expense of performance. With this efficiency requirement in mind, all critical routines and components of NetKet have been written in C++11.

2. Software description

We will first give a general overview of the structure of the code in Section 2.1 and then provide additional details on the functionality of NetKet in Section 2.2.

2.1. Software architecture

The core of NetKet is implemented in C++. For ease of use and in order to facilitate the integration with other frameworks, a Python interface is provided, which exposes all high-level functionality from the C++ core via `pybind11` [16] bindings. Use of the Python interface is recommended for users building on the library for research purposes, while the C++ code should be modified for extending the NetKet library itself.

NetKet is divided into several submodules. The modules `graph`, `hilbert`, and `operator` contain the classes necessary for specifying the structure of the many-body Hilbert space, the Hamiltonian, and other observables of a quantum system.

The core component of NetKet is the `machine` module, which provides different variational representations of the quantum wavefunction, particularly in the form of NQS. Encodings of mixed states, needed to describe dissipative quantum system, are implemented in the `machine.densitymatrix` submodule in the form of Neural Density Operators (NDO) [17]. The `variational`, `supervised`, and `unsupervised` modules contain driver classes for energy optimization, supervised learning, and quantum state tomography, respectively. These driver classes are supported by the `sampler` and `optimizer` modules, which provide classes for performing Variational Monte Carlo (VMC) sampling and optimization steps.

The `exact` module provides functions for exact diagonalization (ED) based on SciPy [18] and time propagation of the full quantum state, in order to allow for easy benchmarking and exploration of small systems within the NetKet framework. The NetKet operator classes implement the SciPy linear-operator interface and can also be converted to sparse and dense matrices, providing interoperability with Python code. In particular, the sparse and dense ED routines provided by the `exact` module are implemented as thin wrappers around SciPy functionality. The `dynamics` module provides basic ODE solvers for exact time propagation.

The utility modules `output`, `stats`, and `util` contain some additional functionality for output and statistics that is used internally in other parts of NetKet.

An overview of the most important modules and their dependencies is given in Fig. 1. A more detailed description of the module contents will be given in the next section.

NetKet uses the Eigen 3 library [19] for linear algebra routines. In the Python interface, Eigen datatypes are transparently converted to and from NumPy [20] arrays by `pybind11`. The NetKet driver classes provide methods to directly write the simulation output to JSON files, which is done with the help of the `nlohmann/json` library for C++ [21]. Parallelization is implemented based on the Message Passing Interface (MPI), allowing to substantially decrease running time. Specifically, the Monte Carlo sampling of expectation values implemented in the `variational.Vmc` class is parallelized, with each node drawing independent samples from the probability distribution which are averaged over all nodes.

2.2. Software functionalities

The core feature of NetKet is the variational representation of quantum states by artificial neural networks. Given a variational state, the task is to optimize its parameters with regard to a specified loss function, such as the total energy for ground state searches or the (negative) overlap with a given target state. In this section, we will discuss the models, types of variational wavefunctions, and learning schemes that are available in NetKet.

2.2.1. Model specification

NetKet currently supports lattice models with a finite Hilbert space of the form $\mathcal{H} = \mathcal{H}_{\text{local}}^{\otimes N}$ where N denotes the number of lattice sites and $\mathcal{H}_{\text{local}}$ denotes the local Hilbert space of each site. The system is defined on a graph with a set of N sites and a set of edges (also called bonds) between pairs of sites. This graph structure is used to help with the definition of operators on the lattice and to encode the spatial structure of the model, which is necessary, e.g., to work with convolutional neural networks (CNNs). NetKet provides the predefined `Hypercube` and `Lattice` graphs. Furthermore, `CustomGraph` supports arbitrary edge-colored graphs, where each edge is associated with an integer label called its color. This color can be used to describe different types of bonds.

General lattice spin models can be described straightforwardly in this manner. Bosonic lattice models can also be easily represented by truncating the local Hilbert space to only allow for

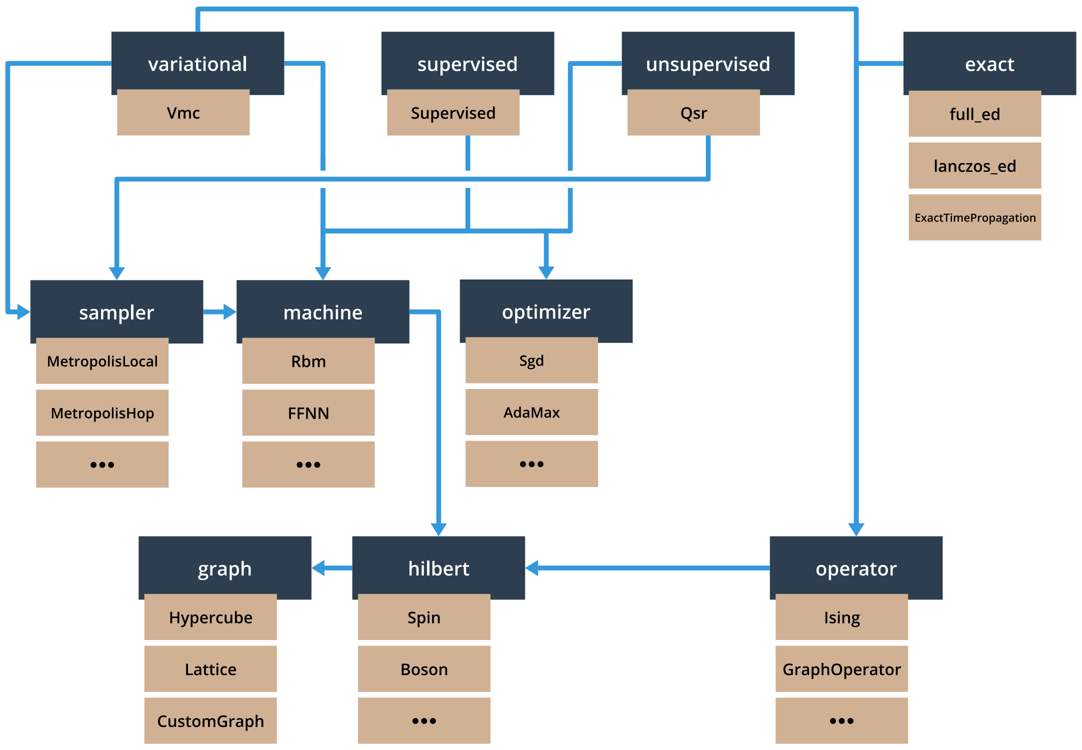


Fig. 1. The main submodules of the `netket` Python module and their dependencies from a user perspective (i.e., only dependencies in the public interface are shown). Below each submodule, examples of contained classes and functions are displayed. In a typical workflow, users will first define a quantum model, specify a variational representation of the wavefunction as well as the Monte Carlo sampling and optimization methods, and then run the simulation using one of the driver classes. A more detailed description of the software architecture and features is given in the main text.

occupations of up to N_{local} bosons per site [22]. NetKet currently provides pre-defined Hamiltonians for the transverse-field Ising, Heisenberg, and Bose–Hubbard models. Other observables and custom Hamiltonians can also be specified: a convenient option for common lattice models is to use the `GraphOperator` class, which allows to construct a Hamiltonian from a family of 2-local operators acting on each bond of a selected color and a family of 1-local operators acting on each site. It is also possible to specify general k-local operators (as well as their products and sums) using the `LocalOperator` class.

While fermionic Hamiltonians are not fully supported in the present version, they can be implemented using a custom Jordan–Wigner mapping and the `LocalOperator` class [23].

2.2.2. *Variational quantum states*

The purpose of variational states is to provide a compact and computationally efficient representation of quantum states. Since generally only a subset of the full many-body Hilbert space will be covered by a given variational ansatz, the aim is to use a parametrization that captures the relevant physical states for a given problem.

The variational wavefunctions supported by NetKet are provided as part of the `machine` module, which currently includes NQS but also Jastrow wavefunctions [24,25] and matrix-product states (MPS) [26–28].

Broadly, there are two main types of NQS available in NetKet: restricted Boltzmann machines (RBM) [29] and feed-forward neural networks (FFNN) [8,9,30,31]. Both types of networks are fully complex, i.e., with both complex-valued parameters and output.

The `machine` module contains the `RbmSpin` class for spin-$\frac{1}{2}$ systems as well as two other variants: the symmetric RBM (`RbmSpinSymm`) to capture lattice symmetries such as translation and inversion symmetries and the multi-valued RBM (`RbmMultiVal`) for systems with larger local Hilbert spaces (such as higher spins or bosonic systems).

FFNNs represent a broad and flexible class of networks and are implemented by the `FFNN` class. They consist of a sequence of layers available from the `layer` submodule, each layer performing either an affine transformation to the input vector or applying a non-linear activation function. There are currently two types of affine maps available:

- Dense fully-connected layers, which for an input $\boldsymbol{x} \in \mathbb{C}^n$ and output $\boldsymbol{y} \in \mathbb{C}^m$ have the form $\boldsymbol{y} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b}$ where $\mathbf{W} \in \mathbb{C}^{m\times n}$ and $\boldsymbol{b} \in \mathbb{C}^m$ are called the *weight matrix* and *bias vector*, respectively.
- Convolutional layers [30,32] for hypercubic lattices.

As activation functions, rectified linear units (`Relu`) [33], hyperbolic tangent (`Tanh`) [34], and the logarithm of the hyperbolic cosine (`Lncosh`) are provided. RBMs without visible bias can be represented as single-layer FFNNs with $\ln\cosh$ activation, allowing for a generalization of these machines to multiple layers [5].

The `machine` module also provides more traditional variational wavefunctions, namely MPS with periodic boundary conditions (`MPSPeriodic`) and long-range Jastrow (`Jastrow`) wavefunctions, which allows for comparison of NQS with results obtained using these approaches.

Finally, NetKet also includes representations of mixed states in the `machine.densitymatrix` submodule, which most notably includes the real-valued NDO ansatz (`NdmSpinPhase`). For compatibility with the rest of the package, the vectorized representation of density matrices can be accessed through the same interface as NQS.

Custom wavefunctions may be provided by implementing subclasses of the `AbstractMachine` class in C++ or in Python by deriving `netket.machine.CxxMachine`.

2.2.3. Supervised learning

In supervised learning, a target wavefunction is given and the task is to optimize a chosen ansatz to represent it. This functionality is contained within the `supervised` module. Given a variational state $|\Psi_{\mathrm{NN}}(\boldsymbol{\alpha})\rangle$ depending on the parameters $\boldsymbol{\alpha} \in \mathbb{C}^m$ and a target state $|\Psi_{\mathrm{tar}}\rangle$, the negative log overlap

$$\mathcal{L}(\boldsymbol{\alpha}) = -\log \frac{\langle\Psi_{\mathrm{tar}}|\Psi_{\mathrm{NN}}(\boldsymbol{\alpha})\rangle}{\langle\Psi_{\mathrm{tar}}|\Psi_{\mathrm{tar}}\rangle} \frac{\langle\Psi_{\mathrm{NN}}(\boldsymbol{\alpha})|\Psi_{\mathrm{tar}}\rangle}{\langle\Psi_{\mathrm{NN}}(\boldsymbol{\alpha})|\Psi_{\mathrm{NN}}(\boldsymbol{\alpha})\rangle} \tag{1}$$

is taken as the loss function to be minimized. The loss is computed in a Monte Carlo fashion by direct sampling of the target wavefunction. To minimize the loss, the gradient $\nabla_{\boldsymbol{\alpha}}\mathcal{L}$ of the loss function with respect to the parameters is calculated. This gradient is then used to update the parameters according to a specified gradient-based optimization scheme. For example, in stochastic gradient descent (SGD) the parameters are updated as

$$\boldsymbol{\alpha} \to \boldsymbol{\alpha} - \lambda \nabla_{\boldsymbol{\alpha}}\mathcal{L} \tag{2}$$

where λ is the learning rate. The different update rules supported by NetKet are contained in the `optimizer` module. Various types of optimizers are available, including SGD, AdaGrad [35], AdaMax and AdaDelta [36], AMSGrad [37], and RMSProp.

2.2.4. Unsupervised learning

NetKet also allows to carry out unsupervised learning of unknown probability distributions, which in this context corresponds to quantum state tomography [38]. Given an unknown quantum state, a neural network can be trained on projective measurement data to discover an approximate reconstruction of the state [4]. In NetKet, this functionality is contained within the `unsupervised.Qsr` class.

For some given target quantum state $|\Psi_{\mathrm{tar}}\rangle$, the training dataset $\mathcal{D}$ consists of a sequence of projective measurements $\boldsymbol{\sigma}^{\boldsymbol{b}}$ in different bases $\boldsymbol{b}$, with underlying probability distribution $P(\boldsymbol{\sigma}^{\boldsymbol{b}}) = |\Psi_{\mathrm{tar}}(\boldsymbol{\sigma}^{\boldsymbol{b}})|^2$. The quantum reconstruction of the target state translates into minimizing the statistical divergence between the distribution of the measurement outcomes and the distribution generated by the NQS. This corresponds, up to a constant dataset entropy contribution, to maximizing the log-likelihood of the network distribution over the measurement data

$$\mathcal{L} = \sum_{\boldsymbol{\sigma}^{\boldsymbol{b}} \in \mathcal{D}} \log \pi(\boldsymbol{\sigma}^{\boldsymbol{b}}) \,, \tag{3}$$

where π denotes the probability distribution

$$\pi(\boldsymbol{\sigma}) = \frac{|\Psi_{\mathrm{NN}}(\boldsymbol{\sigma})|^2}{\sum_{\sigma'} |\Psi_{\mathrm{NN}}(\boldsymbol{\sigma}')|^2}. \tag{4}$$

generated by the NQS wavefunction.

Note that, for every training sample where the measurement basis differs from the reference basis $|\boldsymbol{\sigma}\rangle$ of the NQS, a unitary transformation $\hat{U}$ should be applied to appropriately change the basis, $\Psi_{\mathrm{NN}}(\boldsymbol{\sigma}^{\boldsymbol{b}}) = \hat{U}_{\boldsymbol{b}}\Psi_{\mathrm{NN}}(\boldsymbol{\sigma})$.

The network parameters are updated according to the gradient of the log-likelihood $\mathcal{L}$. This can be computed analytically, and it requires expectation values over both the training data points and the network distribution $\pi(\boldsymbol{\sigma})$. While the first is trivial to compute, the latter should be approximated by a Monte Carlo average over configurations sampled from a Markov chain.

2.2.5. Variational Monte Carlo

Finally, NetKet supports ground state searches for a given many-body quantum Hamiltonian $\hat{H}$. In this context, the task is to optimize the parameters of a variational wavefunction Ψ in order to minimize the energy $\langle\hat{H}\rangle$. The `variational.Vmc` driver class contains the main logic to optimize a variational wavefunction given a Hamiltonian, a sampler, and an optimizer.

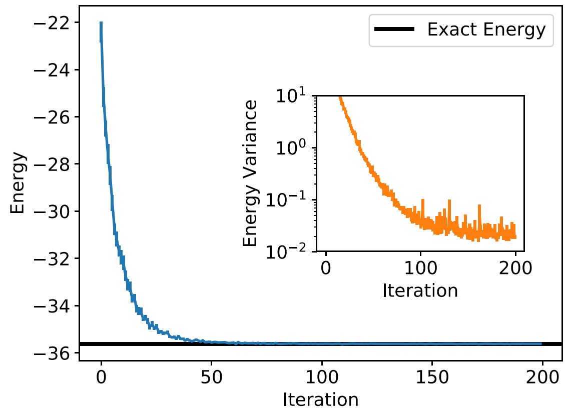


Fig. 2. Variational optimization of the restricted Boltzmann machine for the one-dimensional spin-$\frac{1}{2}$ Heisenberg model. The main plot shows the Monte Carlo energy estimate, which converges to the exact ground state energy up to a relative error $|(E - E_{\mathrm{exact}})/E_{\mathrm{exact}}|$ of 4.16×10^{-5} within the 200 iteration steps shown. The inset shows the Monte Carlo estimate of the energy variance, which becomes zero in an exact eigenstate of the Hamiltonian.

The energy of a wavefunction $\Psi(\boldsymbol{\sigma}) = \langle\boldsymbol{\sigma}|\Psi\rangle$ can be estimated as

$$\begin{aligned}\langle\hat{H}\rangle &= \frac{\sum_{\sigma,\sigma'} \Psi^*(\boldsymbol{\sigma})\,\langle\boldsymbol{\sigma}|\,\hat{H}\,|\boldsymbol{\sigma}'\rangle\,\Psi(\boldsymbol{\sigma}')}{\sum_{\sigma}|\Psi(\boldsymbol{\sigma})|^2} \\ &= \sum_{\sigma}\left(\sum_{\sigma'}\langle\boldsymbol{\sigma}|\,\hat{H}\,|\boldsymbol{\sigma}'\rangle\frac{\Psi(\boldsymbol{\sigma}')}{\Psi(\boldsymbol{\sigma})}\right)\frac{|\Psi(\boldsymbol{\sigma})|^2}{\sum_{\sigma'}|\Psi(\boldsymbol{\sigma}')|^2} \\ &\approx \left\langle\sum_{\sigma'}\langle\boldsymbol{\sigma}|\,\hat{H}\,|\boldsymbol{\sigma}'\rangle\frac{\Psi(\boldsymbol{\sigma}')}{\Psi(\boldsymbol{\sigma})}\right\rangle_{\sigma}\end{aligned} \tag{5}$$

where in the last line $\langle\cdot\rangle_{\sigma}$ denotes a stochastic expectation value taken over a sample of configurations $\{\boldsymbol{\sigma}\}$ drawn from the probability distribution corresponding to the variational wavefunction (4). This sampling is performed by classes from the `sampler` module, which generate Markov chains of configurations using the Metropolis algorithm [39] to ensure detailed balance. Parallel tempering [40] options are also available to improve sampling efficiency.

In order to optimize the parameters of a machine to minimize the energy, a gradient-based optimization scheme can be applied as discussed in the previous section. The energy gradient can be estimated at the same time as $\langle\hat{H}\rangle$ [2,25]. This requires computing the partial derivatives of the wavefunction with respect to the variational parameters, which can be obtained analytically for the RBM [2] or via backpropagation [30,31,34] for multi-layer FFNNs. In this case, the steepest descent update according to Eq. (2) is also a form of SGD, because the energy is estimated using a subset of the full data available from the variational wavefunction. Alternatively, often more stable convergence can be achieved by using the stochastic reconfiguration (SR) method [41,42], which approximates the imaginary time evolution of the system on the submanifold of variational states. The SR approach is closely related to the natural gradient descent method used in machine learning [43]. In the NetKet implementation, SR is performed using either an exact or an iterative linear solver, the latter being recommended when the number of variational parameters is large.

Information on the optimization run (sampler acceptance rates, energy, energy variance, expectation of additional observables, and the current variational parameters) for each iteration

```python
import netket as nk

# Define the graph: a 1D chain of 20 sites with periodic
# boundary conditions
g = nk.graph.Hypercube(length=20, n_dim=1, pbc=True)

# Define the Hilbert Space: spin half degree of freedom at each
# site of the graph, restricted to the zero magnetization sector
hi = nk.hilbert.Spin(s=0.5, total_sz=0.0, graph=g)

# Define the Hamiltonian: spin half Heisenberg model
ha = nk.operator.Heisenberg(hilbert=hi)

# Define the ansatz: Restricted Boltzmann machine
# with 20 hidden units
ma = nk.machine.RbmSpin(hilbert=hi, n_hidden=20)

# Initialise with machine parameters
ma.init_random_parameters(seed=1234, sigma=0.01)

# Define the Sampler: metropolis sampler with local
# exchange moves, i.e. nearest neighbour spin swaps
# which preserve the total magnetization
sa = nk.sampler.MetropolisExchange(graph=g, machine=ma)

# Define the optimiser: Stochastic gradient descent with
# learning rate 0.01.
opt = nk.optimizer.Sgd(learning_rate=0.01)

# Define the VMC object: Stochastic Reconfiguration "Sr" is used
gs = nk.variational.Vmc(hamiltonian=ha, sampler=sa,
                        optimizer=opt, n_samples=1000,
                        use_iterative=True, method='Sr')

# Run the VMC simulation for 1000 iterations
# and save the output into files with prefix "test"
# The machine parameters are stored in "test.wf"
# while the measurements are stored in "test.log"
gs.run(output_prefix='test', n_iter=1000)
```

Listing 1: Example script for finding the ground state of the one-dimensional spin-$\frac{1}{2}$ Heisenberg model using an RBM ansatz.

can be written to a log file in JSON format. Alternatively, they can be accessed directly inside the simulation loop in Python to allow for more flexible output.

3. Illustrative examples

NetKet is available as a Python package and can be obtained from the Python package index (PyPI) [44]. Assuming a properly configured Python environment, NetKet can be installed via the shell command

```
pip install netket
```

which will download, compile, and install the package. A working MPI environment is required to run NetKet. In case multiple MPI installations are present on the system and in order to avoid potential conflicts, we recommend to run the installation command as

```
CC=mpicc CXX=mpicxx pip install netket
```

with the desired MPI environment loaded in order to perform the build with the correct compiler. After a successful installation, the NetKet module can be imported in Python scripts.

Alternatively to installing NetKet locally, NetKet also uses the deployment of BinderHub from mybinder.org [45] to build and deploy a stable version of the software, which can be found at https://mybinder.org/v2/gh/netket/netket/v.2.0. This allows users to run the tutorials or other small jobs without installing NetKet.

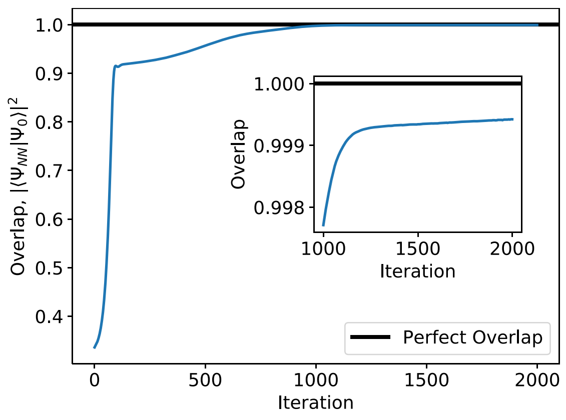


Fig. 3. Supervised learning of the ground state of the one-dimensional spin-$\frac{1}{2}$ transverse field Ising model with 10 sites from ED data, using an RBM with 20 hidden units. The blue line shows the overlap between the RBM wavefunction and the exact wavefunction for each iteration.

3.1. *One-dimensional Heisenberg model*

As a first example, we present a Python script for obtaining a variational RBM representation of the ground state of the spin-$\frac{1}{2}$ Heisenberg model on a one-dimensional chain with periodic boundary conditions. The code for this example is shown in Listing 1. Fig. 2 shows the evolution of the energy expectation value over the course of the optimization run. We see that for a small chain of 20 sites and an RBM with 20 hidden units, the

```python
import netket as nk
from numpy import log

# 1D Lattice
g = nk.graph.Hypercube(length=10, n_dim=1, pbc=True)

# Hilbert space of spins on the graph
hi = nk.hilbert.Spin(s=0.5, graph=g)

# Ising spin Hamiltonian
ha = nk.operator.Ising(h=1.0, hilbert=hi)

# Perform Exact Diagonalization to get lowest eigenvector
res = nk.exact.lanczos_ed(ha, first_n=1, compute_eigenvectors=True)

# Store eigenvector as a list of training samples and targets
# The samples would be the Hilbert space configurations and
# the targets should be wavefunction amplitudes.
hind = nk.hilbert.HilbertIndex(hi)
h_size = hind.n_states
targets = [[log(res.eigenvectors[0][i])] for i in range(h_size)]
samples = [hind.number_to_state(i) for i in range(h_size)]

# Machine: Restricted Boltzmann machine
# with 20 hidden units
ma = nk.machine.RbmSpin(hilbert=hi, n_hidden=20)
ma.init_random_parameters(seed=1234, sigma=0.01)

# Optimizer
op = nk.optimizer.AdaMax()

# Supervised Learning module
spvsd = nk.supervised.Supervised(machine=ma,
                                 optimizer=op,
                                 batch_size=400,
                                 samples=samples,
                                 targets=targets)

# Run the optimization for 2000 iterations
spvsd.run(n_iter=2000, output_prefix='test',
      loss_function="Overlap_phi")
```

Listing 2: Example script for supervised learning. A RBM ansatz is optimized to represent the ground state of the one-dimensional spin-$\frac{1}{2}$ transverse field Ising model obtained by ED for this example.

energy converges to a relative error of the order 10^{-5} within about 100 iteration steps.

3.2. *Supervised learning*

As a second example, we use the supervised learning module in NetKet to optimize an RBM to represent the ground state of the transverse field Ising model. The example script is shown in Listing 2. The exact ground state wavefunction is first obtained by exact diagonalization and then used for training the RBM state by minimizing the overlap loss (1). Fig. 3 shows the evolution of the overlap over the training iterations.

4. Impact

Given the flexibility of NetKet, we envision several potential applications of this library both in data-driven experimental research and in more theoretical, problem-driven research on interacting quantum many-body systems. For example, several important theoretical and practical questions concerning the expressibility of NQS, the learnability of experimental quantum states, and the efficiency at finding ground states of k-local Hamiltonians, can be directly addressed using the current functionality of the software.

Moreover, having an easy-to-extend set of tools to work with NQS-based applications can propel future research in the field, without researchers having to pay a significant cost of entry in terms of algorithm implementation and testing. Since its early release in April 2017, NetKet has already been used for research purposes by several groups worldwide [5,22,23,46–48]. We also hope that, building upon a common set of tools, practices like publishing accompanying codes to research papers, largely popular in the ML community, can become standard practice also for ML applications in quantum physics.

Finally, for a fast-growing community like ML for quantum science, it is also crucial to have pedagogical tools available that can be conveniently used by new generations of students and researchers. Benefiting from a growing set of tutorials and step-by-step explanations, NetKet can be comfortably used in schools and lectures.

5. Conclusions and future directions

We have introduced NetKet, a comprehensive open source framework for the study of many-body quantum systems using machine learning techniques. Central to this framework are variational parameterizations of many-body wavefunctions in the form of artificial neural networks. NetKet is a Python framework implemented in C++11, designed with efficiency as well as ease of use in mind. Several examples, tutorials, and notebooks are provided with our software in order to reduce the learning curve for newcomers.

The NetKet project is meant to continuously evolve in future releases, welcoming suggestions and contributions from its users. For example, future versions may provide a natural interface with general ML frameworks such as PyTorch [49] and Tensorflow [50]. On the algorithmic side, future goals include the extension of NetKet to incorporate unitary dynamics [11, 51], convenient Fermionic operators, as well as full support for density-matrix tomography [17].

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We acknowledge support from the Flatiron Institute of the Simons Foundation. J.E.T.S. gratefully acknowledges support from a fellowship through The Molecular Sciences Software Institute under NSF Grant ACI1547580. H.T. is supported by a grant from the Fondation CFM pour la Recherche. S.E. and I.G. are supported by an ERC Advanced Grant QENOCOBA under the EU Horizon2020 program (grant agreement 742102) and the German Research Foundation (DFG) under Germany's Excellence Strategy through Project No. EXC-2111 - 390814868 (MCQST).

This project makes use of other open source software, namely pybind11 [16], Eigen [19], nlohmann/json [21], NumPy [20], and SciPy [18].

Pre-release versions of NetKet 2.0 have used a Lanczos solver based on the IETL library from the ALPS project [52,53], which implements a variant of the Lanczos algorithm due to Cullum and Willoughby [54,55].

We further acknowledge discussions with, as well as bug reports, comments, and support from S. Arnold, A. Booth, A. Borin, J. Carrasquilla, C. Ciuti, S. Lederer, Y. Levine, T. Neupert, O. Parcollet, A. Rubio, M. A. Sentef, O. Sharir, M. Stoudenmire, and N. Wies.

References

[1] Carleo G, Cirac I, Cranmer K, Daudet L, Schuld M, Tishby N, Vogt-Maranto L, Zdeborová L. Machine learning and the physical sciences. 2019, arXiv:1903.10563.

[2] Carleo G, Troyer M. Solving the quantum many-body problem with artificial neural networks. Science 2017;355(6325):602–6. http://dx.doi.org/10.1126/science.aag2302, URL http://science.sciencemag.org/content/355/6325/602.

[3] Cai Z, Liu J. Approximating quantum many-body wave functions using artificial neural networks. Phys Rev B 2018;97(3):035116. http://dx.doi.org/10.1103/PhysRevB.97.035116, URL https://link.aps.org/doi/10.1103/PhysRevB.97.035116.

[4] Torlai G, Mazzola G, Carrasquilla J, Troyer M, Melko R, Carleo G. Neural-network quantum state tomography. Nat Phys 2018;14(5):447–50. http://dx.doi.org/10.1038/s41567-018-0048-5, URL https://doi.org/10.1038/s41567-018-0048-5.

[5] Choo K, Carleo G, Regnault N, Neupert T. Symmetries and many-body excitations with neural-network quantum states. Phys Rev Lett 2018;121:167204. http://dx.doi.org/10.1103/PhysRevLett.121.167204, URL https://link.aps.org/doi/10.1103/PhysRevLett.121.167204.

[6] Glasser I, Pancotti N, August M, Rodriguez ID, Cirac JI. Neural-network quantum states, string-bond states, and chiral topological states. Phys. Rev. X 2018;8:011006. http://dx.doi.org/10.1103/PhysRevX.8.011006, URL https://link.aps.org/doi/10.1103/PhysRevX.8.011006.

[7] Kaubruegger R, Pastori L, Budich JC. Chiral topological phases from artificial neural networks. Phys. Rev. B 2018;97:195136. http://dx.doi.org/10.1103/PhysRevB.97.195136, URL https://link.aps.org/doi/10.1103/PhysRevB.97.195136.

[8] Saito H. Solving the bose-hubbard model with machine learning. J Phys Soc Japan 2017;86(9):093001. http://dx.doi.org/10.7566/JPSJ.86.093001.

[9] Saito H, Kato M. Machine learning technique to find quantum many-body ground states of bosons on a lattice. J Phys Soc Japan 2018;87(1):014001. http://dx.doi.org/10.7566/JPSJ.87.014001.

[10] Czischek S, Gärttner M, Gasenzer T. Quenches near ising quantum criticality as a challenge for artificial neural networks. Phys. Rev. B 2018;98:024311. http://dx.doi.org/10.1103/PhysRevB.98.024311, URL https://link.aps.org/doi/10.1103/PhysRevB.98.024311.

[11] Jónsson B, Bauer B, Carleo G. Neural-network states for the classical simulation of quantum computing. 2018, arXiv:1808.05232, URL http://arxiv.org/abs/1808.05232.

[12] Hartmann MJ, Carleo G. Neural-network approach to dissipative quantum many-body dynamics. Phys Rev Lett 2019;122:250502. http://dx.doi.org/10.1103/PhysRevLett.122.250502, URL https://link.aps.org/doi/10.1103/PhysRevLett.122.250502.

[13] Yoshioka N, Hamazaki R. Constructing neural stationary states for open quantum many-body systems. Phys Rev B 2019;99(21):214306. http://dx.doi.org/10.1103/PhysRevB.99.214306, URL https://link.aps.org/doi/10.1103/PhysRevB.99.214306.

[14] Nagy A, Savona V. Variational quantum monte carlo method with a neural-network ansatz for open quantum systems. Phys Rev Lett 2019;122(25):250501. http://dx.doi.org/10.1103/PhysRevLett.122.250501, URL https://link.aps.org/doi/10.1103/PhysRevLett.122.250501.

[15] Vicentini F, Biella A, Regnault N, Ciuti C. Variational neural-network ansatz for steady states in open quantum systems. Phys Rev Lett 2019;122(25):250503. http://dx.doi.org/10.1103/PhysRevLett.122.250503, URL https://link.aps.org/doi/10.1103/PhysRevLett.122.250503.

[16] Jakob W, Rhinelander J, Moldovan D. Pybind11 – seamless operability between c++11 and python. 2019.

[17] Torlai G, Melko RG. Latent space purification via neural density operators. Phys Rev Lett 2018;120(24):240503. http://dx.doi.org/10.1103/PhysRevLett.120.240503, URL https://link.aps.org/doi/10.1103/PhysRevLett.120.240503.

[18] Jones E, Oliphant T, Peterson P, et al. SciPy: Open source scientific tools for Python. 2001, URL http://www.scipy.org/. [Online; Accessed 5 July 2019].

[19] Guennebaud G, Jacob B, et al. Eigen v3. 2010.

[20] Oliphant T. NumPy: A Guide to NumPy. USA: Trelgol Publishing; 2006, URL https://www.numpy.org/.

[21] Lohmann N, et al. JSON for modern C++. GitHub Repository 2019.

[22] McBrian K, Carleo G, Khatami E. Ground state phase diagram of the one-dimensional bose-Hubbard model from restricted Boltzmann machines. arXiv:1903.03076.

[23] Choo K, Mezzacapo A, Carleo G. Quantum Chemistry with Neural-Network Quantum States [in preparation].

[24] Manousakis E. The spin-$\frac{1}{2}$ heisenberg antiferromagnet on a square lattice and its application to the cuprous oxides. Rev Modern Phys 1991;63:1–62. http://dx.doi.org/10.1103/RevModPhys.63.1, URL https://link.aps.org/doi/10.1103/RevModPhys.63.1.

[25] Becca F, Sorella S. Quantum Monte Carlo Approaches for Correlated Systems. first ed. Cambridge University Press; 2017, http://dx.doi.org/10.1017/9781316417041.

[26] White SR. Density matrix formulation for quantum renormalization groups. Phys Rev Lett 1992;69(19):2863–6. http://dx.doi.org/10.1103/physrevlett.69.2863.

[27] Rommer S, Östlund S. Class of ansatz wave functions for one-dimensional spin systems and their relation to the density matrix renormalization group. Phys Rev B 1997;55(4):2164–81. http://dx.doi.org/10.1103/physrevb.55.2164.

[28] Schollwöck U. The density-matrix renormalization group in the age of matrix product states. Ann Physics 2011;326(1):96–192. http://dx.doi.org/10.1016/j.aop.2010.09.012.

[29] Hinton GE. Reducing the dimensionality of data with neural networks. Science 2006;313(5786):504–7. http://dx.doi.org/10.1126/science.1127647.

[30] LeCun Y, Bengio Y, Hinton GE. Deep learning. Nature 2015;521(7553):436–44. http://dx.doi.org/10.1038/nature14539.

[31] Goodfellow I, Bengio Y, Courville A. Deep Learning. MIT Press; 2016.

[32] Gu J, Wang Z, Kuen J, Ma L, Shahroudy A, Shuai B, Liu T, Wang X, Wang G, Cai J, Chen T. Recent advances in convolutional neural networks. Pattern Recognit 2018;77:354–77. http://dx.doi.org/10.1016/j.patcog.2017.10.013, URL http://www.sciencedirect.com/science/article/pii/S0031320317304120.

[33] Nair V, Hinton GE. Rectified linear units improve restricted Boltzmann machines. In: Proceedings of the 27th international conference on machine learning. 2010. p. 807–14. URL http://www.icml2010.org/papers/432.pdf.

[34] LeCun Y, Bottou L, Orr GB, Müller K. Efficient BackProp. In: Neural Networks: Tricks of the Trade. second ed.. 2012, p. 9–48. http://dx.doi.org/10.1007/978-3-642-35289-8_3.

[35] Duchi J, Hazan E, Singer Y. Adaptive subgradient methods for online learning and stochastic optimization. J Mach Learn Res 2011;12(Jul):2121–59.

[36] Kingma DP, Ba J. Adam: A method for stochastic optimization. 2014, CoRR abs/1412.6980. arXiv:1412.6980, URL http://arxiv.org/abs/1412.6980.

[37] Reddi SJ, Kale S, Kumar S. On the convergence of adam and beyond. In: International conference on learning representations. 2018. URL https://openreview.net/forum?id=ryQu7f-RZ.

[38] James DFV, Kwiat PG, Munro WJ, White AG. Measurement of qubits. Phys. Rev. A 2001;64:052312. http://dx.doi.org/10.1103/PhysRevA.64.052312, URL https://link.aps.org/doi/10.1103/PhysRevA.64.052312.

[39] Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH, Teller E. Equation of state calculations by fast computing machines. J Chem Phys 1953;21(6):1087–92. http://dx.doi.org/10.1063/1.1699114.

[40] Swendsen RH, Wang J-S. Replica Monte Carlo simulation of spin-glasses. Phys Rev Lett 1986;57:2607–9. http://dx.doi.org/10.1103/PhysRevLett.57.2607, URL https://link.aps.org/doi/10.1103/PhysRevLett.57.2607.

[41] Sorella S. Green function Monte Carlo with stochastic reconfiguration. Phys Rev Lett 1998;80:4558–61. http://dx.doi.org/10.1103/PhysRevLett.80.4558, URL https://link.aps.org/doi/10.1103/PhysRevLett.80.4558.

[42] Casula M, Sorella S. Geminal wave functions with Jastrow correlation: A first application to atoms. J Chem Phys 2003;119(13):6500–11. http://dx.doi.org/10.1063/1.1604379.

[43] Amari S-i. Natural gradient works efficiently in learning. Neural Comput 1998;10(2):251–76. http://dx.doi.org/10.1162/089976698300017746.

[44] PyPI – the Python package index. https://pypi.org/. [Accessed 6 March 2019].

[45] Jupyter P, Bussonnier M, Forde J, Freeman J, Granger B, Head T, Holdgraf C, Kelley K, Nalvarte G, Osheroff A, Pacer M, Panda Y, Perez F, Ragan-Kelley B, Willing C. Binder 2.0 - Reproducible, interactive, sharable environments for science at scale. In: Fatih Akici and David Lippa and Dillon Niederhut and M Pacer, editors, Proceedings of the 17th python in science conference. 2018. p. 113–20. http://dx.doi.org/10.25080/Majora-4af1f417-011.

[46] Choo K, Neupert T, Carleo G. Study of the two-dimensional frustrated J1-J2 model with neural network quantum states. arXiv:1903.06713. URL http://arxiv.org/abs/1903.06713.

[47] Vieijra T, Casert C, Nys J, De Neve W, Haegeman J, Ryckebusch J, Verstraete F. Restricted Boltzmann machines for quantum states with nonabelian or anyonic symmetries. arXiv:1905.06034. URL http://arxiv.org/abs/1905.06034.

[48] Pilati S, Inack EM, Pieri P. Self-learning projective quantum Monte Carlo simulations guided by restricted Boltzmann machines. arXiv:1903.00907. URL http://arxiv.org/abs/1907.00907.

[49] Paszke A, Gross S, Chintala S, Chanan G, Yang E, DeVito Z, Lin Z, Desmaison A, Antiga L, Lerer A. Automatic differentiation in PyTorch. In: NIPS-W. 2017.

[50] Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, Ghemawat S, Goodfellow I, Harp A, Irving G, Isard M, Jia Y, Jozefowicz R, Kaiser L, Kudlur M, Levenberg J, Mané D, Monga R, Moore S, Murray D, Olah C, Schuster M, Shlens J, Steiner B, Sutskever I, Talwar K, Tucker P, Vanhoucke V, Vasudevan V, Viégas F, Vinyals O, Warden P, Wattenberg M, Wicke M, Yu Y, Zheng X. TensorFlow: Large-scale machine learning on heterogeneous systems. 2015, Software available from . URL http://tensorflow.org/.

[51] Carleo G, Becca F, Schiro M, Fabrizio M. Localization and glassy dynamics of many-body quantum systems. Sci Rep 2012;2:243. http://dx.doi.org/10.1038/srep00243.

[52] Bauer B, Carr LD, Evertz HG, Feiguin A, Freire J, Fuchs S, Gamper L, Gukelberger J, Gull E, Guertler S, Hehn A, Igarashi R, Isakov SV, Koop D, Ma PN, Mates P, Matsuo H, Parcollet O, Pawłowski G, Picon JD, Pollet L, Santos E, Scarola VW, Schollwöck U, Silva C, Surer B, Todo S, Trebst S, Troyer M, Wall ML, Werner P, Wessel S. The ALPS project release 2.0: open source software for strongly correlated systems. J Stat Mech Theory Exp 2011;2011(05):P05001. http://dx.doi.org/10.1088/1742-5468/2011/05/p05001, URL https://doi.org/10.1088/1742-5468/2011/05/p05001.

[53] Albuquerque A, Alet F, Corboz P, Dayal P, Feiguin A, Fuchs S, Gamper L, Gull E, Gürtler S, Honecker A, Igarashi R, Körner M, Kozhevnikov A, Läuchli A, Manmana S, Matsumoto M, McCulloch I, Michel F, Noack R, Pawłowski G, Pollet L, Pruschke T, Schollwöck U, Todo S, Trebst S, Troyer M, Werner P, Wessel S. The ALPS project release 1.3: Open-source software for strongly correlated systems. J Magn Magn Mater 2007;310(2):1187–93. http://dx.doi.org/10.1016/j.jmmm.2006.10.304, URL https://doi.org/10.1016/j.jmmm.2006.10.304.

[54] Cullum J, Willoughby RA. Computing eigenvalues of very large symmetric matrices—An implementation of a lanczos algorithm with no reorthogonalization. J Comput Phys 1981;44(2):329–58. http://dx.doi.org/10.1016/0021-9991(81)90056-5.

[55] Cullum J, Willoughby RA. Lanczos Algorithms for Large Symmetric Eigenvalue Computations Vol. I Theory (Progress in Scientific Computing) (Volume 1). Birkhäuser; 1985.

Publication P4

[P4] Filippo Vicentini, **D.H.,** Attila Szabó, Dian Wu, Christopher Roth, Clemens Giuliani, Gabriel Pescia, Jannes Nys, Vladimir Vargas-Calderón, Nikita Astrakhantsev, Giuseppe Carleo.
NetKet 3: Machine Learning Toolbox for Many-Body Quantum Systems.
SciPost Physics Codebases 7. DOI 10.21468/SciPostPhysCodeb.7.

Summary

This software paper provides a comprehensive overview of version 3 of the NetKet framework, detailing both the general structure and usage of the framework as well as highlighting central new features of NetKet 3 compared to the previous major release [P3].

Publication status

This work has been published in SciPost Physics Codebases.

Declaration of contributions

For this software paper, I have written several parts of the initial manuscript (particularly in Sections 2–4 and for the unitary time-evolution example in Section 9.1) and contributed to proofreading and discussions. During the preparation of this paper and the development of NetKet 3, I have continued my involvement in the NetKet project as a core contributor. This has included implementing t-VMC using Python/JAX for NetKet 3 (in collaboration with F.V.), contributions to the improved handling of symmetry groups[27], lattice systems, and MCMC statistics, as well as to design discussions, code reviews, and bug fixes concerning various parts of the codebase.

[27]The GCNN implementation [P4, Section 5] builds on this functionality but was primarily written by A.S. and C.R.

NetKet 3: Machine learning toolbox for many-body quantum systems

Filippo Vicentini[1,2,⋆], Damian Hofmann[3], Attila Szabó[4,5], Dian Wu[1,2], Christopher Roth[6], Clemens Giuliani[1,2], Gabriel Pescia[1,2], Jannes Nys[1,2], Vladimir Vargas-Calderón[7], Nikita Astrakhantsev[8] and Giuseppe Carleo[1,2]

1 École Polytechnique Fédérale de Lausanne (EPFL), Institute of Physics, CH-1015 Lausanne, Switzerland
2 Center for Quantum Science and Engineering, École Polytechnique Fédérale de Lausanne (EPFL), CH-1015 Lausanne, Switzerland
3 Max Planck Institute for the Structure and Dynamics of Matter, Center for Free-Electron Laser Science (CFEL), Luruper Chaussee 149, 22761 Hamburg, Germany
4 Rudolf Peierls Centre for Theoretical Physics, University of Oxford, Oxford OX1 3PU, United Kingdom
5 ISIS Facility, Rutherford Appleton Laboratory, Harwell Campus, Didcot OX11 0QX, United Kingdom
6 Physics Department, University of Texas at Austin
7 Grupo de Superconductividad y Nanotecnología, Departamento de Física, Universidad Nacional de Colombia, Bogotá, Colombia
8 Department of Physics, University of Zurich, CH-8057 Zurich, Switzerland

⋆ filippo.vicentini@epfl.ch

Abstract

We introduce version 3 of NetKet, the machine learning toolbox for many-body quantum physics. NetKet is built around neural quantum states and provides efficient algorithms for their evaluation and optimization. This new version is built on top of JAX, a differentiable programming and accelerated linear algebra framework for the Python programming language. The most significant new feature is the possibility to define arbitrary neural network ansätze in pure Python code using the concise notation of machine-learning frameworks, which allows for just-in-time compilation as well as the implicit generation of gradients thanks to automatic differentiation. NetKet 3 also comes with support for GPU and TPU accelerators, advanced support for discrete symmetry groups, chunking to scale up to thousands of degrees of freedom, drivers for quantum dynamics applications, and improved modularity, allowing users to use only parts of the toolbox as a foundation for their own code.

Received 20-12-2021
Accepted 13-06-2022
Published 24-08-2022
doi:10.21468/SciPostPhysCodeb.7

Check for updates

This publication is part of a bundle: Please cite both the article and the release you used.

DOI	Type
doi:10.21468/SciPostPhysCodeb.7	Article
doi:10.21468/SciPostPhysCodeb.7-r3.4	Codebase release

Contents

1 Introduction

During the last two decades, we have witnessed tremendous advances in machine learning (ML) algorithms which have been used to solve previously difficult problems such as image recognition [1, 2] or natural language processing [3]. This has only been possible thanks to sustained hardware development: the last decade alone has seen a 50-fold increase in available computing power [4]. However, unlocking the full computational potential of modern arithmetic accelerators, such as GPUs, used to require significant technical skills, hampering researchers in their efforts. The incredible pace of algorithmic advances must therefore be attributed, at least in part, to the development of frameworks allowing researchers to tap into the full potential of computer clusters while writing high-level code [5, 6].

In the last few years, researchers in quantum physics have increasingly utilized machine-learning techniques to develop novel algorithms or improve on existing approaches [7]. In the context of variational methods for many-body quantum physics in particular, the method of *neural quantum states* (NQS) has been developed [8]. NQS are based on the idea of using neural networks as an efficient parametrization of the quantum wave function. They are of particular interest because of their potential to represent highly entangled states in more than one dimension with polynomial resources [9], which is a significant challenge for more established families of variational states. NQS are also flexible: they have been successfully used to determine variational ground states of classical [10] and quantum Hamiltonians [11–17] as well as excited states [13], to approximate Hamiltonian unitary dynamics [8, 18–23], and to solve the Lindblad master equation [24–26]. In particular, NQS are currently used in the study of frustrated quantum systems [13, 15–17, 27–30], which have so far been challenging to optimize by established numerical techniques. They have also been used to perform tomographic state reconstruction [31] and efficiently approximate quantum circuits [32].

A complication often encountered when working with NQS is, however, that standard ML frameworks like TensorFlow [33] or PyTorch [34] are not geared towards these kind of *quan-*

tum mechanical problems, and it often takes considerable technical expertise to use them for such non-standard tasks. Alternatively, researchers sometimes avoid those frameworks and implement their routines from scratch, but this often leads to sub-optimal performance. We believe that it is possible to foster research at this intersection of quantum physics and ML by providing an easy-to-use interface exposing quantum mechanical objects to ML frameworks. We therefore introduce version 3 of the NetKet framework [35].[1] NetKet 3 is an open-source Python toolbox expressing several quantum mechanical primitives in the differentiable programming framework JAX [5,36].

NetKet provides an easy-to-use interface to high-performance variational techniques without the need to delve into the details of their implementations, but customizability is not sacrificed and advanced users can inspect, modify, and extend practically every aspect of the package. Moreover, integration of our quantum object primitives with the JAX ecosystem allows users to easily define custom neural-network architectures and compute a range of quantum mechanical quantities, as well as their gradients, which are auto-generated through JAX's tracing-based approach. JAX provides the ability to write numerical code in pure Python using NumPy-like calls for array operations, while still achieving high performance through just-in-time compilation using XLA, the accelerated linear algebra compiler that underlies TensorFlow. We have also integrated JAX and MPI with the help of MPI4JAX [37] to make NetKet scale to hundreds of computing nodes.

1.1 What's new

With the release of version 3, NetKet has moved from internally relying on a custom C++ core to the JAX framework, which allows models and algorithms to be written in pure Python and just-in-time compiled for high performance on both CPU and GPU platforms.[2] By using only Python, the installation process is greatly simplified and the barrier of entry for new contributors is lowered.

iFrom a user perspective, the most important new feature is the possibility of writing custom NQS wave functions using JAX, which allows for quick prototyping and deployment, frees users from having to manually implement gradients due to JAX's support for automatic differentiation, and makes models easily portable to GPU platforms. Other prominent new features are

- support for (real and imaginary time) unitary and Markovian dissipative dynamics;
- support for continuous systems;
- support for composite Hilbert spaces;
- efficient implementations of the quantum geometric tensor and stochastic reconfiguration, which scale to models with millions of parameters;
- group-invariant and group-equivariant layers and architectures which support arbitrary discrete symmetries.

A more advanced feature is an extension mechanism built around multiple dispatch [38], which allows users to override algorithms used internally by NetKet without editing the source itself. This can be used to make NetKet work with custom objects and algorithms to study novel problems that do not easily fit what is already available.

[1]This manuscript refers to NetKet v3.5, released in August 2022.

[2]Google's Tensor Processing Units (TPUs) are also, in principle, supported. However, at the time of writing they only support half-precision `float16`. Some modifications would be necessary to work-around loss of precision and gradient underflow.

JAX fluency. Using NETKET's high-level interface and built-in neural network architectures does not require the user to be familiar with JAX and concepts such as just-in-time compilation and automatic differentiation. However, when defining custom classes such as neural network architectures, operators, or Monte Carlo samplers, some proficiency with writing JAX-compatible code will be required. We refrain from discussing JAX in detail and instead point the reader towards its documentation at jax.readthedocs.io.

1.2 Outline

NETKET provides both an intuitive high-level interface with sensible defaults to welcome beginners, as well as a complete set of options and lower-level functions for flexible use by advanced users. The high-level interface is built around quantum-mechanical objects such as *Hilbert spaces* (`netket.hilbert`) and *operators* (`netket.operator`), presented in Section 2.

The central object in NETKET 3 is the *variational state,* discussed in Section 3, which bring together the neural-network ansatz, its variational parameters, and a Monte-Carlo sampler. In Section 3.2, we give an example on how to define an arbitrary neural network using a NETKET/JAX-compatible framework, while Section 3.4 presents the new API of stochastic samplers. In Section 3.5, we show how to compute the quantum geometric tensor (QGT) with NETKET, and compare the different implementations.

Section 4 shows how to use the three built-in optimization drivers to perform ground-state, steady-state, and dynamics calculations. Section 5 discusses NETKET's implementation of spatial symmetries and symmetric neural quantum states, which can be exploited to lower the size of the variational manifold and to target excited states in nontrivial symmetry sectors. In Section 6, we also show how to study a system with continuous degrees of freedom, such as interacting particles in one or more spatial dimensions.

The final sections present detailed workflow examples of some of the more common use cases of NETKET. In Section 7, we show how to study the ground state and the excited state of a lattice Hamiltonian. Section 9 gives examples of both unitary and Lindbladian dynamical simulations.

To conclude, Section 10 presents scaling benchmarks of NETKET running across multiple devices and a performance comparison with JVMC [39], another library similar in scope to NETKET.

Readers who are already familiar with the previous version of NETKET might be especially interested in the variational state interface described in Section 3.1, which replaces what was called *machine* in NETKET 2 [35], the QGT interface described in Section 3.5, algorithms for dynamics (Section 4.3 and Section 9), and symmetry-aware NQS (Section 5).

1.3 Installing NETKET

NETKET is a package written in pure Python; it requires a recent Python version, currently at least version 3.7. Even though NETKET itself is platform-agnostic, JAX, its main dependency, only works on MacOS and Linux at the time of writing.[3] Installing NETKET is straightforward and can be achieved running the following line inside a python environment:

```
pip install --upgrade netket
```

[3]In principle, JAX runs on Windows, but users must compile it themselves, which is not an easy process.

To enable GPU support, Linux with a recent CUDA version is required and a special version of JAX must be installed. As the appropriate installation procedure can change between JAX versions, we refer the reader to the official documentation[4] for detailed instructions.

NetKet by default does not make use of multiple CPUs that might be available to the user. Exploiting multiple processors, or even running across multiple nodes, requires MPI dependencies, which can be installed using the command

```
pip install --upgrade "netket[mpi]"
```

These dependencies, namely `mpi4py` and `mpi4jax`, can only be installed if a working MPI distribution is already available.

Once NetKet is installed, it can be imported in a Python session or script and its version can be checked as

```
>>> import netket as nk
>>> print(nk.__version__)
3.5.0
```

We recommend that users use an up-to-date version when starting a new project. In code listings, we will often refer to the `netket` module as `nk` for brevity.

NetKet also comes with a set of so-called *experimental* functionalities which are packaged into the `netket.experimental` submodule which mirrors the structure of the standard `netket` module. Experimental APIs are marked as such because they are relatively young and we might want to change the function names or options keyword arguments without guaranteeing backward compatibility as we do for the rest of NetKet. In general, we import the experimental submodule as follows

```
>>> from netket import experimental as nkx
```

and use `nkx` as a shorthand for it.

2 Quantum-mechanical primitives

In general, when working with NetKet, the workflow is the following: first, one defines the Hilbert space of the system (Section 2.1) and the Hamiltonian or super-operator of interest (Section 2.2). Then, one builds a variational state (Section 3.1), usually combining a neural-network model and a stochastic sampler. In this section, we describe the first step in this process, namely, how to define a quantum-mechanical system to be modeled.

2.1 Hilbert spaces

Hilbert-space objects determine the state space of a quantum system and a specific choice of basis. Functionality related to Hilbert spaces is contained in the `nk.hilbert` module; for brevity, we will often leave out the prefix `nk.hilbert` in this section.

All implementations of Hilbert spaces derive from the class `AbstractHilbert` and fall into two classes:

- discrete Hilbert spaces, which inherit from the abstract class `DiscreteHilbert` and include spin (`Spin`), qubit (`Qubit`), Fock (`Fock`) as well as fermionic orbitals (`SpinOrbitalFermions`) Hilbert spaces. Discrete spaces are typically used to describe

[4]https://github.com/google/jax#installation

lattice systems. The lattice structure itself is, however, not part of the Hilbert space class and can be defined separately.

- continuous Hilbert spaces, which inherit from the abstract class `ContinuousHilbert`. Currently, the only concrete continuous space provided by NetKet is `Particle`.

Continuous Hilbert spaces are discussed in Section 6. A general discrete space with N sites has the structure

$$\mathcal{H}_{\text{discrete}} = \text{span}\{|s_0\rangle \otimes \cdots \otimes |s_{N-1}\rangle \mid s_i \in \mathcal{L}_i, i \in \{0, \ldots, N-1\}\}, \tag{1}$$

where $\mathcal{L}_i$ is the set of local quantum numbers at site i (e.g., $\mathcal{L} = \{0,1\}$ for a qubit, $\mathcal{L} = \{\pm 1\}$ for a spin-1/2 system in the σ^z basis, or $\mathcal{L} = \{0,1,\ldots,N_{\max}\}$ for a Fock space with up to $N_{\max}$ particles per site). Constraints on the allowed quantum numbers are supported, resulting in Hilbert spaces that are subspaces of Eq. (1). For example, `Spin(1/2, total_sz=0)` creates a spin-1/2 space which only includes configurations $\{s_i\}$ that satisfy $\sum_{i=1}^N s_i = 0$. The corresponding basis states $|s\rangle$ span the zero-magnetization subspace. Similarly, constraints on the total population in Fock spaces are also supported.

Different spaces can be composed to create coupled systems by using the exponent operator (`**`) and the multiplication operator (`*`). For example, the code below creates the Hilbert space of a bosonic cavity with a cutoff of 10 particles at each site, coupled to 6 spin—$\frac{1}{2}$ degrees of freedom.

```
>>> hi = nk.hilbert.Fock(10) * nk.hilbert.Spin(1/2)**6
>>> print("Size of the hilbert space: ", hi.n_states)
Size of the hilbert space:  704
>>> print("Size of the basis: ", hi.size)
Size of the basis:  7
>>> hi.random_state(jax.random.PRNGKey(0), (2,))
DeviceArray([[10., -1.,  1., -1.,  1., -1., -1.],
             [ 9.,  1., -1., -1.,  1., -1.,  1.]], dtype=float32)
```

All Hilbert objects can generate random basis elements through the function `random_state(rng_key, shape, dtype)`, which has the same signature as standard random number generators in JAX. The first argument is a JAX random-generator state as returned by `jax.random.PRNGKey`, while the other arguments specify the number of output states and optionally the JAX data type. In this example, an array with two state vectors has been returned. The first entry of each corresponds to the Fock space and is thus an integer in $\{0,1,\ldots,10\}$, while the rest contains the spin quantum numbers.

Custom Hilbert spaces can be constructed by defining a class inheriting either from `ContinuousHilbert` for continuous spaces or `DiscreteHilbert` for discrete spaces. In the rest of the paper, we will always be working with discrete Hilbert spaces unless stated otherwise.

NetKet also supports working with super-operators, such as the Liouvillian used to define open quantum systems, and variational mixed states. The density matrix is an element of the space of linear operators acting on a Hilbert space, $\mathbb{B}(\mathcal{H})$. NetKet represents this space using the Choi–Jamilkowski isomorphism [40,41] convention $\mathbb{B}(\mathcal{H}) \sim \mathcal{H} \otimes \mathcal{H}$; this "doubled" Hilbert space is implemented as `DoubledHilbert`. Doubled Hilbert spaces behave largely similarly to standard Hilbert spaces, but their bases have double the number of degrees of freedom; for example, super-operators can be defined straightforwardly as operators acting on them.

2.2 Linear operators

NETKET is designed to allow users to work with large systems, beyond the typically small system sizes that are accessible through exact diagonalization techniques. In order to compute expectation values $\langle\hat{O}\rangle$ on such large spaces, we must be able to efficiently represent the operators $\hat{O}$ and work with their matrix elements $\langle\sigma|\hat{O}|\eta\rangle$ without storing them in memory.

NETKET provides different implementations for the operators, tailored for different use cases, which are available in the `netket.operator` submodule. NETKET operators are always defined relative to a specific underlying Hilbert space object and inherit from one of the abstract classes `DiscreteOperator` or `ContinuousOperator`, depending on the classes of supported Hilbert spaces. We defer the discussion of operators acting on a continuous space to Section 6 and focus on discrete-space operators in the remainder of this section.

An operator acting on a discrete space can be represented as a matrix with some matrix elements $\langle\sigma|\hat{O}|\eta\rangle$. As most of those elements are zero in physical systems, a standard approach is to store the operator as sparse matrices, a format that lowers the memory cost by only storing non-zero entries. However, the number of non-zero matrix elements still scales exponentially with the number of degrees of freedom, so sparse matrices cannot scale to the thousands of lattice sites that we want to support, either. For this reason, NETKET uses one of three custom formats to represent operators:

- `LocalOperator` is an implementation that can efficiently represent sums of K-local operators, that is, operators that only act nontrivially on a set of K sites. The memory cost of this format grows linearly with the number of operator terms and the number of degrees of freedom, but it scales exponentially in K.

- `PauliStrings` is an implementation that efficiently represents a product of Pauli X, Y, Z operators acting on the whole system. This format only works with qubit-like Hilbert spaces, but it is extremely efficient and has negligible memory cost.

- `FermionOperator2nd` is an efficient implementation of second-quantized fermionic operators built out of the on-site creation and annihilation operators $f_i^\dagger, f_i$. It works together with `SpinOrbitalFermions` and the equivalent `Fock` spaces.

- Special implementations like `Ising`, which hard-code the matrix elements of the operator. Those are the most efficient, though they cannot be customized at all.

The `nk.operator` submodule also contains ready-made implementations of commonly used operators, such as Pauli matrices, bosonic ladder or projection operators, and common Hamiltonians such as the Heisenberg, or the Bose–Hubbard models.

2.2.1 Manipulating operators

Operators can be manipulated similarly to standard matrices: they can be added, subtracted, and multiplied using standard Python operators. In the example below we show how to construct the operator

$$\hat{O} = \left(\hat{\sigma}_0^x + \hat{\sigma}_1^x\right)^2 = 2(\hat{\sigma}_0^x\hat{\sigma}_1^x + 1), \tag{2}$$

starting from the Pauli X operator acting on the i-th site, σ_i^x, given by the function `nk.operator.spin.sigmax(hi, i)`:

```
1  >>> hi = nk.hilbert.Spin(1/2)**2
2  >>> op = nk.operator.spin.sigmax(hi,0) + nk.operator.spin.sigmax(hi,1)
3  >>> op = op * op
4  >>> op
5  LocalOperator(dim=2, acting_on=[[0], [0, 1], [1]], constant=0,
       dtype=float64)
6  >>> op.to_dense()
7  array([[2., 0., 0., 2.],
8         [0., 2., 2., 0.],
9         [0., 2., 2., 0.],
10        [2., 0., 0., 2.]])
```

Note that each operator requires the Hilbert space object `hi` as well as the specific sites it acts on as constructor arguments. In the last step (line 6), we convert the operator into a dense matrix using the `to_dense()` method; it is also possible to convert an operator into a SciPy sparse matrix using `to_sparse()` .

While it is possible to inspect those operators and (if the Hilbert space is small enough) to convert them to dense matrices, NetKet's operators are built in order to support efficient row indexing, similar to row-sparse (CSR) matrices. Given a basis vector $|\sigma\rangle$ in a Hilbert space, one can efficiently query the list of basis states $|\eta\rangle$ and matrix elements $O(\sigma,\eta)$ such that

$$O(\sigma,\eta) = \langle\sigma|\hat{O}|\eta\rangle \neq 0\,, \tag{3}$$

using the function `operator.get_conn(sigma)` , which returns both the vector of non-zero matrix elements and the corresponding list of indices $|\eta\rangle$, stored as a matrix.[5]

3 Variational quantum states

In this section, we first introduce the general interface of variational states, which can be used to represent both pure states (vectors in the Hilbert space) and mixed states (positive-definite density operators). We then present how to define variational ansätze and the stochastic samplers needed that generate Monte Carlo states.

3.1 Abstract interface

A variational state describes a parametrized quantum state that depends on a (possibly large) set of variational parameters θ. The quantum state can be either pure (denoted as $|\psi_\theta\rangle$) or mixed (written as a density matrix $\hat{\rho}_\theta$). NetKet defines an abstract interface, `netket.vqs.VariationalState` , for such objects; all classes that implement this interface will automatically work with all the high-level drivers (e.g., ground-state optimization or time-dependent variational dynamics) discussed in Section 4. The `VariationalState` interface is relatively simple, as it has only four requirements:

- The parameters θ of the variational state are exposed through the attribute `parameters` and should be stored as an array or a nested dictionary of arrays.

[5]This querying is currently performed in Python code, just-in-time compiled using NUMBA [42], which runs on the CPU. If you run your computations on a GPU with a small number of samples, this might introduce a considerable slowdown. We are aware of this issue and plan to adapt our operators to be indexed directly on the GPU in the future.

- The expectation value $\langle \hat{A} \rangle_\theta$ of an operator $\hat{A}$ can be computed or estimated by the method `expect`.
- The gradient of an expectation value with respect to the variational parameters, $\partial \langle \hat{A} \rangle_\theta / \partial \theta_j$, is computed by the method `expect_and_grad` [6].
- The quantum geometric tensor (Section 3.5) of a variational state can be constructed with the method `quantum_geometric_tensor`.

At the time of writing, NetKet exposes three types of variational state:

- `nk.vqs.ExactState` represents a variational pure state $|\psi_\theta\rangle$ and computes expectation values, gradients and the geometric tensor by performing exact summation over the full Hilbert space.
- `nk.vqs.MCState` (short for *Monte Carlo state*) represents a variational pure state and computes expectation values, gradients and the geometric tensor by performing Markov chain Monte Carlo (MCMC) sampling over the Hilbert space.
- `nk.vqs.MCMixedState` represents a variational mixed state and computes expectation values by sampling diagonal entries of the density matrix.

Variational states based on Monte Carlo sampling are the main tools that we expose to users, together with a wide variety of high-performance Monte Carlo samplers. More details about stochastic estimates and Monte Carlo sampling will be discussed in Section 3.3 and Section 3.4.

Dispatch and algorithm selection. With three different types of variational state and several different operators supported, it is hard to write a well-performing algorithm that works with all possible combinations of types that users might require. In order not to sacrifice performance for generic algorithms, NetKet uses the approach of multiple dispatch based on the plum module [38]. Combined with jax's just-in-time compilation, this solution bears a strong resemblance to the approach commonly used in the Julia language [6].

Every time the user calls `VariationalState.expect` or `.expect_and_grad`, the types of the variational state and the operator are used to select the most specific algorithm that applies to those two types. This allows NetKet to provide generic algorithms that work for all operators, but keeps it easy to supply custom algorithms for specific operator types if desired.

This mechanism is also exposed to users: it is possible to override the algorithms used by NetKet to compute expectation values and gradients without modifying the source code of NetKet but simply by defining new dispatch rules using the syntax shown below.

```python
@nk.vqs.expect.dispatch
def expect(vstate: MCState, operator: Ising):
    # more efficient implementation than default one
    #
    # expectation_value = ...
    #
    return expectation_value
```

[6]For complex-valued parameters $\theta_j \in \mathbb{C}$, `expect_and_grad` returns the conjugate gradient $\partial \langle \hat{A} \rangle_\theta / \partial \theta_j^*$ instead. This is done because the conjugate gradient corresponds to the direction of steepest ascent when optimizing a real-valued function of complex variables [43].

3.2 Defining the variational ansatz

The main feature defining a variational state is the parameter-dependent mapping of an input configuration to the corresponding probability amplitude or, in other words, the quantum wave function (for pure states)

$$(\theta, s) \mapsto \psi_\theta(s) = \langle s|\psi_\theta\rangle \,, \tag{4}$$

or quantum density matrix (for mixed states)

$$(\theta, s, s') \mapsto \rho_\theta(s, s') = \langle s|\,\hat{\rho}_\theta\,\left|s'\right\rangle . \tag{5}$$

In NETKET, this mapping is called a *model* (of the quantum state).[7] In the case of NQS, the model is given by a neural network. For defining models, NETKET primarily relies on FLAX [44], a JAX-based neural-network library[8]. Ansätze are implemented as FLAX modules that map input configurations (the structure of which is determined by the Hilbert space) to the corresponding log-probability amplitudes. For example, pure quantum states are evaluated as

$$\ln \psi_\theta(s) = \texttt{module.apply}\,(\theta, s). \tag{6}$$

The use of log-amplitudes has the benefit that the log-derivatives $\partial \ln \psi_\theta(s)/\partial \theta_j$, often needed in variational optimization algorithms, are directly available through automatic differentiation of the model. It also makes it easier for the model to learn amplitudes with absolute values ranging over several orders of magnitude, which is common for many types of quantum states.

Real and complex amplitudes. NETKET supports both real-valued and complex-valued model outputs. However, since model outputs correspond to log-amplitudes, real-valued networks can only represent states that have exclusively non-negative amplitudes, $\ln \psi_\theta(s) \in \mathbb{R} \Rightarrow \psi_\theta(s) \geq 0$.
Since the input configurations s are real, in many pre-defined NETKET models the data type of the network parameters ($\theta \in \mathbb{R}^{N_\mathrm{P}}$ or $\theta \in \mathbb{C}^{N_\mathrm{P}}$) determines whether an ansatz represents a general or a real non-negative state. This should be kept in mind in particular when optimizing Hamiltonians with ground states that can have negative amplitudes.

3.2.1 Custom models using Flax

The recommended way to define a custom module is to subclass `flax.linen.Module` and to provide a custom implementation of the `__call__` method. As an example, we define a simple one-layer NQS with a wave function of the form

$$\ln \psi(s) = \sum_{j=1}^{M} \tanh[Ws + b]_j \,, \tag{7}$$

[7]The notion of "model" in NETKET 3 is related to the "machine" classes in NETKET 2 [35]. However, while NETKET 2 machines both define the mapping (4) and store the current parameters, this has been decoupled in NETKET 3. The model only specifies the mapping, while the parameters are stored in the variational state classes.

[8]While our primary focus has been the support of FLAX, NETKET can in principle be used with *any* JAX-compatible neural network model. For example, NETKET currently includes a compatibility layer which ensures that models defined using the HAIKU framework by DeepMind [45] will work automatically as well. Furthermore, any model represented by a pair of `init` and `apply` functions (as used, e.g., in the STAX framework included with JAX) is also supported.

with the number of visible units N matching the number of physical sites, a number of hidden units M, and complex parameters $W \in \mathbb{C}^{M\times N}$ (the weight matrix) and $b \in \mathbb{C}^{M}$ (the bias vector). Using NETKET and FLAX, this ansatz can be implemented as follows:

```
1  import netket as nk
2  import jax.numpy as jnp
3  import flax
4  import flax.linen as nn
5
6  class OneLayerNQS(nn.Module):
7      # Module hyperparameter:
8      n_hidden_units: int
9
10     @nn.compact
11     def __call__(self, s):
12         n_visible_units = s.shape[-1]
13         # define parameters
14         # the arguments are: name, initializer, shape, dtype
15         W = self.param(
16             "weights",
17             nn.initializers.normal(),
18             (self.n_hidden_units, n_visible_units),
19             jnp.complex128,
20         )
21         b = self.param(
22             "bias",
23             nn.initializers.normal(),
24             (self.n_hidden_units,),
25             jnp.complex128,
26         )
27
28         # multiply with weight matrix over last dimension of s
29         y = jnp.einsum("ij,...j", W, s)
30         # add bias
31         y += b
32         # apply tanh activation and sum
33         y = jnp.sum(jnp.tanh(y), axis=-1)
34
35         return y
```

The decorator `flax.linen.compact` used on `__call__` (line 10) makes it possible to define the network parameters directly in the body of the call function via `self.param` as done above (lines 15 and 21). For performance reasons, the input to the module is batched. This means that, instead of passing a single array of quantum numbers s of size N, a batch of multiple state vectors is passed as a matrix of shape `(batch_size, N)`. Therefore, operations like the sum over all feature indices in the example above need to be explicitly performed over the last axis.

Just-in-time compilation. Note that the network will be just-in-time (JIT) compiled to efficient machine code for the target device (CPU, GPU, or TPU) using `jax.jit`, which means that all code inside the `__call__` method needs to written in a way compatible with `jax.jit`.

In particular, users should use `jax.numpy` for NumPy calls that need to happen at runtime; explicit Python control flow, such as `for` loops and `if` statements, should also be avoided, unless one explicitly wants to have them evaluate once at compile time. We refer users to the JAX documentation for further information on how to write efficient JIT-compatible code.

The module defined above can be used by first initializing the parameters using `module.init` and then computing log-amplitudes through `module.apply`:

```
>>> module = OneLayerNQS(n_hidden_units=16)
# init takes two arguments, a PRNG key for random initialization
# and a dummy array used to determine the input shape
# (here with a batch size of one):
>>> params = module.init(nk.jax.PRNGKey(0), jnp.zeros((1, 8)))
>>> module.apply(params, jnp.array([[-1, 1, -1, 1, -1, 1, -1, 1]]))
DeviceArray([-0.00047843+0.07939122j], dtype=complex128)
```

3.2.2 Network parametrization and pytrees

Parameter data types. NetKet supports models with both real-valued and complex-valued network parameters. The data type of the parameters does not determine the output type. It is possible to define a model with real parameters that produces complex output. A simple example is the sum of two real-valued and real-parameter networks, representing real and imaginary part of the log-amplitudes (and thus phase and absolute value of the wave function) [29,31]:

$$\ln \psi_{(\theta,\eta)}(s) = f_\theta(s) + i g_\eta(s) \quad \Leftrightarrow \quad |\psi_{(\theta,\eta)}(s)| = \exp[f_\theta(s)], \quad \arg \psi_{(\theta,\eta)}(s) = g_\eta(s), \tag{8}$$

(where all $\theta_i, \eta_i \in \mathbb{R}$ and $f(s), g(s) \in \mathbb{R}$).

More generally, any model with N_p complex parameters $\theta = \alpha + i\beta$ can be represented by a model with $2N_\mathrm{p}$ real parameters (α, β). While these parametrizations are formally equivalent, the choice of complex parameters can be particularly useful in the case where the variational mapping is holomorphic or, equivalently, complex differentiable with respect to θ. This is the case for many standard network architectures such as RBMs or feed-forward networks, since both linear transformations and typical activation functions are holomorphic (such as tanh, cosh, and their logarithms) or piecewise holomorphic (such as ReLU), which is sufficient in practice. Note, however, that there are also common architectures, such as autoregressive networks, that are not holomorphic. In the holomorphic case, the computational cost of differentiating the model, e.g., to compute the quantum geometric tensor (Section 3.5), can be reduced by exploiting the Cauchy–Riemann equations [46],

$$i \nabla_\alpha \psi_{(\alpha,\beta)}(s) = \nabla_\beta \psi_{(\alpha,\beta)}(s) \,. \tag{9}$$

Note that NetKet generally supports models with *arbitrary parametrizations* (i.e., real and both holomorphic and non-holomorphic complex parametrizations). The default assumption is that models with complex weights are non-holomorphic, but some objects (most notably the quantum geometric tensor) accept a flag `holomorphic=True` to enable a more efficient code path for holomorphic networks.

It is the user's responsibility to only set `holomorphic=True` for models that are, in fact, holomorphic. If this is incorrectly specified, NETKET code may give incorrect results. To check whether a specific architecture is holomorphic, one can verify the condition

$$\partial\psi_\theta(s)/\partial\theta_j^* = (\partial/\partial\alpha_j + i\partial/\partial\beta_j)\psi_\theta(s) = 0\,, \tag{10}$$

which is equivalent to Eq. (9).

Pytrees. In NETKET, model parameters do not need to be stored as a contiguous vector. Instead, models can support any collection of parameters that forms a so-called *pytree*. Pytree is JAX terminology for collections of numerical tensors stored as the leaf nodes inside layers of nested standard Python containers (such as lists, tuples, and dictionaries).[9] Any object that is not itself a pytree, in particular NumPy or JAX arrays, is referred to as a *leaf*. Networks defined as FLAX modules store their parameters in a (potentially nested) dictionary, which provides name-based access to the network parameters.[10] For the `OneLayerNQS` defined above, the parameter pytree has the structure:

```
# For readability, the actual array data has been replaced with ...
  below.
>>> print(params)
FrozenDict({
    params: {
        weights: DeviceArray(..., dtype=complex128),
        bias: DeviceArray(..., dtype=complex128),
    },
})
```

The names of the entries in the parameter dictionary correspond to those given in the `param` call when defining the model. NETKET functions often work directly with both plain arrays and pytrees of arrays. Furthermore, any Python function can be applied to the leaves of a pytree using `jax.tree_map`. For example, the following code prints a pytree containing the shape of each leaf of `params`, preserving the nested dictionary structure:

```
>>> print(jax.tree_map(jnp.shape, params))
FrozenDict({
    params: {
        weights: (16, 8),
        bias: (16,),
    },
})
```

Functions accepting multiple leaves as arguments can be mapped over the corresponding number of pytrees (with compatible structure) using `jax.tree_map`. For example, the difference of two parameter pytrees of the same model can be computed using

[9] See https://github.com/google/jax/blob/jax-v0.2.28/docs/pytrees.md for a detailed introduction of pytrees.

[10] Specifically, FLAX stores networks parameters in an immutable `FrozenDict` object, which otherwise has the same semantics as a standard Python dictionary and, in particular, is also a valid pytree. The parameters can be modified by converting to a standard mutable `dict` via `flax.core.unfreeze(params)`.

Table 1: List of models included in NetKet's `nk.models` submodule, together with relevant references.

Name	NetKet class	References
Jastrow ansatz	`Jastrow`	[47,48]
Restricted Boltzmann machine (RBM)	`RBM`, `RBMMultiVal`, `RBMModPhase`	[8]
Symmetric RBM	`RBMSymm`	[8]
Group-Equivariant Convolutional Neural Network	`GCNN`	Section 5 [49]
Autoregressive Neural Network	`ARNNDense`, `ARNNConv1D`, `ARNNConv2D`, `FastARNNConv1D`, `FastARNNConv2D`	[50]
Neural Density Matrix	`NDM`	[24,51]

`delta = jax.tree_map(lambda a, b: a - b, params1, params2)`. NetKet provides an additional set of utility functions to perform linear algebra operations on such pytrees in the `nk.jax` submodule.

3.2.3 Pre-defined ansätze included with NetKet

NetKet provides a collection of pre-defined modules under `nk.models`, which allow quick access to many commonly used NQS architectures (Table 1):

- **Jastrow:** The Jastrow ansatz [47,48] is an extremely simple yet effective many-body ansatz that can capture some inter-particle correlations. The log-wavefunction is the linear function $\log\psi(\sigma) = \sum_i \sigma_i W_{i,j}\sigma_j$. Evaluation of this ansatz is very fast but it is also the least powerful model implemented in NetKet;

- **RBM:** The restricted Boltzmann machine (RBM) ansatz is composed by a dense layer followed by a nonlinearity. If the Hilbert space has N degrees of freedom of size d, `RBM` has αN features in the dense layer. This ansatz requires `param_dtype=complex` to represent states that are non-positive valued. `RBMMultiVal` is a one-hot encoding layer followed by an `RBM` with $\alpha d N$ features in its dense layer. Finally, `RBMModPhase` consists of two real-valued RBMs that encode respectively the modulus and phase of the wavefunction as $\log\psi(\sigma) =$ `RBM` $(\sigma) + i$ `RBM` (σ). This ansatz only supports real parameters. If considering Hilbert spaces with local dimension $d > 2$, plain RBMs usually require a very large feature density α and `RBMMultiVal` s perform better.

- **RBMSymm:** A symmetry-invariant RBM. Only symmetry groups that can be represented as permutations of the computational basis are supported (see Section 5). This architecture has fewer parameters than an RBM, but it is more expensive to evaluate. It requires `param_dtype=complex` to represent states that are non-positive valued.

- **GCNN:** A symmetry-equivariant feed-forward network (see Section 5.2). Only symmetry groups that can be represented as permutations of the computational basis are supported. This model is much more complex and computationally intensive than `RBMSymm`, but can also lead to more accurate results. It can also be used to target an excited state of a lattice Hamiltonian. When working with states that are real but non-positive, one can use real parameters together with `complex_output=True`. If the states are to have a complex phase, `param_dtype=complex` is required.

- **Autoregressive networks:** ARNNs are models that can produce uncorrelated samples when sampled with `nk.sampler.ARNNSampler`. Those architectures can be efficiently sampled on GPUs, but they are much more expensive than traditional RBMs.

- **NDM:** A positive-semidefinite density matrix ansatz, comprised of a component describing the pure part and one describing the mixed part of the state. The pure part is equivalent to an RBM with feature density α, while the mixed part is an RBM with feature density β. This network only supports real parameters.

3.2.4 Custom layers included with Flax and NetKet

The `nk.nn` submodule contains generic modules such as masked dense, masked convolutional and symmetric layers to be used as building blocks for custom neural networks. Those layers are complementary to those provided by FLAX and can be combined together to develop novel neural-network architectures.[11]

As an example, a multi-layer NQS with two convolutional and one final dense layer acting as a weighted sum can be defined as follows:

```python
class MultiLayerCNN(nn.Module):
    features1: int
    features2: int
    kernel_size: int

    @nn.compact
    def __call__(self, s):
        # define layers
        layer1 = nn.Conv(
            features=self.features1,
            kernel_size=self.kernel_size,
        )
        layer2 = nn.Conv(
            features=self.features2,
            kernel_size=self.kernel_size,
        )
        weighted_sum = nn.Dense(features=1)

        # apply layers and tanh activations
        y = jnp.tanh(layer1(s))
        y = jnp.tanh(layer2(y))
        y = weighted_sum(y)
        # last axis only has one entry, so we just return that
        # but keep the batch dimension
```

[11]In the past FLAX had minor issues with complex numbers and therefore NetKet included versions of some standard layers, such as `Dense` and `Conv`, that handle complex numbers properly. Starting with FLAX version 0.5, released in May 2022, those issues have been addressed and we now recommend the use of FLAX layers also with complex numbers.

```
            return y[..., 0]
```

FLAX network layers are available from the `flax.linen` submodule (imported as `nn` in the example above), NETKET layers from `netket.nn`.

3.3 Estimating observables

For any variational ansatz, it is crucial to also have efficient algorithms for computing quantities of interest, in particular observables and their gradients. Since evaluating the wave function on all configurations is infeasible for larger Hilbert spaces, NQS approaches rely on Monte Carlo sampling of quantum expectation values.

Pure states. The quantum expectation value of an operator $\hat{A}$ on a non-normalized pure state $|\psi\rangle$ can be written as a classical expectation value over the Born distribution $p(s) \propto |\psi(s)|^2$ using the identity

$$\langle \hat{A} \rangle = \frac{\langle \psi | \hat{A} | \psi \rangle}{\langle \psi | \psi \rangle} = \sum_s \frac{|\psi(s)|^2}{\langle \psi | \psi \rangle} \tilde{A}(s) = \sum_s p(s) \tilde{A}(s) = \mathbb{E}[\tilde{A}], \tag{11}$$

where $\tilde{A}$ is the *local estimator*

$$\tilde{A}(s) = \frac{\langle s | \hat{A} | \psi \rangle}{\langle s | \psi \rangle} = \sum_{s'} \frac{\psi(s')}{\psi(s)} \langle s | \hat{A} | s' \rangle, \tag{12}$$

also known as the *local energy* when $\hat{A}$ is the Hamiltonian [52]. Even though the sum in Eq. (12) runs over the full Hilbert space basis, the local estimator can be efficiently computed if the operator is sufficiently sparse in the given basis, i.e., all but a tractable number of matrix elements $\langle s | \hat{A} | s' \rangle$ are zero. Thus, an efficient algorithm is required that, given s, yields all connected configurations s' together with their respective matrix elements, as described in Section 2.2. Given the derivatives of the log-amplitudes

$$O_i(s) = \frac{\partial \ln \psi_\theta(s)}{\partial \theta_i}, \tag{13}$$

gradients of expectation values can also be evaluated. Define the *force vector* as the covariance

$$\tilde{f}_i = \mathrm{Cov}[O_i, \tilde{A}] = \mathbb{E}[O_i^*(\tilde{A} - \mathbb{E}[\tilde{A}])]. \tag{14}$$

Then, if $\theta_i \in \mathbb{R}$ is a real-valued parameter,

$$\frac{\partial \langle \hat{A} \rangle}{\partial \theta_i} = 2\,\mathrm{Re}[\tilde{f}_i]. \tag{15}$$

If $\theta_i \in \mathbb{C}$ and the mapping $\theta_i \mapsto \psi_\theta(s)$ is complex differentiable (holomorphic),

$$\frac{\partial \langle \hat{A} \rangle}{\partial \theta_i^*} = \tilde{f}_i. \tag{16}$$

In case of a non-holomorphic mapping, $\mathrm{Re}[\theta_i]$ and $\mathrm{Im}[\theta_i]$ can be treated as two independent real parameters and Eq. (15) applies to each.

The required classical expectation values are then estimated by averaging over a sequence $\{s_i\}_{i=1}^{N_s}$ of configurations distributed according to the Born distribution $p(s) \propto |\psi(s)|^2$; e.g., Eq. (11) becomes

$$\mathbb{E}[\tilde{A}] \approx \frac{1}{N_s} \sum_{i=1}^{N_s} \tilde{A}(s). \tag{17}$$

For some models, in particular autoregressive neural networks [50], one can efficiently draw samples from the Born distribution directly. For a general ansatz, however, this is not possible and Markov-chain Monte Carlo (MCMC) sampling methods [52] must be used: these generate a sequence (Markov chain) of samples that asymptotically follows the Born distribution. Such a chain can be generated using the Metropolis–Hastings algorithm [53], which is implemented in NetKet's sampler interface, described in the next section.

Mixed states. When evaluating observables for mixed states, it is possible to exploit a slightly different identity,

$$\langle \hat{A} \rangle = \frac{\mathrm{Tr}\big[\hat{\rho}\hat{A}\big]}{\mathrm{Tr}[\hat{\rho}\,]} = \sum_s \frac{\rho(s,s)}{\mathrm{Tr}[\hat{\rho}\,]}\tilde{A}_\rho(s) = \mathbb{E}[\tilde{A}_\rho], \tag{18}$$

which rewrites the quantum expectation value as a classical expectation over the probability distribution defined by the diagonal of the density matrix $p(s) \propto \rho(s,s)$. Here, $\tilde{A}_\rho$ denotes the local estimator of the observable *over a mixed state,*

$$\tilde{A}_\rho(s) = \frac{\langle s|\hat{\rho}\hat{A}|s\rangle}{\langle s|\,\hat{\rho}\,|s\rangle} = \sum_{s'} \frac{\rho(s,s')}{\rho(s,s)}\langle s'|\hat{A}|s\rangle\,. \tag{19}$$

It is then possible to follow the same procedures detailed in the previous paragraph for pure states to compute the gradient of an expectation value of an operator over a mixed state by replacing the probability distribution over which the average is computed and the local estimator.

3.3.1 Reducing memory usage with chunking

The number of variational state evaluations required to compute the local estimators (12) typically scales superlinearly[12] in the number of sites N. For optimal performance, NetKet by default performs those evaluations in a single call using batched inputs. However, for large Hilbert spaces or very deep models it might be impossible to fit all required intermediate buffers into the available memory, leading to out-of-memory errors. This is encountered particularly often in calculations on GPUs, which have more limited memory.

To avoid those errors, NetKet's `nk.vqs.VariationalState` exposes an attribute called `chunk_size`, which controls the maximum number of configurations for which a model is evaluated at the same time[13]. The chunk size effectively bounds the maximum amount of memory required to evaluate the variational function at the expense of an increased computational cost in some operations involving the derivatives of the model. For this reason, we suggest using the largest chunk size that fits in memory.

Chunking is supported for the majority of operations, such as computing expectation values and their gradients, as well as the evaluation the quantum geometric tensor. If a chunk size is specified but an operation does not support it, NetKet will print a warning and attempt to perform the operation without chunking.

3.4 Monte Carlo samplers

The sampling algorithm used to obtain a sequence of configurations from the probability distribution defined by the variational ansatz is specified by sampler classes inheriting from

[12]The exact scaling depends on the sparsity of the observable in the computational basis (which in a lattice model primarily depends on the locality of the operator and the dimension of the lattice).

[13]The chunk size can be specified at model construction and freely changed later. Chunking can also be disabled at any time by setting `VariationalState.chunk_size = None`.

`nk.sampler.AbstractSampler`. Following the purely functional design of JAX, we define the sampler to be a stateless collection of settings and parameters, while storing all mutable state such as the PRNG key and the statistics of acceptances in an immutable sampler state object. Both the sampler and the sampler state are stored in the variational state, but they can be used independently, as they are decoupled from the rest of NetKet.

The Metropolis–Hastings algorithm is used to generate samples from an arbitrary probability distribution. In each step, it suggests a transition from the current configuration s to a proposed configuration s'. The proposal is accepted with probability

$$P_{\text{acc}}(s \to s') = \min\left(1, \frac{P(s')}{P(s)} \frac{g(s \mid s')}{g(s' \mid s)}\right), \tag{20}$$

where P is the distribution being sampled from and $g(s' \mid s)$ is the conditional probability of proposing s' given the current s. We use $L(s,s') = \log[g(s \mid s')/g(s' \mid s)]$ to denote the correcting factor to the log probability due to the transition kernel. This factor is needed for asymmetric kernels that might propose one move with higher probability than its reverse. Simple kernels, such as a local spin flip or exchange, are symmetric, therefore $L(s,s') = L(s',s) = 1$, but other proposals, such as Hamiltonian sampling, are not necessarily symmetric and need this factor.

At the time of writing, NetKet exposes four types of rules to use with the Metropolis sampler: `MetropolisLocal`, which changes one discrete local degree of freedom in each transition; `MetropolisExchange`, which exchanges two local degrees of freedom respecting a conserved quantity (e.g., total particle number or magnetization); `MetropolisHamiltonian`, which transitions the configuration according to the off-diagonal elements of the Hamiltonian; and `MetropolisGaussian`, which moves a configuration with continuous degrees of freedom according to a Gaussian distribution.

The different transition kernels in these samplers are represented by `MetropolisRule` objects. To define a Metropolis sampling algorithm with a new transition kernel, one only needs to subclass `MetropolisRule` and implement the `transition` method, which gives s' and $L(s,s')$ in each transition. For example, the following transition rule changes the local degree of freedom on two sites at a time:

```python
from netket.hilbert.random import flip_state
from netket.sampler import MetropolisRule
from netket.utils.struct import dataclass

# To be jax-compatible, it must be a dataclass
@dataclass
class TwoLocalRule(MetropolisRule):
    def transition(rule, sampler, machine, parameters, state, key, σ):
        # Deduce the number of MCMC chains from input shape
        n_chains = σ.shape[0]
        # Load the Hilbert space of the sampler
        hilb = sampler.hilbert
        # Split the rng key into 2: one for each random operation
        key_indx, key_flip = jax.random.split(key, 2)
        # Pick two random sites on every chain
        indxs = jax.random.randint(
            key_indx, shape=(n_chains, 2), minval=0, maxval=hilb.size
        )
        # flip those sites
        σp, _ = flip_state(hilb, key_flip, σ, indxs)

```

Table 2: List of samplers in NETKET with their class names and a description.

Type	Name	Usage
MCMC (Metropolis)	`MetropolisLocal`	discrete Hilbert spaces
	`MetropolisExchange`	permutations of local states, conserving total magnetization in spin systems
	`MetropolisHamiltonian`	preserving symmetries of the Hamiltonian
	`MetropolisGaussian`	continuous Hilbert spaces
Direct	`ExactSampler`	small Hilbert spaces, performs MC sampling from the exact distribution
	`ARDirectSampler`	autoregressive models

```
        # If this transition had a correcting factor L, it’s possible
        # to return it as a vector in the second value
        return σp, None
```

Once a custom rule is defined, a MCMC sampler using such rule can be constructed with the command `sampler = MetropolisSampler(hilbert, TwoLocalRule())` . Besides Metropolis algorithms, more advanced Markov chain algorithms can also be implemented as NETKET samplers. Currently, parallel tempering is provided as an experimental feature.

Some models allow us to directly generate samples that are exactly distributed according to the desired probability, without the use of Markov chains and the issue of autocorrelation, which often leads to more efficient sampling. In this case, direct samplers can be implemented with an interface similar to Markov chain samplers. Currently NETKET has implemented `ARDirectSampler` to be used with ARNNs. For benchmarking purposes, NETKET also provides `ExactSampler` , which allows direct sampling from any model by computing the full Born distribution $|\psi(s)|^2$ for all s. Table 2 is a list of all the samplers.

3.5 Quantum geometric tensor

The quantum geometric tensor (QGT) [54] of a pure state is the metric tensor induced by the Fubini–Study distance [55, 56]

$$d(\psi,\phi) = \cos^{-1}\sqrt{\frac{\langle\psi|\phi\rangle\langle\phi|\psi\rangle}{\langle\psi|\psi\rangle\langle\phi|\phi\rangle}}, \tag{21}$$

which is the natural and gauge-invariant distance between two pure quantum states $|\psi\rangle$ and $|\phi\rangle$. The QGT is commonly used for time evolution (see Section 4.3) and for quantum natural gradient descent [57], which was originally developed in the VMC community under the name of *stochastic reconfiguration* (SR) [58]. Quantum natural gradient descent is directly related to the natural gradient descent developed in the machine learning community [59].

From now on, we assume that the state $|\psi_\theta\rangle$ is parametrized by a set of parameters θ. Assuming further that $|\phi\rangle = |\psi_{\theta+\delta\theta}\rangle$, the distance (21) can be expanded to second order in the infinitesimal parameter change $\delta\theta$ as $d(\psi_\theta,\psi_{\theta+\delta\theta})^2 = (\delta\theta)^\dagger G(\delta\theta)$, where G is the quantum geometric tensor. For a holomorphic mapping $\theta \mapsto |\psi_\theta\rangle$, the QGT is given by

$$G_{ij}(\theta) = \frac{\left\langle \partial_{\theta_i}\psi_\theta \middle| \partial_{\theta_j}\psi_\theta \right\rangle}{\langle \psi|\psi\rangle} - \frac{\left\langle \partial_{\theta_i}\psi_\theta \middle| \psi_\theta \right\rangle \left\langle \psi_\theta \middle| \partial_{\theta_j}\psi_\theta \right\rangle}{\langle \psi|\psi\rangle^2}, \tag{22}$$

where the indices i, j label the parameters and $\left\langle x \middle| \partial_{\theta_i}\psi_\theta \right\rangle = \partial_{\theta_i} \langle x|\psi_\theta\rangle$. Similar to expectation values and their gradients, Eq. (22) can be rewritten as a classical covariance with respect to the Born distribution $\propto |\psi(s)|^2$:

$$G_{ij}(\theta) = \mathrm{Cov}[O_i, O_j] = \mathbb{E}\left[O_i^*(O_j - \mathbb{E}[O_j])\right], \tag{23}$$

where O_i are the log-derivatives (13) of the ansatz.[14] This allows the quantum geometric tensor to be estimated using the same sampling procedure used to obtain expectation values and gradients. The QGT or its stochastic estimate is also commonly known as the *S matrix* [52] or *quantum Fisher matrix* (QFM) in analogy to the classical Fisher information matrix [57, 59, 60].

For applications such as quantum natural gradient descent or time-evolution it is usually not necessary to access the full, dense matrix representation $G_{ij}(\theta)$ of the quantum geometric tensor, but only to compute its product with a vector, $\tilde{v}_i = \sum_j G_{ij}(\theta)\, v_j$. When the variational ansatz ψ_θ has millions of parameters, the QGT can indeed be too large to be stored in memory. Exploiting the Gram matrix structure of the geometric tensor [61], we can directly compute its action on a vector without ever calculating the full matrix, trading memory requirements for an increased computational cost.

Given a variational state `vs`, a QGT object can be obtained by calling:

```
>>> qgt = vs.quantum_geometric_tensor()
```

This `qgt` object does not store the full matrix, but can still be applied to a vector with the same shape as the parameters:

```
>>> vec = jax.tree_map(jnp.ones_like, vs.parameters)
>>> qgt_times_vec = qgt @ vec
```

It can be converted to a dense matrix by calling `to_dense`:

```
# get the matrix (2d array) of the qgt
>>> qgt_dense = qgt.to_dense()
# flatten vec into a 1d Array
>>> grad_dense, unravel = nk.jax.tree_ravel(vec)
>>> qgt_times_vec = unravel(qgt_dense @ vec_dense)
```

The QGT can then be used together with a direct solver, such as `jnp.linalg.eigh`, `jnp.linalg.svd`, or `jnp.linalg.qr`.

Mixed states. When working with mixed states, which are encoded in a density matrix, it is necessary to pick a suitable metric to induce the QGT. Even if the most physical distances for density matrices are the spectral norm or other trace-based norms [62, 63], it is generally hard to use them to define an expression for the QGT that can be efficiently sampled and computed at polynomial cost. While this might be regarded as a *barbaric* choice, it leads to an expression equivalent to Eq. (23), where the expectation value is over the joint-distribution of the of row

[14] Strictly speaking, this estimator is only correct if $\psi_\theta(s) = 0 \implies \partial_{\theta_k}\psi_\theta(s) = 0$. This is because we multiplied and divided by $\psi_\theta(s)$ in the derivation of the estimator, which is only valid if $\psi_\theta(s) \neq 0$.

and column labels (s, s') of the squared density matrix $\propto \left|\rho(s, s')\right|^2$. Therefore, when working with mixed states, we resort to the L_2 norm, which is equivalent to treating the density matrix as a vector ("pure state") in an enlarged Hilbert space.

3.5.1 Implementation differences

There is some freedom in the way one can calculate the QGT, each different implementation taking a different tradeoff between computational and memory cost. In the example above, we have relied on NETKET to automatically select the best implementation according to some internal heuristics, but if one wants to push variational methods to the limits, it is useful to understand the two different implementations on offer:

- `QGTJacobian` , which computes and stores the Jacobian $O_i(s)$ of the model (cf. Appendix B.1) at initialization (using reverse-mode automatic differentiation) and can be applied to a vector by performing two matrix-vector multiplications;
- `QGTOnTheFly` , which lazily computes the application of the quantum geometric tensor on a vector through automatic differentiation.

QGTOnTheFly is the most flexible and can be scaled to arbitrarily large systems. It is based on the observation that the QGT is the Jacobian of the model multiplied with its conjugate, which means that its action can be calculated by combining forward and reverse-mode automatic differentiation. At initialization, it only computes the linearization (forward pass), and then it effectively recomputes the gradients every time it is applied to a vector. However, since it never has to store these gradients, it is not limited by the available memory, which also makes it perform well for shallow neural-network models like RBMs. This method works with both holomorphic and non-holomorphic ansätze with no difference in performance.

QGTJacobian. For deep networks with ill-conditioned[15] quantum geometric tensors, recomputing the gradients at every step in an iterative solver might be very costly. `QGTJacobian` can therefore achieve better performance at the cost of considerably higher memory requirements because it precomputes the Jacobian at construction and stores it. The downside is that it has to store a *matrix* of shape $N_{\text{samples}} \times N_{\text{parameters}}$, which might not fit in the memory of a GPU. We note that there are two different implementations of `QGTJacobian` : `QGTJacobianDense` and `QGTJacobianPyTree` . The difference among the two is that in the former the Jacobian is stored contiguously in memory, leading to a better throughput on GPUs, while the latter stores them in the same structure as the parameters (so each parameter block is separated from the others). Converting from the non-contiguous (`QGTJacobianPyTree`) to the contiguous (`QGTJacobianDense`) format has, however, a computational and memory cost which might shadow its benefit. Moreover, the dense format does not work with non-homogeneous parameter data types. The basic `QGTJacobian` algorithm supports both holomorphic and non-holomorphic NQS, but a better performing algorithm for holomorphic ansätze can be accessed instantiating it with the option `holomorphic=True` .

Key differences between the different QGT implementations are summarized in Table 3. Implementations can be selected, and options passed to them, as shown below:

[15]The number of steps required to find a solution with an iterative linear solver grows with the condition number of the matrix. Therefore, an ill-conditioned matrix requires many steps of iterative solver. For a discussion on this issue, see the paragraph on *Linear Systems* of this section.

Table 3: Overview of the three QGT implementations currently provided by NetKet with their respective options and limitations.

Implementation	Extra arguments	Use-cases	Limitations
`QGTOnTheFly`	None	shallow networks, large numbers of parameters and samples, few solver steps	Might be more computationally expensive for deep networks compared to `QGTJacobian`.
`QGTJacobianDense`	`mode` `holomorphic` `rescale_shift`	deep networks with narrow layers	requires homogeneous parameter types, memory bound
`QGTJacobianPyTree`		deep networks with heterogeneous parameters	memory bound

```
>>> from netket.optimizer.qgt import (
...     QGTJacobianPyTree, QGTJacobianDense, QGTOnTheFly
... )
>>> qgt1 = vs.quantum_geometric_tensor(QGTOnTheFly)
>>> qgt2 =
    vs.quantum_geometric_tensor(QGTJacobianPyTree(holomorphic=True))
>>> qgt3 = vs.quantum_geometric_tensor(QGTJacobianDense)
```

Holomorphicity. When performing time evolution or natural gradient descent, one does not always need the full quantum geometric tensor: for ansätze with real parameters, as well as in the case of non-holomorphic wave functions,[16] only the real part of the QGT is used. The real and imaginary parts of the QGT are only required when working with a holomorphic ansatz. (An in-depth discussion of why this is the case can be found at [64, Table 1].) For this reason, NetKet's QGT implementations return the full geometric tensor only for holomorphic complex-parameter ansätze, and its real part in all other cases.

3.5.2 Solving linear systems

For most applications involving the QGT, a linear system of equations of the kind

$$\sum_j G_{ij}\delta_j = f_i\,, \tag{24}$$

needs to be solved, where $G_{i,j}$ is the quantum geometric tensor of a NQS, and f_i is a gradient. This can be done using the standard JAX/NumPy functions, assuming f is a pytree with the same structure as the variational parameters:

```
>>> # iterative solver
>>> x, info = jax.scipy.sparse.linalg.cg(qgt, f)
>>> # direct solver, acting on the dense matrix
>>> x, info = jax.numpy.linalg.cholesky(qgt.to_dense(), f)
```

[16]Non-holomorphic functions of complex parameters are internally handled by both JAX and NetKet as real-parameter functions that take the real and imaginary parts of the "complex parameters" separately.

However, we recommend that users call the `solve` method on the QGT object, which allows some additional optimization that may improve performance:

```
>>> x, info = qgt.solve(jax.scipy.sparse.linalg.cg, f)
>>> x, info = qgt.solve(nk.optimizer.solver.cholesky, f)
```

While this works with any of the representations it is advisable to only use Jacobian based implementations (`QGTJacobianPyTree` or `QGTJacobianDense`) with direct solvers, since constructing the QGT matrix from `QGTOnTheFly` requires multiplication with all basis vectors, which is not as efficient. Finally, we highlight the fact that users can write their own functions to solve the linear system (24) using advanced regularization schemes (see for instance ref. [21]) and use them together with `qgt.solve` , as long as they respect the standard NumPy solver interface.

When working with iterative solvers such as *cg*, *gmres* or *minres*, the number of steps required to find a solution grows with the condition number of the matrix. Therefore, an ill-conditioned geometric tensor requires many steps of iterative solver, increasing the computational cost. Even then or when using non-iterative methods such as singular value decomposition (SVD), the high condition number can cause instabilities by amplifying noise in the right-hand side of the linear equation [20, 21, 23]. This is especially true for NQS, which typically feature QGTs with a spectrum spanning many orders of magnitude [60], often making QGT-based algorithms challenging to stabilize [15, 21, 23].

To counter that, there is empirical evidence that in some situations, increasing the number of samples used to estimate the QGT and gradients helps to stabilize the solution [23]. Furthermore, it is possible to apply various regularization techniques to the equation. A standard option is to add a small diagonal shift ϵ to the QGT matrix before inverting it, thus solving the linear equation

$$\sum_j (G_{i,j} + \epsilon)\delta'_j = f_i \,. \tag{25}$$

When ϵ is small, the solution δ' will be close to the desired solution. Otherwise it is biased towards the plain force f, which is still acceptable in gradient-based optimization. To add this diagonal shift in NetKet, one of the following approaches can be used:

```
>>> qgt_1 = vs.quantum_geometric_tensor(QGTOnTheFly(diag_shift=0.001))
QGTOnTheFly(diag_shift=0.001)
>>> qgt_2 = qgt_1.replace(diag_shift=0.005)
QGTOnTheFly(diag_shift=0.005)
>>> qgt_3 = qgt_2 + 0.005
QGTOnTheFly(diag_shift=0.01)
```

Regularizing the QGT with a diagonal shift is an effective technique that can be used when performing SR/natural gradient descent for ground state search (see Section 4.1). Note, however, that since the diagonal shift biases the solution of the linear equation towards the plain gradient, it may bias the evolution of the system away from the physical trajectory in cases such as real-time evolution. In those cases, non-iterative solvers such as those based on SVD can be used, the stability of which can be controlled by suppressing smaller singular values. It has also been suggested in the literature to improve stability by suppressing particularly noisy gradient components [21, 39]. This is not currently implemented in NetKet, but planned for a future release. SVD-based regularization also comes at the cost of potentially suppressing physically relevant dynamics [23], making it necessary to find the right balance between stabilization and physical accuracy, and increased computational time as SVD is usually less efficient than iterative solvers.

4 Algorithms for variational states

The main use case of NetKet is variational optimization of wave function ansätze. In the current version NetKet, three algorithms are provided out of the box via high-level driver classes: variational Monte Carlo (VMC) for finding ground states of (Hermitian) Hamiltonians, time-dependent variational Monte Carlo (t-VMC) for real- and imaginary-time evolution, and steady-state search of Liouvillian open-system dynamics.

These drivers are part of the `nk.driver` module but we also export them from the `nk` namespace. They are constructed from the relevant physical model (e.g., a Hamiltonian), a variational state, and other objects used by the optimization method. They all support the `run` method, which performs a number of optimization steps and logs their progress (e.g., variational energies and network parameters) in a variety of output formats.

We highlight that these drivers are built on top of the functionalities described in Sections 2 and 3, and users are free to implement their own drivers or optimization loops, as demonstrated in Section 4.4.

4.1 Ground-state search

NetKet provides the variational driver `nk.VMC` for searching for minimal-energy states using VMC [52]. In the simplest case, the `VMC` constructor takes three arguments: the Hamiltonian, an optimizer and the variational state (see Section 3.1). NetKet makes use of optimizers provided by the JAX-based OPTAX library [65],[17] which can be directly passed to `VMC`, allowing the user to build complex training schedules or custom optimizers. In each optimization step, new samples of the variational state are drawn and used to estimate the gradient of the Hamiltonian with respect to the parameters θ of the ansatz [52] based on the force vector [compare Eq. (14)]

$$\tilde{f}_i = \mathrm{Cov}[O_i, \tilde{H}] = \mathbb{E}[O_i^*(\tilde{H} - \mathbb{E}[\tilde{H}])], \tag{26}$$

where $\tilde{H}$ is the local estimator (12) of the Hamiltonian, known as the *local energy*, and O_i is the log-derivative (13) of the wave function. All expectation values in Eq. (26) are evaluated over the Born distribution $\propto |\psi(\,\cdot\,)|^2$ and can therefore be estimated by averaging over the Monte Carlo samples. Given the vector $\tilde{f}$, the direction of steepest descent is given by the energy gradient

$$f \equiv \nabla_\theta \langle \hat{H} \rangle = 2\,\mathrm{Re}[\tilde{f}\,] \quad \text{(real)}\,, \tag{27}$$

or complex co-gradient [43]

$$f \equiv \nabla_{\theta^*} \langle \hat{H} \rangle = \tilde{f} \quad \text{(complex holomorphic)}\,. \tag{28}$$

Here we have distinguished the case of i) real parameters and ii) complex parameters with a variational mapping that is holomorphic with respect to θ. For non-holomorphic ansätze (cf. Section 3.2.2), complex parameters can be treated pairs of separate real-valued parameters (real and imaginary part) in the sense of eq. (27). Therefore, this case can be considered equivalent to the real parameter case.

The gradients are then passed on to the OPTAX optimizer, which may transform them (using, e.g., Adam) further before updating the parameters θ. Using the simple stochastic gradient descent optimizer `optax.sgd` (alias `nk.optimizer.Sgd`), the update rule is

$$\theta_i \mapsto \theta_i - \eta f_i\,, \tag{29}$$

[17]The `nk.optimizer` submodule includes a few optimizers for ease of use and backward compatibility: these are simply re-exports from OPTAX.

where the gradient f is given by Eq. (27) or (28) as appropriate.

Below we give a short snippet showing how to use the VMC `driver` to find the ground-state of the Ising Hamiltonian.

```python
# Define the geometry of the lattice
g = nk.graph.Hypercube(length=10, n_dim=1, pbc=False)
# Hilbert space of spins on the graph
hi = nk.hilbert.Spin(s=1 / 2, N=g.n_nodes)
# Construct the Hamiltonian
hamiltonian = nk.operator.Ising(hi, graph=g, h=0.5)

# define a variational state with a Metropolis Sampler
sa = nk.sampler.MetropolisLocal(hi)
vstate = nk.vqs.MCState(sa, nk.models.RBM())

# Construct the VMC driver
vmc = nk.VMC(hamiltonian,
             nk.optimizer.Sgd(learning_rate=0.1),
             variational_state=vstate)

# run the optimisation for 300 steps
output = vmc.run(300)
```

To improve on plain stochastic gradient descent, the `VMC` interface allows passing a keyword argument `preconditioner` . This must be a function that maps a variational state and the gradient vector f_i to the vector δ_i to be passed to the optimizer as gradients instead of f_i. An important use case is *stochastic reconfiguration* [52], where the gradient is preconditioned by solving the linear system of equations

$$\sum_j \mathrm{Re}[G_{ij}]\delta_j = f_i = 2\,\mathrm{Re}[\tilde{f}_i] \quad (\text{real})\,, \tag{30}$$

or

$$\sum_j G_{ij}\delta_j = f_i \quad (\text{complex holomorphic})\,. \tag{31}$$

The corresponding preconditioner can be created from a QGT class and a JAX-compatible linear solver (the default is `jax.scipy.sparse.linalg.cg`) using `nk.optimizer.SR` :

```python
# Construct the SR object with the chosen algorithm
sr = nk.optimizer.SR(
qgt = nk.optimizer.qgt.QGTOnTheFly,
    solver=jax.scipy.sparse.linalg.bicgstab,
    diag_shift=0.01,
)

# Construct the VMC driver
vmc = nk.VMC(
    hamiltonian,                          # The Hamiltonian to optimize
    nk.optimizer.sgd(learning_rate=0.1),  # The optimizer
    variational_state=vstate,             # The variational state
    preconditioner=sr,                    # The preconditioner
)
```

4.2 Finding steady states

In order to study open quantum systems, NETKET provides the `nk.SteadyState` variational driver for determining the variational steady-state $\hat{\rho}_{ss}$ defined as the stationary point of an arbitrary super-operator $\mathcal{L}$,

$$0 = \frac{d\hat{\rho}}{dt} = \mathcal{L}\hat{\rho}\,. \tag{32}$$

The search is performed by minimizing the Frobenius norm of the time-derivative [24], which defines the cost function

$$\mathcal{C}(\theta) = \frac{\|\mathcal{L}\hat{\rho}\|_2^2}{\|\hat{\rho}\|_2^2} = \frac{\mathrm{Tr}\big[\hat{\rho}^\dagger\mathcal{L}^\dagger\mathcal{L}\hat{\rho}\big]}{\mathrm{Tr}\big[\hat{\rho}^\dagger\hat{\rho}\big]}\,, \tag{33}$$

which has a global minimum for the steady state. The stochastic gradient is estimated over the probability distribution of the entries of the vectorized density matrix according to the formula:

$$f_i \equiv \frac{\partial}{\partial\theta_i^*}\frac{\|\mathcal{L}\hat{\rho}\|_2^2}{\|\hat{\rho}\|_2^2} = \mathbb{E}\big[\tilde{\mathcal{L}}\nabla_i^*\tilde{\mathcal{L}}\big] - \mathbb{E}[O_i^*\tilde{\mathcal{L}}^2]\,, \tag{34}$$

where $\tilde{\mathcal{L}}(s,s') = \sum_{m,m'}\mathcal{L}(s,s';m,m')\rho(m,m')/\rho(s,s')$ is the local estimator proposed in [24], and the expectation values are taken with respect to the “Born distribution” of the vectorized density matrix, $p(s,s') \propto |\rho(s,s')|^2$. The optimization works like the ground-state optimization provided by `nk.VMC` : the gradient is passed to an OPTAX optimizer, which may transform it further before updating the parameters θ. The simplest optimizer, `optax.sgd` , would update the parameters according to the equation

$$\theta_i \to \theta_i - \eta f_i\,. \tag{35}$$

To improve the performance of the optimization, it is possible to pass the keyword argument `preconditioner` to specify a gradient preconditioner, such as *stochastic reconfiguration* that uses the quantum geometric tensor to transform the gradient. The geometric tensor is computed according to the L_2 norm of the vectorized density matrix (see Section 3.5).

As an example, we provide a snippet to study the steady state of a transverse-field Ising chain with 10 spins and spin relaxation corresponding to the Lindblad master equation

$$\mathcal{L}\hat{\rho} = -i\big[\hat{H},\hat{\rho}\big] + \sum_i \hat{\sigma}_i^-\hat{\rho}\hat{\sigma}_i^+ - \frac{1}{2}\big\{\hat{\sigma}_i^+\hat{\sigma}_i^-,\hat{\rho}\big\}\,. \tag{36}$$

We first define the Hamiltonian and a list of jump operators, which are stored in a `LocalLiouvillian` object, which is a lazy representation of the super-operator $\mathcal{L}$. Next, a variational mixed state is built by defining a sampler over the doubled Hilbert space and optionally a different sampler for the diagonal distribution $p(s) \propto \rho(s,s)$, which is used to estimate expectation values of operators. The number of samples used to estimate super-operators and operators can be specified separately, as shown in the example by specifying `n_samples` and `n_samples_diag` .

```python
# Define the geometry of the lattice
g = nk.graph.Hypercube(length=10, n_dim=1, pbc=False)
# Hilbert space of spins on the graph
hi = nk.hilbert.Spin(s=1 / 2, N=g.n_nodes)

# Construct the Liouvillian Master Equation
ha = nk.operator.Ising(hi, graph=g, h=0.5)
j_ops = [nk.operator.spin.sigmam(hi, i) for i in range(g.n_nodes)]
```

```python
# Create the Liouvillian with Hamiltonian and jump operators
lind = nk.operator.LocalLiouvillian(ha, j_ops)

# Observable
sz = sum([nk.operator.spin.sigmam(hi, i) for i in range(g.n_nodes)])

# Neural Density Matrix
sa = nk.sampler.MetropolisLocal(lind.hilbert)
vs = nk.vqs.MCMixedState(
    sa, nk.models.NDM(beta=1), n_samples=2000, n_samples_diag=500
)
# Optimizer
op = nk.optimizer.Sgd(0.01)
sr = nk.optimizer.SR(diag_shift=0.01)

ss = nk.SteadyState(lind, op, variational_state=vs, preconditioner=sr)
out = ss.run(n_iter=300, obs={"Sz": sz})
```

4.3 Time propagation

Time propagation of variational states can be performed by incorporating the time dependence in the variational parameters and deriving an equation of motion that gives a trajectory in parameters space $\theta(t)$ that approximates the desired quantum dynamics. For real-time dynamics of pure and mixed NQS, such an equation of motion can be derived from the time-dependent variational principle (TDVP) [64, 66, 67]. When combined with VMC sampling to estimate the equation of motion (EOM), this is known as time-dependent variational Monte Carlo (t-VMC) [52,68] and is the primary approach currently used in NQS literature [8,18,19,21–23]. For complex holomorphic parametrizations[18], the TDVP equation of motion is

$$\sum_j G_{ij}(\theta)\,\dot{\theta}_j = \gamma f_i(\theta, t), \tag{37}$$

with the QGT G and force vector f defined in Sections 4.1 and 4.2. After solving Eq. (37), the resulting parameter derivative $\dot{\theta}$ can be passed to an ODE solver. The factor γ determines the type of evolution:

- For $\gamma = -i$, the EOM approximates the real-time Schrödinger equation on the variational manifold, the simulation of which is the main use case for the t-VMC implementation provided by NetKet.
- For $\gamma = -1$, the EOM approximates the imaginary-time Schrödinger equation on the variational manifold. When solved using the first-order Euler scheme $\theta(t+dt) = \theta(t)+\dot{\theta}\,dt$, this EOM is equivalent to stochastic reconfiguration with learning rate dt. Imaginary-time propagation with higher-order ODE solvers can therefore also be used for ground state search as an alternative to VMC. This can result in improved convergence in some cases [17].
- For $\gamma = 1$ and with the Lindbladian super-operator taking the place of the Hamiltonian in the definition of the force f, this ansatz yields the dissipative real-time evolution according to the Gorini-Kossakowski-Lindblad-Sudarshan master equation [69]. Our implementation uses the QGT induced by the vector norm [25] as discussed in the last paragraph of Section 3.5.

[18]The TDVP can be implemented for real-parameter wavefunctions by taking real parts of the right-hand side and QGT similar to VMC (Section 4.1) [39,46]. This is not yet available in the current version of NetKet, but will be added in a future release.

The current version of NetKet provides a set of Runge–Kutta (RK) solvers based on JAX and a driver class `TDVP` implementing the t-VMC algorithm for the three use cases listed above. At the time of writing, these features are provided as a preview version in the `netket.experimental` namespace as their API is still subject to ongoing development. The ODE solvers are located in the submodule `netket.experimental.dynamics`, the driver under `netket.experimental.TDVP`.

Runge-Kutta solvers implement the propagation scheme [70]

$$\theta(t+dt)=\theta(t)+dt\sum\nolimits_{l=1}^{L}b_l k_l\,, \tag{38}$$

using a linear combination of slopes

$$k_l = F\left(\theta(t)+\sum\nolimits_{m=1}^{l-1}a_{lm}k_m,\ t+c_l\,dt\right), \tag{39}$$

each determined by the solution $F(\theta,t)=\dot{\theta}$ of the equation of motion (37) at an intermediate time step. The coefficients $\{a_{lm}\}$, $\{b_l\}$, and $\{c_l\}$ determine the specific RK scheme and its order. NetKet further supports step size control when using adaptive RK schemes. In this case, the step size is dynamically adjusted based on an embedded error estimate that can be computed with little overhead during the RK step (38) [70]. Step size control requires a norm on the parameters space in order to estimate the magnitude of the error. Typically, the Euclidean norm $\|\delta\|=\sqrt{\sum_i|\delta_i|^2}$ is used. However, since different directions in parameters space influence the quantum state to different degrees, it can be beneficial to use the norm $\|\delta\|_G=\sqrt{\sum_i \delta_i^* G_{ij}\delta_j}$ induced by the QGT G as suggested in Ref. [21], which takes this curvature into account and is also provided as an option with the NetKet time-evolution driver.

An example demonstrating the use of NetKet's time evolution functionality is provided in Sec. 9.

4.4 Implementing custom algorithms using NetKet

While key algorithms for energy optimization, steady states, and time propagation are provided out of the box in the current NetKet version, there are many more applications of NQS. While we wish to provide new high-level driver classes for additional use cases, such as quantum state tomography [31] or general overlap optimization [32], it is already possible and encouraged for users to implement their own algorithms on top of NetKet. For this reason, we provide the core building blocks of NQS algorithms in a composable fashion.

For example, it is possible to write a simple loop that solves the TDVP equation of motion (37) for a holomorphic variational ansatz and using a first-order Euler scheme [i.e., $\theta(t+dt)=\theta(t)+\dot{\theta}(t)dt$] very concisely, making use of the elementary building blocks provided by the `VariationalState` class:

```python
def custom_simple_tdvp(
    hamiltonian: AbstractOperator,  # Hamiltonian
    vstate: VariationalState,       # variational state
    t0: float,                      # initial time
    dt: float,                      # time step
    t_end: float,                   # end time
):
    t = t0
    while t < t_end:
        # compute the energy gradient f
        energy, f = vstate.expect_and_grad(hamiltonian)
        G = vstate.quantum_geometric_tensor()
        # multiply the gradient by -1.0j for unitary dynamics
        gamma_f = jax.tree_map(lambda x: -1.0j * x, f)
        # Solve the linear system using any solver, such as CG
        # (or write your own regularization scheme)
        dtheta, _ = G.solve(jax.scipy.sparse.linalg.cg, gamma_f)
        # update the parameters (theta = theta + dt * dtheta)
        vs.parameters = jax.tree_map(
            lambda x, y: x + dt * y, vs.parameters, dtheta
        )
        t = t + dt
```

While the included `TDVP` driver (Section 4.3) provides many additional features (such as error handling, step size control, or higher-order integrators) and makes use of JAX's just-in-time compilation, this simple implementation already provides basic functionality and shows how NetKet can be used for quick prototyping.

5 Symmetry-aware neural quantum states

NetKet includes a powerful set of utilities for implementing NQS ansätze that are symmetric or transform correctly under the action of certain discrete symmetry groups. Only groups that are isomorphic to a set of permutations of the computational basis are supported. This is useful for modeling symmetric (e.g., lattice) Hamiltonians, whose eigenstates transform under irreducible representations of their symmetry groups. Restricting the Hilbert space to individual symmetry sectors can improve the convergence of variational optimization [71] and the accuracy of its result [14,16,49,72]. Additionally, symmetry restrictions can be used to find excited states [13, 16, 30], provided they are the lowest energy level in a particular symmetry sector.

While there is a growing interest for other symmetry groups, such as continuous ones like $SU(2)$ or $SO(3)$, they cannot be compactly represented in the computational basis and therefore the approach described in this chapter cannot be used. Finding efficient encodings for continuous groups is still an open research problem and it's not yet clear which strategy will work best [16].

NetKet uses *group convolutional neural networks* (GCNNs) to build wave functions that are symmetric over a finite group G. GCNNs generalize convolutional neural networks, invariant under the Abelian translation group, to general symmetry groups G which may contain non-commuting elements [73]. GCNNs were originally designed to be invariant, but they can be modified to transform under an arbitrary irreducible representation (irrep) of G, using the projection operator [74]

$$|\psi_\chi\rangle = \frac{d_\chi}{|G|}\sum_{g\in G}\chi_g^*\, g|\psi\rangle\,, \tag{40}$$

where g runs over all symmetry operations in G, with corresponding characters χ_g. Under the trivial irrep, where all characters are unity, the invariant model is recovered.

NetKet can infer the full space group of a lattice, defined as a set of permutations of lattice sites, starting from a geometric description of its point group. It can also generate nontrivial irrep characters [to be used in (40) for states with nonzero wave vectors or transforming non-trivially under point-group symmetries] using a convenient interface that approximates standard crystallographic formalism [75]. In addition, NetKet provides powerful group-theoretic algorithms for arbitrary permutation groups of lattice sites, allowing new symmetry elements to be easily defined.

Pre-built GCNNs are then provided in the `nk.models` submodule, which can be constructed by specifying few parameters, such as the number of features in each layer, and the lattice or permutation group under which the network should transform. Symmetric RBMs [8] are also implemented as one-layer GCNNs that aggregate convolutional features using a product rather than a sum. These pre-built network architectures are made up of individual layers found in the `nk.nn` submodule, which can be used directly to build custom symmetric ansätze.

Section 5.1 describes the NetKet interface for constructing space groups of lattices and their irreps. Usage of GCNNs is described in Section 5.2, while appendix A provides mathematical and implementation details.

5.1 Symmetry groups and representation theory

NetKet supports symmetry groups that act on a discrete Hilbert space defined on a lattice. On such a Hilbert space, space-group symmetries act by permuting sites; most generally, therefore, arbitrary subgroups of the symmetric group S_N of a lattice of N sites are supported. A symmetry group can be specified directly as a list of permutations, as in the following example, which enforces the symmetry $\psi(s_0, s_1, s_2, s_3) = \psi(s_3, s_1, s_2, s_0)$ for all four-spin configurations $s = (s_0, \ldots, s_3)$, $s_i = \pm 1$:

```
hi = nk.hilbert.Spin(1/2, N=4)
symms = [
    [0, 1, 2, 3],  # identity element
    [3, 1, 2, 0],  # swap first and last site
]
model = nk.models.RBMSymm(symms, alpha=1)
```

The listed permutations are required to form a group and, in particular, the identity operation $e : s \mapsto s$ must always be included as the first element.

It is inconvenient and error-prone to specify all space-group symmetries of a large lattice by their indices. Therefore, NetKet provides support for abstract representations of permutation and point groups through the `nk.utils.group` module, complete with algorithms to compute irreducible representations [76–78]. The module also contains a library of two- and three-dimensional point groups, which can be turned into lattice-site permutation groups using the graph class `nk.graph.Lattice` (but not general `Graph` objects, for they carry no geometric information about the system):

```
from netket.utils.group.planar import D
from netket.graph import Lattice

# construct a centred rectangular lattice
lattice = Lattice(
    basis_vectors = [[2,0], [0,1]], # each row is a lattice vector
    extent = (5,5),
    site_offsets = [[0,0], [1,0.5]], # each row is the position of a
    site in the unit cell
    point_group = D(2) # the point group of the lattice, here Z_2 x Z_2
)
```

NetKet contains specialized constructors for some lattices (e.g., `Square` or `Pyrochlore`), which come with a default point group; however, these can be overridden in methods like `Lattice.space_group`:

```
from netket.utils.group.planar import rotation, reflection_group, D
```

```python
from netket.utils.group import PointGroup, Identity
from netket.graph import Honeycomb

# construct the D_6 point group of the honeycomb lattice by hand
cyclic_6 = PointGroup(
    [Identity()] + [rotation(360 / n * i) for i in range(1, n)],
    ndim=2,
)
# the @ operator returns the Cartesian product of groups
# but doesn't check for group structure
dihedral_6 = reflection_group(angle=0) @ cyclic_6

assert dihedral_6 == D(6)

lattice = Honeycomb([6,6])

# returns the full space group of ‘lattice‘ as a PermutationGroup
space_group = lattice.space_group()
# the space group is spanned by 6^2 translations and 12 point-group
    symmetries
assert len(space_group) == 12 * 6 * 6

# do this if the Hamiltonian breaks reflection symmetry
# can also be used for generic Lattices that have no default point group
space_group = lattice.space_group(cyclic_6)
```

Irreducible representation (irrep) matrices can be computed for any point or permutation group object using the method `irrep_matrices()`. Characters (the traces of these matrices) are returned by the method `character_table()` as a matrix, each row of which lists the characters of all group elements. Character tables closer to the format familiar from quantum chemistry texts are produced by `character_table_readable()`. Irrep matrices and character tables are calculated using adaptations of Dixon's [76] and Burnside's [77] algorithms, respectively.

It would, however, be impractical to inspect irreps of a large space group directly to specify the symmetry sector on which to project a GCNN wave function. Exploiting the semidirect-product structure of space groups [78], space-group irreps are usually[19] described in terms of a set of symmetry-related wave vectors (known as a *star*) and an irrep of the subgroup of the point group that leaves the same invariant (known as the *little group*) [75]. Irreps can be constructed in this paradigm using `SpaceGroupBuilder` objects returned by `Lattice.space_group_builder()`:

```python
from netket.graph import Triangular

lattice = Triangular([6,6])
momentum = [0,0]
# space_group_builder() takes an optional PointGroup argument
sgb = lattice.space_group_builder()

# choosing a representation
# this one corresponds to the B_2 irrep at the Gamma point
chi = sgb.space_group_irreps(momentum)[3]
```

[19]Representation theory for wave vectors on the surface of the Brillouin zone in a nonsymmorphic space group is much more complicated [78] and is not currently implemented in NetKet.

The irrep `chi`, generated using the little group, is equivalent to one of the irreps in `Lattice.space_group().character_table()` and can thus be used for symmetry-projecting GCNN ansätze. The order in which irreps of the little group are returned can readily be checked in an interactive session:

```
>>> sgb.little_group(momentum).character_table_readable()
(['1xId()', '2xRot(60)', '2xRot(120)', '1xRot(180)', '3xRefl(0)',
    '3xRefl(-30)'],
array([[ 1.,  1.,  1.,  1.,  1.,  1.],     # this is irrep A1
       [ 1.,  1.,  1.,  1., -1., -1.],     # A2
       [ 1., -1.,  1., -1.,  1., -1.],     # B1
       [ 1., -1.,  1., -1., -1.,  1.],     # B2
       [ 2.,  1., -1., -2.,  0.,  0.],     # E1
       [ 2., -1., -1.,  2.,  0.,  0.]]))   # E2
```

The main caveat in using this machinery is that the point groups predefined in NetKet all leave the origin invariant (except for `cubic.Fd3m` which represents the "nonsymmorphic point group" of the diamond/pyrochlore lattice) and thus only work well with lattices in which the origin has full point-group symmetry. This behaviour can be changed (see the definition of `cubic.Fd3m` for an example), but it is generally safer to define lattices using the proper Wyckoff positions [75], of which the origin is usually maximally symmetric.

5.2 Using group convolutional neural networks (GCNNs)

NetKet uses GCNNs [49, 73] to create NQS ansätze that are symmetric under space groups of lattices. These networks consist of alternating *group convolutional layers* and pointwise nonlinearities. The former can be thought of as a generalization of convolutional layers to a generic finite group G. They are *equivariant,* that is, if their inputs are transformed by some space-group symmetry, features in all subsequent layers are transformed accordingly. As a result, the output of a GCNN can be understood as amplitudes of the wave functions $g|\psi\rangle$ for all $g \in G$, which can be combined into a symmetric wave function using the projection operator (40). Further details about equivariance and group convolutions are given in Appendix A.1.

GCNNs are constructed by the function `nk.models.GCNN`. Symmetries are specified either as a `PermutationGroup` or a `Lattice`. In the latter case, the symmetry group is given by `space_group()`; an optional `point_group` argument to `GCNN` can be used to override the default point group. By default, output transforms under the trivial irrep $\chi_g \equiv 1$, that is, all output features are averaged together to obtain a wave function that is fully symmetric under the whole space group. Other irreps can be specified through the `characters` argument, which takes a vector of the same size as the space group.

```
from netket.graph import Triangular
from netket.models import GCNN
from netket.utils.group.planar import C

lattice = Triangular([6,6])
momentum = [0,0]
sgb = lattice.space_group_builder()
chi = sgb.space_group_irreps(momentum)[3]

# This transforms as the trivial irrep Gamma A_1
gcnn1 = GCNN(lattice, layers = 4, features=4)

# This transforms as Gamma B_2
```

Table 4: Comparison of GCNN implementations. $f_{\text{in,out}}$ stands for the number of input and output features, $|G|, |P|, |T|$ for the sizes of the space group, point group, and translation group, respectively. d_a are the dimensions of irreps of G; in a large space group, $\sum_a d_a^3$ scales as $|G||P|$.

	`mode="irreps"`	`mode="fft"`
Can be used for	any group	only space groups
Symmetries can be specified by	• `Lattice` • `PermutationGroup` • Symmetry permutations and irrep matrices	• `Lattice` • `PermutationGroup` and shape of translation group • Symmetry permutations, product table, and shape of translation group
Kernel memory footprint per layer	$O(f_{\text{in}} f_{\text{out}} \|G\|)$	$O(f_{\text{in}} f_{\text{out}} \|G\|\|P\|)$
Evaluation time per layer per sample	$O[(f_{\text{in}} + f_{\text{out}})\|G\|^2 + f_{\text{in}} f_{\text{out}} \sum_a d_a^3]$	$O[(f_{\text{in}} + f_{\text{out}})\|G\| \log \|T\| + f_{\text{in}} f_{\text{out}} \|G\|\|P\|]$
Preferable for	• large point groups • if expanded `"fft"` kernels don't fit in memory	• small point groups • very large batches (see App. A.2)

```python
gcnn2 = GCNN(lattice.space_group(), characters=chi, layers=4,
    features=4)

# This does not enforce reflection symmetry
gcnn3 = GCNN(lattice, point_group=C(6), layers=4, features=4)
```

NetKet currently supports two implementations of GCNNs, one based on group Fourier transforms (`mode="irreps"`), the other using fast Fourier transforms on each coset of the translation group (`mode="fft"`): these are discussed in more detail in Appendix A.2. Their behavior is equivalent, but their performance and calling sequence is different, as explained in Table 4. A default `mode="auto"` is also available. For spin models, parity symmetry (taking σ^z to $-\sigma^z$) is a useful extension of the U(1) spin symmetry group enforced by fixing magnetization along the σ^z axis. Parity-enforcing GCNNs can be constructed using the `parity` argument, which can be set to ± 1.

In addition to deep GCNNs, fully symmetric RBMs [8] are implemented in `nk.models.RBMSymm` as a single-layer GCNN from which the wave function is computed as

$$\psi = \prod_{i,g\in G} \cosh f_g^{(i)} \implies \log\psi = \sum_{i,g\in G} \ln\cosh f_g^{(i)}. \tag{41}$$

Due to the products (rather than sums) used, this ansatz only supports wave functions that transform under the trivial irrep. An RBM-like structure closer to that of ref. [72] can be achieved using a single-layer GCNN:

```python
from netket.models import GCNN, RBMSymm
from netket.nn import logcosh

# fully symmetric RBM
```

```
rbm1 = RBMSymm(group, alpha=4)

# symmetrized RBM similar to (Nomura, 2021)
rbm2 = GCNN(group, layers=1, features=4, output_activation=logcosh)
```

6 Quantum systems with continuous degrees of freedom

In this section we will introduce the tools provided by NETKET to study systems with continuous degrees of freedom. The interface is very similar to the one introduced in Section 2 for discrete degrees of freedom.

6.1 Continuous Hilbert spaces

Similar to the discrete Hilbert spaces, the bosonic Hilbert space of N particles in continuous space has the structure

$$\mathcal{H}_{\text{continuous}} = \text{span}\{|x_0\rangle \otimes \cdots \otimes |x_{N-1}\rangle : x_i \in \mathcal{L}_i, i \in \{0,\ldots,N-1\}\}, \tag{42}$$

where $\mathcal{L}_i$ is the space available to each individual boson: for example, $\mathcal{L}_i$ is $\mathbb{R}^d$ for a free particle in d spatial dimensions, and $[0,L]^d$ for particles confined to a d-dimensional box of side length L. In the case of finite simulation cells, the boundaries can be equipped with periodic boundary conditions.

In the following snippet, we define the Hilbert space of five bosons in two spatial dimensions, confined to $[0,10]^2$ with periodic boundary conditions:

```
>>> hilb = nk.hilbert.Particle(N=5, L=(10.0, 10.0), pbc=True)
>>> print("Size of the basis: ", hilb.size)
Size of the basis:  10
>>> hilb.random_state(nk.jax.PRNGKey(0), (2,))
[[0.02952452 0.21660899 2.836163   3.5628846  4.5622005  5.9473248
   6.104126   8.14864    9.163713   9.263418  ]
  [9.85617    0.4667458  2.211265   4.1587596  4.250165   6.69916
   6.5165453  7.3764215  8.508119   0.08060825]]
```

As we discussed in Section 2.1, the Hilbert objects only define the computational basis. For that reason, the flag `pbc=True` only affects what configurations can be generated by samplers and how to compute the distance between two different sets of positions. This option does not enforce any boundary condition for the wave-function, which would have to be accounted for into the variational ansatz.

6.2 Linear operators

Similar to the discrete-variable case, expectation values of operators can be estimated as classical averages of the local estimator

$$\tilde{O}(x) = \frac{\langle x|\hat{O}|\psi\rangle}{\langle x|\psi\rangle} \tag{43}$$

over the (continuous) Born distribution $p(x) \propto |\psi(x)|^2$. NETKET provides the base class `ContinuousOperator` to write custom (local) operators and readily implements Hamiltonians of the form ($\hbar = 1$ in our units)

$$\hat{H} = -\frac{1}{2}\sum_i \frac{1}{m_i}\nabla_i^2 + \hat{V}\big(\{\mathbf{x}_i\}\big), \tag{44}$$

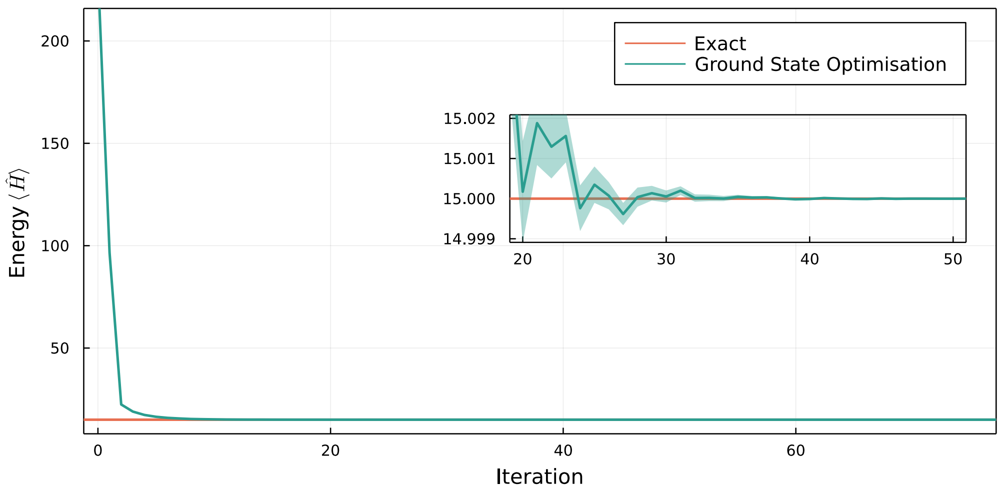


Figure 1: VMC energy estimate as a function of the optimization step for a continuous-space system of $N = 10$ particles in $d = 3$ spatial dimensions subject to a harmonic confinement.

using the predefined operators `KineticEnergy` and `PotentialEnergy`. For example, a harmonically confined system described by $\hat{H} = -\frac{1}{2}\sum_i \nabla_i^2 + \frac{1}{2}\sum_i \hat{\mathbf{x}}_i^2$ can be implemented as

```python
# This function takes a single vector and returns a scalar
def v(x):
    return 0.5*jnp.linalg.norm(x) ** 2

# Construct the Kinetic energy term with unit mass
H_kin = nk.operator.KineticEnergy(hilb, mass=1.0)
# Construct the Potential energy term using the potential defined above
H_pot = nk.operator.PotentialEnergy(hilb, v)

# Sum the two objects into a single Operator
H = H_kin + H_pot
```

Operators defined on continuous Hilbert spaces cannot be converted to a matrix form or used in exact diagonalization, in contrast to those defined on discrete Hilbert spaces. Continuous operators can still be used to compute expectation values and their gradients with a variational state.

6.3 Samplers

Out of the built-in samplers in the current version of NetKet (Section 3.4), only the Markov chain Monte Carlo sampler `MetropolisSampler` supports continuous degrees of freedom, as both `ExactSampler` and the autoregressive `ARNNSampler` rely on the sampled basis being countable. For continuous spaces, we provide the transition rule `sampler.rules.GaussianRule` which proposes new states by adding a random shift to every degree of freedom sampled from a Gaussian of customizable width. More complex transition rules can be defined following the instructions provided in Section 3.4.

6.4 Harmonic oscillators

As a complete example of how to use continuous-space Hilbert spaces, operators, variational states, and the VMC driver together, consider 10 particles in three-dimensional space, confined by a harmonic potential $V(x) = x^2/2$. The exact ground-state energy of this system is known to be $E_0 = 15$. We use the multivariate Gaussian ansatz $\log \psi(x) = -x^T \Sigma^{-1} x$, where $\Sigma = T T^\intercal$ and T is randomly initialized using a Gaussian with zero mean and variance one. Note that the form of Σ ensures that it is positive definite.

```python
import netket as nk
import jax.numpy as jnp

def v(x):
    return 0.5*jnp.linalg.norm(x) ** 2

hilb = nk.hilbert.Particle(N=10, L=(jnp.inf,jnp.inf,jnp.inf), pbc=False)
ekin = nk.operator.KineticEnergy(hilb, mass=1.0)
pot = nk.operator.PotentialEnergy(hilb, v)
ha = ekin + pot

sa = nk.sampler.MetropolisGaussian(hilb, sigma=0.1, n_chains=16,
    n_sweeps=32)
model = nk.models.Gaussian(param_dtype=float)
vs = nk.vqs.MCState(sa, model, n_samples=10 ** 4, n_discard=2000)

op = nk.optimizer.Sgd(0.05)
sr = nk.optimizer.SR(diag_shift=0.01)

gs = nk.VMC(ha, op, sa, variational_state=vs, preconditioner=sr)
gs.run(n_iter=100, out="HO_10_3d")
```

We show the training curve of above snippet in Fig. 1; exact ground-state energy is recovered to a very high accuracy.

6.5 Interacting system with continuous degrees of freedom

In this example we want to tackle an interacting system of bosonic Helium particles in one continuous spatial dimension. The two-body interaction is given by the *Aziz* potential which qualitatively resembles a Lennard-Jones potential [79–81]. The Hamiltonian reads

$$H = -\frac{\hbar^2}{2m} \sum_i \nabla_i^2 + \sum_{i<j} V_{\text{Aziz}}(r_{ij})\,. \tag{45}$$

We will examine the system at a density $\rho = \frac{N}{L} = 0.3 \mathring{A}^{-1}$ with $N = 10$ particles in units where $\hbar = m = k_b = 1$. To confine the system it is placed in a box of size L equipped with periodic boundary conditions. The Hilbert space and sampler are initialized as shown above (r_m is the length-scaled defined in the Aziz potential):

```python
import netket as nk

N = 10
d = 0.3 # 1/Angstrom
rm = 2.9673 # Angstrom
L = N / (0.3 * rm)
hilb = nk.hilbert.Particle(N=N, L=L, pbc=True)
```

```python
sab = nk.sampler.MetropolisGaussian(hilb, sigma=0.05, n_chains=16,
    n_sweeps=32)
```

6.5.1 Defining the Hamiltonian

We can define the Hamiltonian through the action of the interaction-potential on a sample of positions x, and combine it with the predefined kinetic energy operator. Since we are using periodic boundary conditions, we will use the Minimum Image Convention (MIC) to compute distances between particles. In the following snippet the Aziz potential (in the units above) is defined and the Hamiltonian is instantiated:

```python

def minimum_distance(x, sdim):
    """Computes distances between particles using mimimum image
   convention"""
    n_particles = x.shape[0] // sdim
    x = x.reshape(-1, sdim)

    distances = (-x[jnp.newaxis, :, :] + x[:, jnp.newaxis, :])[
        jnp.triu_indices(n_particles, 1)
    ]
    distances = jnp.remainder(distances + L / 2.0, L) - L / 2.0

    return jnp.linalg.norm(distances, axis=1)

def potential(x, sdim):
    """Compute Aziz potential for single sample x"""
    eps = 7.846373
    A = 0.544850 * 10 ** 6
    alpha = 13.353384
    c6 = 1.37332412
    c8 = 0.4253785
    c10 = 0.178100
    D = 1.241314

    dis = minimum_distance(x, sdim)
    return jnp.sum(
        eps
        * (
            A * jnp.exp(-alpha * dis)
            - (c6 / dis ** 6 + c8 / dis ** 8 + c10 / dis ** 10)
            * jnp.where(dis < D, jnp.exp(-((D / dis - 1) ** 2)), 1.0)
        )
    )

ekin = nk.operator.KineticEnergy(hilb, mass=1.0)
pot = nk.operator.PotentialEnergy(hilb, lambda x: potential(x, 1))
ha = ekin + pot
```

6.5.2 Defining and training the variational Ansatz

There are two properties that the variational Ansatz for this system must obey:

1. It must be invariant with respect to the permutations of its particles, because they are bosons;

2. As the interaction resembles a Lennard-Jones potential we have a strong divergence in the potential energy when particles get close to each other. This divergence must be compensated by the kinetic energy.

We satisfy the permutation-invariance by using a neural network architecture called *DeepSets*. *DeepSets* exploit the fact that any function $f(x_1,...,x_N)$ which is invariant under permutations of its inputs can be decomposed as [82]:

$$f(x_1,...,x_N) = \rho\left(\sum_i \phi(x_i)\right), \tag{46}$$

where ρ and ϕ are arbitrary functions. In this specific example, x_i denotes a single-particle position and ρ and ϕ are parameterized with dense feed forward neural networks.

The second requirement is fulfilled by using *Kato's cusp condition* which states that [83]

$$\lim_{r\to 0}\left(\frac{\nabla^2\psi_c(r)}{\psi_c(r)} + V(r)\right) < \infty, \tag{47}$$

where r denotes the distance between the particles and ψ_c is the cusp wave-function. For the case of a Lennard-Jones potential ($\propto r^{-12}$-divergence), we have

$$\psi_c(r) = \exp\left[-\frac{1}{2}\left(\frac{b}{r}\right)^5\right], \tag{48}$$

where b is a variational parameter.

We also need to handle the periodic conditions, making sure that the wave-function does not exhibit divergent behaviour at the edges of the (periodic) box. To this end we will use a surrogate distance function for the minimum image convention, namely $d_{\sin}(x_i,x_j) = \frac{L}{2}\sin\left(\frac{\pi}{L}r_{ij}\right)$ in the variational Ansatz. Additionally we replace the single-particle coordinates in Eq. (46) by the two-particle distances $d_{\sin}(x_i,x_j)^2$, such that all in all our variational Ansatz reads

$$\psi(x_1,...,x_N) = \exp\left[\rho\left(\sum_{i<j}\phi(d_{\sin}(x_i,x_j)^2)\right)\right]\cdot\exp\left[-\frac{1}{2}\left(\frac{b}{d_{\sin}(x_i,x_j)}\right)^5\right]. \tag{49}$$

The variational ansatz we described here is implemented in NetKet as `DeepSeetRelDistance`, and a more in-depth discussion can be found in this reference [84].

Having defined the ansatz, we run the VMC driver with the given variatianal Ansatz to find an estimation of the ground-state energy of the system. This is done as follows:

```python
model = nk.models.DeepSetRelDistance(
    hilbert=hilb,
    cusp_exponent=5,
    layers_phi=2,
    layers_rho=3,
    features_phi=(16, 16),
    features_rho=(16, 16, 1),
)
vs = nk.vqs.MCState(sab, model, n_samples=4096, n_discard_per_chain=128)

op = nk.optimizer.Sgd(0.001)
sr = nk.optimizer.SR(diag_shift=0.01)

gs = nk.VMC(ha, op, sab, variational_state=vs, preconditioner=sr)
gs.run(n_iter=1000, out="Helium_10_1d")
```

The result of this optimization and a comparison to literature results is displayed in Fig. 2.

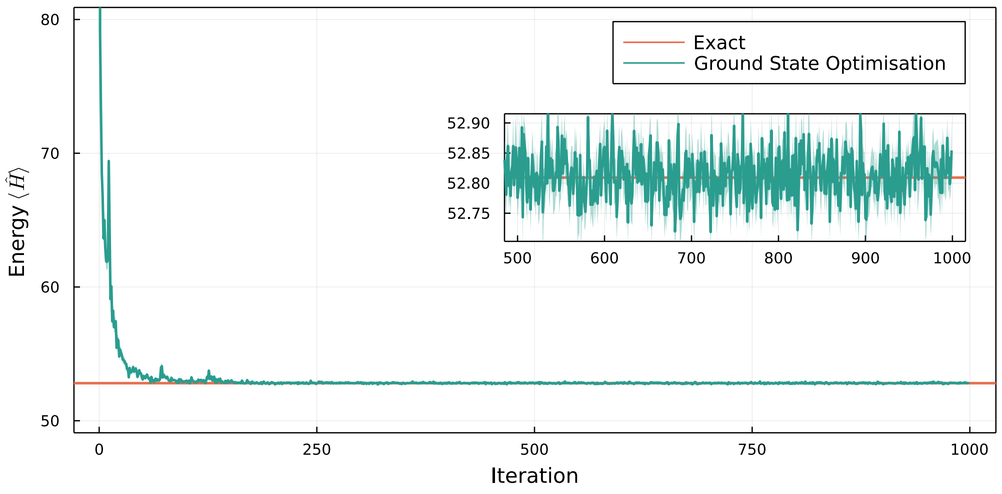


Figure 2: VMC energy estimate as a function of the optimization step for a continuous-space system of $N = 10$ particles in $d = 1$ spatial dimensions subject to a LJ-like interaction potential placed within a periodic box. The green dashed line is the result given in the supplementary material of [80].

7 Example: Finding ground and excited states of a lattice model

In this example, we define the J_1–J_2 Heisenberg Hamiltonian

$$H = J_1 \sum_{\langle ij \rangle} \vec{\sigma}_i \cdot \vec{\sigma}_j + J_2 \sum_{\langle\langle ij \rangle\rangle} \vec{\sigma}_i \cdot \vec{\sigma}_j \,, \tag{50}$$

on a 10×10 square lattice and use the VMC code introduced in Section 4.1 to find a variational approximation of its ground state. This model gives rise to several phases of matter, including magnetically ordered states, a valence bond solid, and a quantum spin liquid. Here, we set $J_1 = 1, J_2 = 0.5$, inside the spin liquid phase [30, 85].

Our example is optimized to run on a single GPU with 16 GB of memory. We will make note of what should be changed when running the simulation on CPUs.

7.1 Defining the lattice and the Hamiltonian

We use the `Lattice` class to define the square lattice and generate its space-group symmetries. By passing `max_neighbor_order=2` to the constructor, we generate graph edges for both nearest and next-nearest neighbours. The pre-defined `Heisenberg` class supports passing different coupling constants for both types of edge.

```python
import netket as nk
import numpy as np
import json
from math import pi

L = 10
# Build square lattice with nearest and next-nearest neighbor edges
lattice = nk.graph.Square(L, max_neighbor_order=2)
hi = nk.hilbert.Spin(s=1 / 2, total_sz=0, N=lattice.n_nodes)
# Heisenberg with coupling J=1.0 for nearest neighbors
# and J=0.5 for next-nearest neighbors
```

```python
H = nk.operator.Heisenberg(hilbert=hi, graph=lattice, J=[1.0, 0.5])
```

7.2 Defining and training a symmetric ansatz

To enforce all spatial symmetries of (50), we use the GCNN ansatz described in Section 5. By default, the GCNN projects onto the symmetric representation, which contains the ground state for this geometry. We select singlet states by only sampling basis states with $\sum_i S_i^z = 0$ and setting spin parity using `parity=1`. We use a model with four layers, each containing four feature vectors (i.e., four hidden units for each of the $8L^2$ space-group symmetries). To exploit the high degree of parallelism of GPUs, we set `sampler.n_chains` equal to `vstate.n_samples`[20]. When using CPUs, `n_chains` should be set to a smaller value.

```python
machine = GCNN(
    symmetries=lattice,
    parity=1,
    layers=4,
    features=4,
    param_dtype=np.complex128,
)
sampler = nk.sampler.MetropolisExchange(
    hilbert=hi,
    n_chains=1024,
    graph=lattice,
    d_max=2,
)
opt = nk.optimizer.Sgd(learning_rate=0.02)
sr = nk.optimizer.SR(diag_shift=0.01)
vstate = nk.vqs.MCState(
    sampler=sampler,
    model=machine,
    n_samples=1024,
    n_discard_per_chain=0,
    chunk_size=4096,
)
gs = nk.VMC(H, opt, variational_state=vstate, preconditioner=sr)
gs.run(n_iter=200, out="ground_state")

data = json.load(open("ground_state.log"))
print(np.mean(data["Energy"]["Mean"]["real"][-20:])/400)
#  Output: -0.49562531096409457
print(np.std(data["Energy"]["Mean"]["real"][-20:])/400)
#  Output: 0.0002492304032505182
```

We specify `chunk_size=4096` in the variational state in order to reduce memory consumption. As we have $L^2 = 100$ sites, at every VMC step we will need to evaluate the network for $\mathcal{O}(N_{\text{samples}} L^2) = \mathcal{O}(10^3 \cdot 10^2)$ different configurations, but the memory available on commercial GPUs will not be enough to perform this computation in a single pass. Instead, by

[20] Note that this results in a somewhat non-standard MC scheme where, instead of an ensemble of chains with generally non-zero internal autocorrelation, the sampler produces an ensemble of independently drawn single configurations (`n_samples_per_chain==1`). Since the sampler state of the previous VMC step is used as initial state for the next step, there can still be a residual autocorrelation with the previous samples, which is, however, alleviated by the sampler performing N_{sites} intermediate updates before yielding the single requested sample with the settings used here.

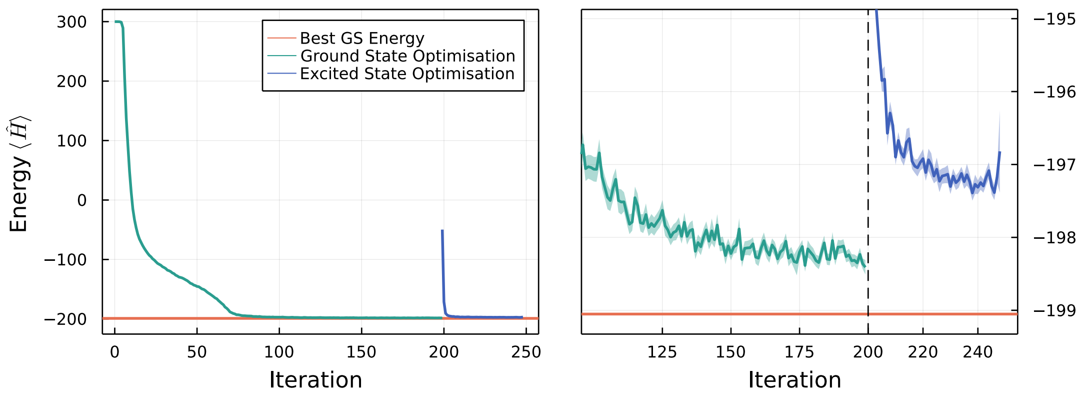


Figure 3: Energy evolution of variational ground states (green) and excited states after transfer learning (blue), compared to the lowest known variational energy for the 10×10 square-lattice J_1–J_2 model [30] (black). The variational energies can be further improved by allowing more training steps, Monte Carlo samples, etc. The plot on the right zooms in on the lowest energies.

setting `chunk_size` NetKet will split the calculation in many smaller sub-calculations (see Section 3.3.1 for more details).

This calculation, which takes about 30 minutes on an NVIDIA A100 GPU, already delivers a fairly accurate variational energy. The evolution of variational energy during the training procedure is shown in Fig. 3.

We note that a typical initialization of a GCNN gives rise to ferromagnetic correlations, which can make training an antiferromagnetic Hamiltonian unstable [15, 49]. Therefore, it is often good practice to pre-optimize the phases by restricting all amplitudes to unity by setting `equal_amplitudes=True` switch and training only the phases of the network. These parameters can then be loaded into a model with `equal_amplitudes=False`.

7.3 Finding an excited state

We can also find low-lying excited states using this procedure, by projecting the wavefunction onto a different irrep. Here, we consider the first gapless mode in the Anderson tower of states of the Néel antiferromagnet [86], a triplet at wave vector (π, π) that transforms trivially under all point-group symmetries. This mode is still gapless in the quantum spin liquid; we project out spin-singlets by focusing on parity odd states.

We expedite this calculation by using parameters optimized for the ground-state sector as an initial guess. The resulting wave function will already have a low variational energy (as shown in Fig. 3) and correlations typical for low-energy eigenstates.

```python
# store the optimized ground-state parameters
saved_params = vstate.parameters
# Compute the characters of the first excited state
characters = lattice.space_group_builder().space_group_irreps(pi, pi)[0]
# Construct a model respecting the first-excited state symmetries
machine = GCNN(
    symmetries=lattice,
    characters=characters,
    parity=-1,
    layers=4,
    features=4,
    param_dtype=complex,
)
vstate = nk.vqs.MCState(
    sampler=sampler,
    model=machine,
    n_samples=1024,
    n_discard_per_chain=0,
    chunk_size=4096,
)
# assign the old parameters to the new variational state
vstate.parameters = saved_params
gs = nk.VMC(H, opt, variational_state=vstate, preconditioner=sr)
gs.run(n_iter=50, out='excited_state')

data = json.load(open("excited_state.log"))
print(np.mean(data["Energy"]["Mean"]["real"][-10:])/400)
#  Output: -0.49301426054097885
print(np.std(data["Energy"]["Mean"]["real"][-10:])/400)
#  Output: 0.0003802670752071611
```

8 Example: Fermions on a lattice

NetKet can also be used to simulate fermionic systems with a finite number of orbitals. Functionality related to fermions is kept in the `netket.experimental` in order to signal that some parts of the API might still slightly change while we gather feedback from the community. We usually import this namespace as `nkx` as follows:

```python
from netket import experimental as nkx
```

and then use `nkx` freely.

The Hilbert space for discrete fermionic systems is called `SpinOrbitalFermions`. It supports fermions, with and without a spin- degree of freedom, which occupy a set of orbitals (such as the sites of a lattice). Internally, it uses a tensor product of a `Fock` space for each spin component. For a set of spin-1/2 fermions, we can fix the number of fermions with up (↑) and down (↓) spins through the `n_fermions` keyword argument. The `SpinOrbitalFermions` generates samples that correspond to occupation numbers $|n\rangle = \left|n_{1,\uparrow}, \dots, n_{N_o,\uparrow}, n_{1,\downarrow}, \dots, n_{N_o,\downarrow}\right\rangle$, for a given ordering of the N_o orbitals (or sites).

In the example below, we determine the ground state of the Fermi-Hubbard model on a square lattice

$$\hat{H} = -t \sum_{\langle i,j\rangle} \sum_{\sigma=\{\uparrow,\downarrow\}} f^\dagger_{i,\sigma} f_{j,\sigma} + h.c. + U \sum_i n_{i,\uparrow} n_{i,\downarrow} \,, \tag{51}$$

where $n_{i,\sigma} = f^\dagger_{i,\sigma} f_{i,\sigma}$.

NetKet implements a class `FermionOperator2nd` that represents an operator in second quantization. This class does not separate spin and orbital indices. Internally, the `FermionOperator2nd` computes matrix elements of a fermion operator $f^\dagger_i$ on an orbital i through the Jordan-Wigner transformation

$$f^\dagger_i \to \left(\bigotimes_{j<i} Z_j \right) \left(\frac{X_i + iY_i}{2} \right) , \tag{52}$$

or in terms of occupation numbers

$$\langle n | f_i^\dagger | n' \rangle = (-1)^{\sum_{j<i} n_j} \delta_{n'_i+1, n_i} \prod_{j \neq i} \delta_{n_j, n'_j} \,. \tag{53}$$

One can create a `FermionOperator2nd` object from an external `FermionOperator` object from the OPENFERMION library, which is popular for symbolic manipulation of fermionic operators [87]. The small intermezzo code below shows how this works in practice for an operator $f_1^\dagger f_2 + 4 f_3 f_0^\dagger$

```python
from openfermion import FermionOperator
from netket.operator import FermionOperator2nd

of_operator = FermionOperator("1^ 2") + 4*FermionOperator("3 0^")
nk_operator1 = FermionOperator2nd.from_openfermion(of_operator)

nk_operator2 = FermionOperator2nd("1^ 2") + 4*FermionOperator2nd("3 0^")
```

where `nk_operator1` and `nk_operator2` are equivalent.

The mapping between fermions and qubit degrees of freedom is not unique [88], and the Jordan-Wigner transformation is one well-know example of such a transformation. However, by interfacing with toolboxes specialized in symbolic manipulation, we open up a range of possibilities, especially in combination with `PauliStrings.from_openfermion`, which converts `openfermion.QubitOperator` from OPENFERMION to a `PauliStrings` object in NETKET. This allows one to, for example, use a wider range of alternatives to the Jordan-Wigner transformation implemented in OPENFERMION, or other variations.

Going back to our example of the Fermi-Hubbard model, NETKET also implements a more easy to use set of creation and annihilation operators that clearly separate the orbitals and spin indices

- $f_{i,\sigma}^\dagger$: `nkx.operator.fermion.create`
- $f_{i,\sigma}$: `nkx.operator.fermion.destroy`
- $n_{i,\sigma}$: `nkx.operator.fermion.number`

Each operator takes a site and spin projection (`sz`) in order to find the right position in the Hilbert space samples. We will create a helper function to abbreviate the creation, annihilation and number operators in the example below.

```python
from netket import experimental as nkx
from netket.experimental.operator.fermion import (
    create as c, destroy as cdag, number as nc)

# create the graph our fermions can hop on
L = 4
g = nk.graph.Hypercube(length=L, n_dim=2, pbc=True)
n_sites = g.n_nodes

# create a hilbert space with 2 up and 2 down spins
hi = nkx.hilbert.SpinOrbitalFermions(n_sites, s=1 / 2, n_fermions=(2,
    2))

t = 1      # tunneling/hopping
```

```python
U = 0.01  # coulomb

up, down = +0.5, -0.5
ham = 0.0
for sz in (up, down):
    for u, v in g.edges():
        ham += -t * (cdag(hi, u, sz) * c(hi, v, sz) +
                     cdag(hi, v, sz) * c(hi, u, sz))
for u in g.nodes():
    ham += U * nc(hi, u, up) * nc(hi, u, down)
```

Sampling: To run a VMC optimization, we need a proper sampling algorithm that takes into account the constraints of the computational basis we are working with. As the `SpinOrbitalFermions` basis consereves total spin-magnetization, we cannot use samplers like `MetropolisLocal` which randomly change the population $n_{i,\sigma}$ on a site, thus changing the total spin. We can instead use `MetropolisExchange` , which moves fermions around according to the physical lattice graph of $L \times L$ vertices, but the computational basis defined by the Hilbert space contains $(2s+1)L^2$ occupation numbers. By taking a disjoint copy of the lattice, we can move the fermions around independently for both spins and therefore conserve the number of fermions with up and down spin. Notice that in the chosen representation, where is no need to anti-symmetrize our ansatz.

```python
sa = nk.sampler.MetropolisExchange(hi,
    graph=nk.graph.disjoint_union(g, g),
    n_chains=16)

ma = nk.models.RBM(alpha=1, param_dtype=complex)
vs = nk.vqs.MCState(sa, ma, n_discard_per_chain=100, n_samples=512)

opt = nk.optimizer.Sgd(learning_rate=0.01)
sr = nk.optimizer.SR(diag_shift=0.1)

gs = nk.driver.VMC(ham, opt, variational_state=vs, preconditioner=sr)
gs.run(500, out='fermions')
```

9 Example: Real-time dynamics

We demonstrate the simulation of NQS dynamics in the transverse-field Ising model (TFIM) on an L site chain with periodic boundaries, using a restricted Boltzmann machine (RBM) as the NQS ansatz. The Hamiltonian reads

$$\hat{H}_{\text{Ising}} = J \sum_{\langle ij \rangle} \hat{\sigma}^z_i \hat{\sigma}^z_j - h \sum_i \hat{\sigma}^x_i \,, \tag{54}$$

with $J = 1$ and $h = 1$ and periodic boundary conditions. We will estimate the expectation value of the transverse magnetization $\hat{S}^x = \sum_i \hat{\sigma}^x_i$ along the way.

We simulate the dynamics starting from an initial state $|\psi(t_0)\rangle$ that is the ground state for the TFIM Hamiltonian with $h = 1/2$. The random weight initialization of a neural network yields a random initial state. Therefore, we determine the weights corresponding to this initial state by performing a ground-state optimization. Even though the TFIM ground state can be parametrized using an RBM ansatz with real-valued weights, we need to use complex-valued

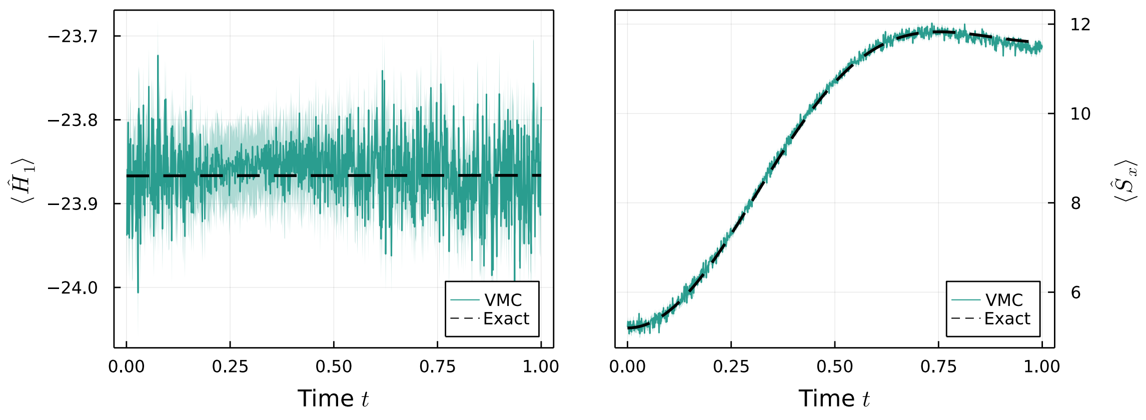


Figure 4: Comparison between the exact (dashed line) and variational dynamics of a quench on the transverse-field Ising model. **(Left):** Expectation value of the quenched Hamiltonian, which is conserved by the unitary dynamics. The shaded area represents the uncertainty due to Monte Carlo sampling. **(Right):** Expectation value of the total magnetization along the x axis during the evolution.

weights here, in order to describe the complex-phase of the wave function that arises during the time evolution[21]. In this example, we work with a chain of $L = 20$ sites, which can be easily simulated on a typical laptop with the parameters below.

```python
import netket as nk
import netket.experimental as nkx

# Spin Hilbert space on 20-site chain with PBC
L = 20
chain = nk.graph.Chain(L)
hi = nk.hilbert.Spin(s=1/2) ** L

# Define RBM ansatz and variational state
rbm = nk.models.RBM(alpha=1, param_dtype=complex)
sampler = nk.sampler.MetropolisLocal(hi)
vstate = nk.vqs.MCState(sampler, rbm, n_samples=4096)

# Hamiltonian at J=1 (default) and external field h=1/2
ha0 = nk.operator.Ising(hi, chain, h=0.5)
# Observable (transverse magnetization)
obs = {"sx": sum(nk.operator.spin.sigmax(hi, i) for i in range(L))}

# First, find the ground state of ha0 to use it as initial state
optimizer = nk.optimizer.Sgd(0.01)
sr = nk.optimizer.SR()
vmc = nk.VMC(ha0, optimizer, variational_state=vstate,
    preconditioner=sr)
# We run VMC with SR for 300 steps
vmc.run(300, out="ising1d_groundstate_log", obs=obs)
```

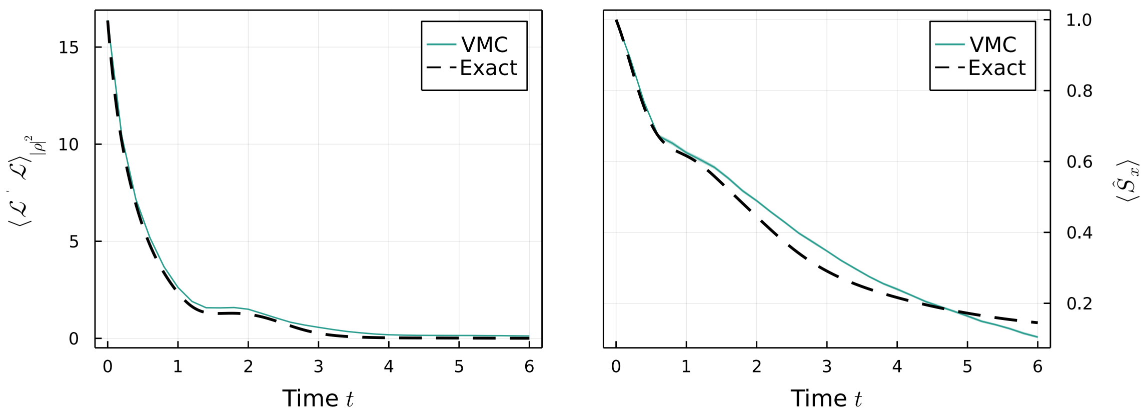


Figure 5: Comparison between the exact (dashed line) and variational dynamics of a random initial density matrix evolved according to the Lindblad Master equation. **(Left):** Expectation value of the $\langle\langle\rho|\mathcal{L}^\dagger\mathcal{L}|\rho\rangle\rangle$ convergence estimator. **(Right):** Expectation value of the total magnetization along the $\hat{x}$ axis during the evolution. We remark that the evolution is near-exact in the region where the dissipative terms dominate the dynamics, while there is a sizable error when the unitary dynamics starts to play a role. The error could be reduced by considering smaller time steps.

9.1 Unitary dynamics

Starting from the ground state, we can compute the dynamics after a quench of the transverse field strength to $h = 1$. We use the second-order Heun scheme for time stepping, with a step size of $dt = 0.01$, and explicitly specify a QGT implementation (compare Sec. 3.5) in order to make use of the more efficient code path for holomorphic models.

```python
# Quenched Hamiltonian
ha1 = nk.operator.Ising(hi, chain, h=1.0)
# Heun integrator configuration
integrator = nkx.dynamics.Heun(dt=0.001)
# QGT options
qgt = nk.optimizer.qgt.QGTJacobianDense(holomorphic=True)
# Creating the time-evolution driver
te = nkx.TDVP(ha1, vstate, integrator, qgt=qgt)
# Run the t-VMC solver until time T=1.0
te.run(T=1.0, out="ising1d_quench_log", obs=obs)
```

In Fig. 5 we show the results of this calculation, comparing against an exact solution computed using QuTiP [89, 90].

9.2 Dissipative dynamics (Lindblad master equation)

In Section 4.3 we have shown that the time-dependent variational principle can also be used to study the dissipative dynamics of an open quantum system. In this section we give a concrete example, studying the transverse-field Ising model coupled to a zero-temperature bath. The coupling is modeled through the spin depolarization operators $\hat{\sigma}_i^-$ acting on every site i.

As the dissipative dynamics converges to the steady state, which is also an attractor of the non-unitary dynamics, we will use a weak-simulation of the dynamics[22] to determine

[21]It is possible to first use a real-weight RBM for the initial state preparation and then switch to complex-valued weights for the dynamics. For the sake of simplicity, we leave out this extra step in the present example.

[22]We use weak and strong simulation in the sense of the theory of numerical SDE schemes [91]. This means

the steady-state more efficiently than using the natural gradient descent scheme proposed in ref. [24]. This scheme is similar to what was proposed in Ref. [25].

We employ the positive-definite RBM ansatz proposed in Ref. [51]. A version of that network with complex-valued parameters is provided in NetKet with the name `nk.models.NDM`.

```python
# The graph of the Hamiltonian
g = nk.graph.Chain(L, pbc=False)
# Hilbert space
hi = nk.hilbert.Spin(0.5)**g.n_nodes
# The Hamiltonian
ha = nk.operator.Ising(hi, graph=g, h=-1.3, J=0.5)
# Define the list of jump operators
j_ops = [nk.operator.spin.sigmam(hi, i) for i in range(g.n_nodes)]
# Construct the Liouvillian
lind = nk.operator.LocalLiouvillian(ha, j_ops)

# observable
Sx = sum([nk.operator.spin.sigmax(hi, i) for i in
    range(g.n_nodes)])/g.n_nodes

# Positive-definite RBM-like ansatz (Torlai et al.)
ma = nk.models.NDM(alpha=2, beta=3)
# MetropolisLocal sampling on the Choi's doubled space.
sa = nk.sampler.MetropolisLocal(lind.hilbert, n_chains=16)
# Mixed Variational State. Use less samples for the observables.
vs = nk.vqs.MCMixedState(
     sa, ma, n_samples=12000, n_samples_diag=1000,
    n_discard_per_chain=100
)
# Setup the ODE integrator and QGT.
integrator = nkx.dynamics.Heun(dt=0.01)
# The NDM ansatz is not holomorphic because it uses conjugation
qgt = nk.optimizer.qgt.QGTJacobianPyTree(holomorphic=False,
    diag_shift=1e-3)
te = nkx.TDVP(lind, variational_state=vs, integrator=integrator,
    qgt=qgt)

# run the simulation and compute observables
te.run(T=6.0, obs={"Sx": Sx, "LdagL": lind.H @ lind})
```

In the listing above, we first construct the Liouvillian by assembling the Hamiltonian and the jump operators, then we construct the variational mixed state. We chose a different number of samples for the diagonal, used when sampling the observables, as that happens on a smaller space with respect to the full system. For the geometric tensor, we choose the `QGTJacobianPyTree` and specify that the ansatz is non-holomorphic (while this is already the default, a warning would be printed otherwise, asking the user to be explicit). The choice is motivated by the fact that the TDVP driver by default uses an SVD-based solver, which works best `QGTJacobian` -based implementations. However, `NDM` uses a mix of complex and real parameters which is not supported by `QGTJacobianDense`, and would throw an error. Normally, to simulate a meaningful dynamics you'd want to keep the diagonal shift small, but since we are striving for a weak simulation a large value helps stabilize the dynamics.

that weak integration is an integration which is not accurate at finite times but which converges to the right state at long times. A strong integration yields the correct state at every time t.

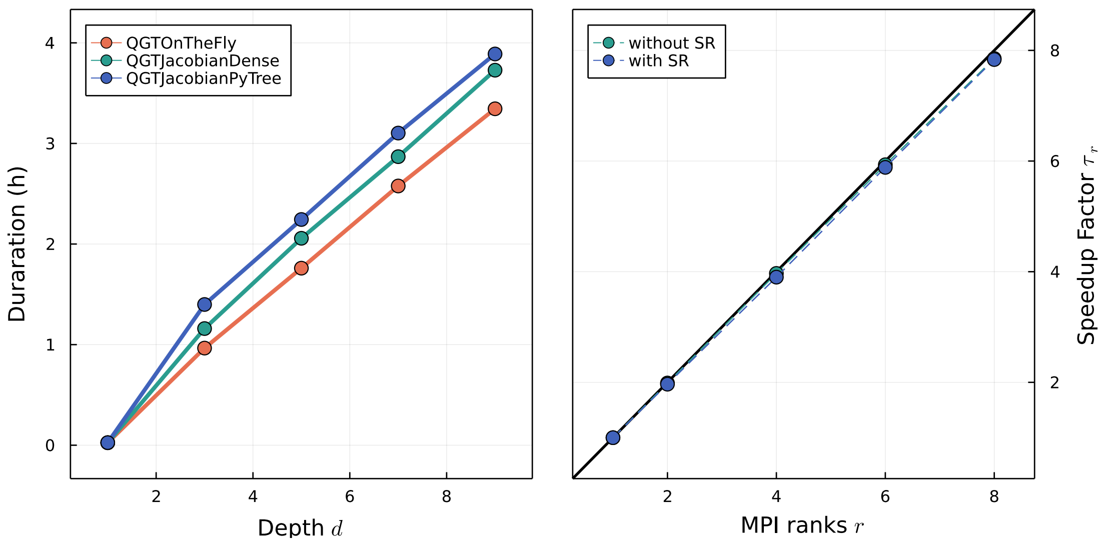


Figure 6: **(Left):** Benchmark of NETKET's VMC implementation. Each data point shows the minimum time spent (out of 5 repetitions) to evolve a DNN with depth d layers and complex weights over 100 VMC steps for the 1D transverse-field Ising model with $L = 256$ and $N_{\text{samples}} = 2^{14} = 16384$. **(Right):** Scaling behavior of the required computational time as a function of the number of MPI ranks on a single node. We repeat 5 VMC optimizations and report the minimum time required for 100 steps for the 1D Ising model with $L = 256$ and an RBM with $\alpha = 1$ with complex weights. The black line represents ideal scaling behavior.

10 Benchmarks

10.1 Variational Monte Carlo

We benchmark NETKET by measuring its performance on the 1D/2D transverse-field Ising model defined as in Eq. (54) with $J = 1$ and $h = 1$ and periodic boundary conditions.

We first monitor the scaling behavior of NETKET's VMC implementation by running an energy optimization consisting of 100 steps. In order to carry out a meaningful benchmark, we run a first VMC step to trigger the JIT-compiler on the relevant JAX and NUMBA functions, while all reported timings are for the evaluation of JIT compiled functions only. The left panel of Fig. 6 depicts the scaling behavior of the computational time as a function of the complexity of the NQS model, using three implementations of the QGT. Hereby, we increase the complexity of the NQS by optimizing a DNN with an increasing amount of layers, where depth d represents the number of dense layers (with $\alpha = 1$), each followed by a sigmoid activation function. Such a DNN with d layers has $\mathcal{O}((d-1)L^2 + L)$ free parameters.

10.2 MPI for NETKET

We benchmark the scaling behavior of NETKET as a function of the computational resources available to perform parallel computations. Therefore, NETKET uses MPI for JAX through MPI4JAX [37]. The effectiveness of the MPI implementation is illustrated in Fig. 7 for a VMC optimization with and without Stochastic Reconfiguration (SR). Throughout our analyses, we provide each MPI rank with 2 CPU cores. We introduce the speedup factor $\tau_r = \Delta t_1/\Delta t_r$ where Δt_r refers to the time required to perform the computations on r MPI ranks. Similarly, we define τ_n as the speedup factor when using n nodes. The right panel of Fig. 7 demonstrates that even on a single node, our MPI implementation can introduce significant speedups by run-

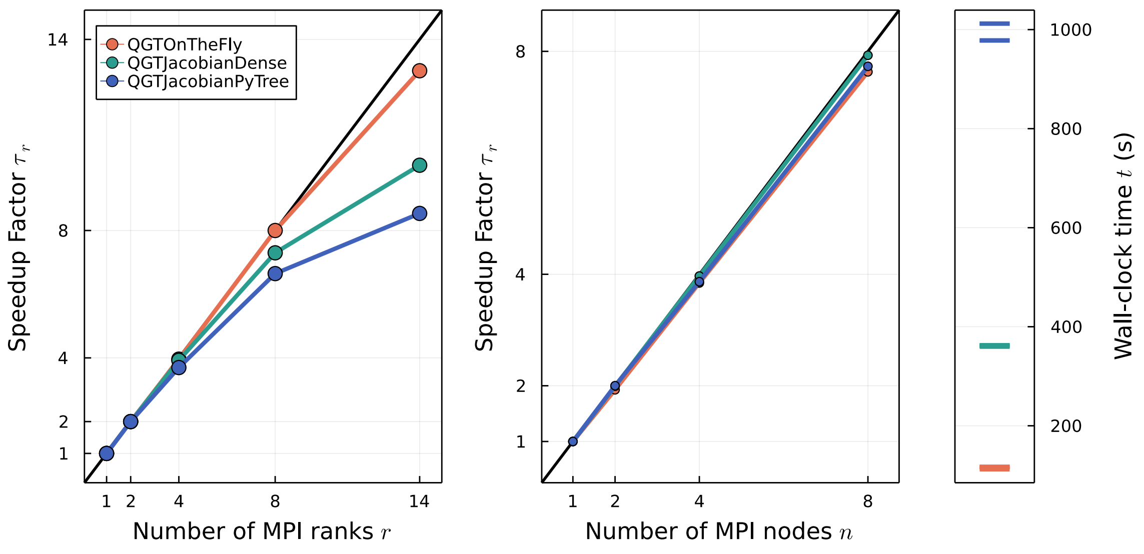


Figure 7: **(Left):** Speedup factor τ observed by increasing the number of MPI ranks r on a single node while keeping the problem size constant (strong scaling). **(Center):** Speedup factor observed by scaling across multiple nodes n, and scaling the number of samples accordingly (weak scaling). **(Right):** Scatter plot of the absolute wall clock time in seconds for the runs reported in the weak scaling (center) plot. There are 4 points for every color, but they overlap for most implementations because of the almost-ideal weak scaling. We repeat 5 iterations of constructing the QGT and 1000 matrix multiplications with the gradient vector for the 2D Ising model with $L = 8$ and a DNN with 9 layers and complex weights. In the left panel, we keep the number of samples $N_{\text{samples}} = 2^{14}$ constant, while in the center panel, we increase the number of samples to $2^{14} \times n$, while we correct the timing by the number of nodes n to show the speedup factor for a constant number of 2^{14} samples. We remark that while the speedup factor is resistant to changes in the network architecture, the absolute timing might favor one or another implementation depending on several details and can change depending on the architecture and problem at hand.

ning multiple Markov Chain samplers in parallel. This is consistent with the fact that JAX is not able to make use of multiple CPU cores unless working on very large matrices.

The performance of NETKET on challenging Hamiltonians, as well as its scalability with both system size and model complexity depends on the implementation details of the quantum geometric tensor [see Eq. (22)] and its matrix multiplication with the gradient vector, as discussed in Section 3.5. We therefore isolate these operations and benchmark the QGT constructor, combined with 1000 matrix multiplications with the gradient vector (where the latter imitates many steps in the iterative solver). In Fig. 7, we show the scaling behavior of these operations with respect to both the number of ranks (on a single node), and its scaling behavior with respect the number of nodes (thereby including communication over the Infiniband network). One can observe that the scaling behavior is close to optimal, especially when the number of samples per rank is sufficiently large.

10.3 Comparison with jVMC

We compare NETKET with jVMC [39], another open-source Python package supporting VMC optimization of NQS. Results are shown in Fig. 6. We remark that since both NETKET and jVMC are JAX-based, performance on sampling, expectation values and gradients is roughly

Table 5: Comparison of performance between NETKET and jVMC. All times are indicated in seconds and have been taken on a workstation with an `AMD Ryzen Threadripper 3970X 32-Core` processor and 2x`Nvidia RTX 3090` GPUs. Timings are for one VMC step using a complex-valued RBM model with hidden unit density of $\alpha = N_{\text{hidden}}/L = 1$ on the 1-dimensional TFIM model (54) with $L = 64$ sites. Other parameters are: 2^{14} total samples, 2^{10} independent Markov chains per GPU (or across all CPUs). Calculations for NETKET where performed using `QGTJacobianDense(holomorphic=True)`. NETKET multi-GPU calculations use CUDA-enabled MPI for inter-process communication, while jVMC uses JAX built-in mechanism. The run labeled with (MPI×16) is run on the same workstation but 16 MPI processes are used to better take advantage of the multiple cores of the processors.

		NETKET	jVMC
SVD solver	CPU (32 cores)	48	337
	GPU (×1)	24	44
	GPU (×2)	20	36
CG solver	CPU (32 cores)	86	N/A
	CPU (32 cores, MPIx16)	7.5	N/A
	GPU (×1)	3.9	N/A
	GPU (×2)	1.7	N/A

equivalent when using the same hyperparameters [39]. However, a performance difference arises in algorithms requiring the use of the QGT, such as TDVP or natural gradient (SR). Such difference will vanish in cases where the cost of solving the QGT linear system is small with respect to the cost of computing the energy and its gradient.

At the time of writing, jVMC only implements a singular-value decomposition (SVD) solver to invert the QGT matrix. The same type of solver can be used also in NETKET (for a detailed discussion, see Section 3.5). We limit the computations to models with less than 7000 parameters in order for the QGT to have less than $49 \cdot 10^3$ elements, which is approximately the maximum matrix size that can be diagonalized in reasonable time on the GPUs we have access to[23]. For that reason we chose the size of $L = 64$ spins.

As shown in Table 5, NETKET outperforms by almost an order of magnitude jVMC on a 32-core CPU using SVD-based solvers. On GPU, jVMC requires significantly less computational time than on CPU, yet, NETKET outperforms jVMC by about 50% in a full VMC iteration. We remark that in this benchmark both packages scale poorly when going from a single GPU to two. This is because the diagonalization of the QGT, in this case the bottleneck, cannot be parallelized.

In order to scale efficiently to many GPUs, our QGT implementations can be combined with iterative solvers to scale up to potentially millions of parameters, as well as significantly larger system sizes. As expected from Fig. 7, increasing the number of MPI ranks reduces the total time by a factor nearing the number of ranks (with eventually a saturation in the speedup). Notice also that the CG-solver becomes significantly more efficient on GPU.

[23]Using distributed linear-algebra libraries such as ScaLaPack [92], ELPA [93] or the recent [94] would allow us to avoid this barrier, however we are not aware of any Python binding for those libraries. Regardless, if those libraries exposed a distributed linear solver to Python, using it with NETKET would be as simple as using it as the linear solver for the QGT.

11 Discussion and conclusion

We have presented NETKET 3, a modular Python toolbox to study complex quantum-mechanical problems with machine learning-inspired tools. Compared to version 2 [35], the major new feature is the ability to define arbitrary neural-network ansätze for either wave functions or density matrices using the flexible JAX framework; we believe this makes NETKET much more useful to non-technical users. Another significant improvement is that NETKET is now completely modular. Users of NETKET 3 can decide to use only the neural-network architectures, the stochastic samplers, the quantum geometric tensor, or the operators without necessarily requiring the `VariationalState` interface, which is convenient but geared towards the most common applications of variational NQS. Care has been taken to ensure that the algorithms implemented can scale to very large systems and models with millions of parameters thanks to more efficient implementations of the geometric tensor and other algorithmic bottlenecks. Thanks to its JAX foundations, NETKET 3 can now also make effective use of GPU hardware, without any need for manual low-level programming for these platforms.

Even with all the new features that have been introduced with this version, there are many things that we would like to see integrated into NETKET in the future. To name a few: native support for fermionic systems; support for more general geometries in continuous systems; improvements to the dynamics submodule in order to support a wider variety of ODE solvers and more advanced regularization and diagnostic schemes [21, 23]; new drivers for quantum state reconstruction [31] and overlap optimization [32]; a more general way to define arbitrary cost functions to be optimized. However, we think that our new JAX-based core is very welcoming to contributors, and we believe that this constitutes a solid foundation upon which to build in the future. Moreover, we are now explicitly committed to stability of the user-facing API, in order to make sure that code written today will keep working for a reasonable time, while we iterate and refine NETKET.

Since a project is only as big as its community, the most important developments are probably those related to documentation, learning material, and developing a community where users answer each other's questions in the spirit of open, shared science. We are taking steps to make all of this happen.

Acknowledgments

We are grateful to the authors and maintainers of JAX [5], FLAX [44], and OPTAX [65] for guidance and constructive discussions on how to best use their tools. F. V. thanks W. Bruinsma for adapting plum-dispatch to our needs and making life more Julian in the Python world. We also thank all the users who engaged with us on GitHub reporting bugs, sharing ideas, and ultimately helping us to produce a better framework.

Funding information. This work is supported by the Swiss National Science Foundation under Grant No. 200021_200336. D. H. acknowledges support by the Max Planck-New York City Center for Nonequilibrium Quantum Phenomena. A. Sz. gratefully acknowledges the ISIS Neutron and Muon Source and the Oxford–ShanghaiTech collaboration for support of the Keeley–Rutherford fellowship at Wadham College, Oxford. J. N. was supported by Microsoft Research.

A Details of group convolutions

A.1 Group convolutions and equivariance

As explained in Section 5.2, GCNNs generate the action of every element g of the symmetry group G on a wave function $|\psi\rangle$, written as $|\psi_g\rangle = g|\psi\rangle$, which are then combined into a symmetric wave function using the projection operator (40). Amplitudes of the computational basis states $|\boldsymbol{\sigma}\rangle$ are related to one another in these wave functions as

$$\psi_g(\boldsymbol{\sigma}) = \langle\boldsymbol{\sigma}|g|\psi\rangle = \psi(g^{-1}\boldsymbol{\sigma})\,. \tag{55}$$

Therefore, feature maps inside the GCNN are indexed by group elements rather than lattice sites, and all layers must be *equivariant:* that is, if their input is transformed by a space-group symmetry, their output must be transformed the same way, so that (55) would always hold. Pointwise nonlinearities clearly fit this bill [73]; we now consider what linear transformations are allowed.

First, input feature maps naturally defined on lattice sites must be embedded into group-valued features:

$$\mathbf{f}_g = \sum_{\vec{r}} \mathbf{W}_{g^{-1}\vec{r}}\sigma_{\vec{r}} = \sum_{\vec{r}} \mathbf{W}_r \sigma_{g\vec{r}} = \sum_{\vec{r}} \mathbf{W}_{\vec{r}}(g^{-1}\boldsymbol{\sigma})_{\vec{r}}\,, \tag{56}$$

consistent with (55). We also see that the embedding (56) is equivariant: if the input is transformed by some symmetry operation u, the output transforms as

$$\sum_{\vec{r}} \mathbf{W}_{g^{-1}\vec{r}}\sigma_{u\vec{r}} = \sum_{\vec{r}} \mathbf{W}_{g^{-1}u^{-1}\vec{r}}\sigma_{\vec{r}} = \mathbf{f}_{ug}\,. \tag{57}$$

This layer is implemented in NetKet as `nk.nn.DenseSymm` .

To build deeper GCNNs, we also need to map group-valued features onto one another in an equivariant fashion. This is achieved by *group convolutional* layers, which transform input features as[24]

$$\boldsymbol{\phi}_g = \sum_{h\in G} \mathbf{W}_{h^{-1}g}\mathbf{f}_h\,. \tag{58}$$

This layer is implemented in NetKet as `nk.nn.DenseEquivariant` . It is equivariant under multiplying with a group element u from the left:

$$\sum_{h\in G} \mathbf{W}_{h^{-1}g}\mathbf{f}_{uh} = \sum_{h\in G} \mathbf{W}_{h^{-1}ug}\mathbf{f}_h = \boldsymbol{\phi}_{ug}\,, \tag{59}$$

which is consistent with how the embedding layer (56) is equivariant, cf. (57). Indeed, it can be composed with (56):

$$\boldsymbol{\phi}_g = \sum_h \mathbf{W}_{h^{-1}g}\sum_{\vec{r}} \mathbf{W}'_{h^{-1}\vec{r}}\sigma_{\vec{r}} = \sum_{\vec{r}}\left(\sum_h \mathbf{W}_{h^{-1}}\mathbf{W}'_{h^{-1}g^{-1}\vec{r}}\right)\sigma_{\vec{r}} \equiv \sum_{\vec{r}} \mathbf{W}''_{g^{-1}\vec{r}}\sigma_{\vec{r}}\,, \tag{60}$$

as the expression in brackets only depends on $g^{-1}\vec{r}$.

Finally, the output features of the last layer of the GCNN are turned into the wave functions $\psi_g(\boldsymbol{\sigma}) = \sum_i \exp\left(f_g^{(i)}\right)$ and projected on an irrep using (40) (we drop the irrelevant constant prefactor):

$$\psi(\boldsymbol{\sigma}) = \sum_{i,g} \chi_g^* \exp\left(f_g^{(i)}\right), \tag{61}$$

[24]Our convention differs from that of Ref. [73], which in fact implements group correlation rather than convolution. The two conventions are equivalent (the indexing of the kernels differs by taking the inverse of each element); we use convolutions to simplify the Fourier transform-based implementations of Sec. A.2.

where χ_g are the characters of the irrep. In addition to allowing nontrivial symmetries, our choice of summing a large number of terms in the ansatz appears to improve the stability of variational optimization for sign-problematic Hamiltonians [15, 30, 49, 72].

A.2 Fast group convolutions using Fourier transforms

The simplest implementation of a group convolutional layer is expanding each of the $f_\text{in}f_\text{out}$ kernels, containing $|G|$ entries, to a $|G|\times|G|$ matrix defined as

$$\tilde{W}^{(a,b)}_{g,h} = W^{(a,b)}_{g^{-1}h}\,, \tag{62}$$

where a,b index input and output features, respectively. The resulting tensor of size $f_\text{in}f_\text{out}|G|^2$ can then be contracted straightforwardly with the input features:

$$\phi^{(b)}_h = \sum_{a,g} \tilde{W}^{(a,b)}_{g,h} f^{(a)}_g \tag{63}$$

is equivalent to (58). Embedding layers (56) can be constructed analogously. This method is the easiest to interpret and code, serving as a useful check on other methods; however, the enlarged kernels require substantial amounts of memory, which already becomes a serious problem on modestly sized lattices and networks. Furthermore, evaluating a convolution using this method takes $O(f_\text{in}f_\text{out}|G|^2)$ time. NetKet implements two approaches to improve on this scaling.

The first approach uses *group Fourier transforms,* which generalize the usual discrete Fourier transform for arbitrary finite groups. The forward and backward transformations are defined by

$$\hat{f}(\rho) = \sum_{g\in G} f(g)\rho(g), \qquad\qquad f(g) = \frac{1}{|G|}\sum_\rho d_\rho \operatorname{Tr}\big[\hat{f}(\rho)\rho(g^{-1})\big]\,. \tag{64}$$

In the forward transformation, ρ is a representation of the group G; $f(g)$ is a function defined on group elements, while $\hat{f}(\rho)$ is a matrix of the same shape as the representatives $\rho(g)$. The sum in the backward transformation runs over all inequivalent irreps ρ, of dimension d_ρ, of the group. Since $\sum_\rho d_\rho^2 = |G|$, this transformation does not increase the amount of memory needed to store inputs, outputs, or kernels.[25] Group convolutions can readily be implemented by multiplying the Fourier transform matrices (we drop feature indices for brevity):

$$\hat{\phi}(\rho) = \sum_g \phi_g \rho(g) = \sum_{g,h} f_h W_{h^{-1}g}\rho(h)\rho(h^{-1}g) = \hat{f}(\rho)\hat{W}(\rho)\,. \tag{65}$$

To calculate a convolution using this approach, the input features are Fourier transformed [$O(f_\text{in}|G|^2)$ as there is no generic fast Fourier transform algorithm for group Fourier transforms], multiplied with the kernel Fourier transform for each irrep [$O(f_\text{in}f_\text{out}d_\rho^3)$ for an irrep of dimension d_ρ], and the output is transformed back [$O(f_\text{out}|G|^2)$], yielding the total runtime $O[(f_\text{in}+f_\text{out})|G|^2+f_\text{in}f_\text{out}\sum_\rho d_\rho^3]$. In a large space group, most irreps are defined on a star of $|P|$ wave vectors (P is the point group) and thus have dimension $|P|$; accordingly, $\sum_\rho d_\rho^3 \approx |G||P|$.

The second approach, based on Ref. [73], exploits the fact that the translation group T is a normal subgroup of the space group G, so each $g\in G$ can be written as $t_g p_g$, where t_g is a translation and p_g is a fixed coset representative (in symmorphic groups, we can choose these

[25] If some irreps cannot be expressed as matrices with real entries, the Fourier transform of real inputs/outputs/kernels is complex too, temporarily doubling the amount of memory used.

to be point-group symmetries). Now, we can define the expanded kernels (we drop feature indices again to reduce clutter)

$$\tilde{W}_t^{(p_g,p_h)} \equiv W_{p_g^{-1} t p_h}\,, \tag{66}$$

such that

$$\phi_h = \sum_g f_g W_{g^{-1}h} = \sum_{t_g,p_g} f_{t_g p_g} W_{p_g t_g^{-1} t_h p_h} \equiv \sum_{t_g,p_g} \tilde{f}_{t_g}^{(p_g)} \tilde{W}_{t_h - t_g}^{(p_g,p_h)}\,. \tag{67}$$

In the last form, we split the space-group feature map f into cosets of the translation group and observe that the latter is Abelian. In fact, the translation group is equivalent to the set of valid lattice vectors, so the sum over t_g in (67) is a standard convolution. Ref. [73] proposes to perform this convolution using standard cuDNN routines. However, we are usually interested in convolutions that span the entire lattice in periodic boundary conditions: these can be performed more efficiently using fast Fourier transforms (FFTs) as the Fourier transform of a convolution is the product of Fourier transforms. Therefore, we FFT the kernels $\tilde{W}$ and features $\tilde{f}$, contract them as appropriate, and FFT the result back:

$$\tilde{\phi}^{(b,p_h)} = \mathcal{F}^{-1}\left[\sum_{a,p_g} \mathcal{F}\left(\tilde{f}^{(a,p_g)}\right) \mathcal{F}\left(\tilde{W}^{(a,b;p_g,p_h)}\right)\right], \tag{68}$$

where the Fourier transform is understood to act on the omitted translation-group indices, and the Fourier transforms are multiplied pointwise.

Calculating a convolution in this approach involves $f_{\text{in}}|P|$ forward FFTs $[O(|T|\log|T|)$ each], $|T|$ tensor inner products $[O(f_{\text{in}} f_{\text{out}}|P|^2$ each], and $f_{\text{out}}|P|$ backward FFTs; as $|G| = |T||P|$, this yields a total of $O[(f_{\text{in}} + f_{\text{out}})|G|\log|T| + f_{\text{in}} f_{\text{out}}|G||P|]$. For large lattices, which bring out the better asymptotic scaling of FFTs, this improves significantly on the runtime of the group Fourier transform-based approach, especially in the pre- and postprocessing stages. By contrast, the group Fourier transform approach is better for large point groups, as it avoids constructing the $|P|^2$ reshaped kernels $\tilde{W}^{p_g p_h}$, which can be prohibitive for large lattices.

In practice, as both FFTs and group Fourier transforms involve steps more complicated than simple tensor multiplication, their performance is hard to assess beyond asymptotes, especially on a GPU. On CPUs, the FFT-based approach tends to be faster. On GPUs, computation time tends to scale sub-linearly with the number of operations so long as the process is efficiently parallelized. As all operations of the group Fourier transform implementation involve multiplications of large matrices, it can fully exploit the large GPU registers even with relatively few samples. By contrast, FFTs cannot be fully vectorized, meaning that larger batches are required to make full use of the computing power of the GPU. In practice therefore, the FFT-based approach may not perform better until most of the GPU memory becomes involved in evaluating a batch.

B Implementation details of the quantum geometric tensor

In the following, we discuss our implementations of the quantum geometric tensor, introduced in section 3.5, in more detail. In particular, we show how the action of the quantum geometric tensor on a vector can be computed efficiently without storing the full matrix. Appendix B.1 introduces relevant automatic-differentiation concepts in general terms; the concrete algorithms used by `QGTJacobian` and `QGTOnTheFly` are discussed in Appendices B.2 and B.3, respectively.

B.1 Jacobians and their products

We assume that our NQS is modeled by the scalar parametric function $f(s) = \ln \psi_\theta(s)$, where θ is a vector of variational parameters and s is a basis vector of the Hilbert space. Consistent with the notation of the main text, $O_j(s) = \partial_{\theta_j} \ln \psi_\theta(s)$ are the log-derivatives (13) of the NQS.

We also assume that f can be vectorized and evaluated for a batch of inputs $\{s_k\}_{k=1\dots N_s}$, yielding the vector $f_k = \ln \psi_\theta(s_k)$. The Jacobian of this function is therefore the matrix

$$J_{kl} = O_l(s_k) = \frac{\partial \ln \psi_\theta(s_k)}{\partial \theta_l}, \tag{69}$$

each row corresponds to the gradient of f evaluated at a different input s_k, so $k = 1 \dots N_s$ and $l = 1 \dots N_{\text{parameters}}$.

The Jacobian matrix can be computed in JAX with `jax.jacrev(log_wavefunction)(s)`, which returns a matrix.[26] However, it is often not needed to have access to the full Jacobian: for example, when computing the gradient (26) of the variational energy, we only need the product of the Jacobian with a vector, namely $\Delta E_{\text{loc}}(s_k) = \tilde{H}(s_k) - \mathbb{E}[\tilde{H}]$.

A vector can be contracted with the Jacobian along its dimension corresponding to either parameters or outputs:

- *Jacobian–vector products* (Jvp), $\tilde{\mathbf{v}} = J\mathbf{v}$, can be computed using forward-mode automatic differentiation;
- *vector–Jacobian products* (vJp), $\tilde{\mathbf{v}} = \mathbf{v}^T J$, can be computed through backward-mode automatic differentiation (backward propagation).

Modern automatic-differentiation frameworks like that of JAX implement primitives that evaluate Jvp and vJp, and construct higher-level functions such as `jax.grad` or `jax.jacrev` on top of those functions; that is, one can extract the best performance from JAX by making use of vJp and Jvp as much as possible [95].

B.2 QGTJacobian

Writing the estimator (23) of the quantum geometric tensor explicitly in terms a finite number of samples s_k, we obtain

$$\begin{aligned} G_{ij} &= \mathbb{E}\left[O_i^* O_j\right] - \mathbb{E}\left[O_i\right]^* \mathbb{E}\left[O_j\right] \\ &\approx \frac{1}{N_s}\sum_{k=1}^{N_s} O_i(s_k)^* O_j(s_k) - \frac{1}{{N_s}^2}\left(\sum_{k=1}^{N_s} O_i(s_k)\right)^*\left(\sum_{k=1}^{N_s} O_j(s_k)\right) \\ &= \frac{1}{N_s}\sum_{k=1}^{N_s}\left(O_i(s_k) - \sum_{k=1}^{N_s}\frac{O_i(s_k)}{N_s}\right)^*\left(O_j(s_k) - \sum_{k=1}^{N_s}\frac{O_j(s_k)}{N_s}\right) \\ &= \frac{1}{N_s}\sum_{k=1}^{N_s}\left(J_{ki} - \sum_{k=1}^{N_s}\frac{J_{ki}}{N_s}\right)^*\left(J_{kj} - \sum_{k=1}^{N_s}\frac{J_{kj}}{N_s}\right) \\ &= \frac{1}{N_s}\sum_{k=1}^{N_s}(\Delta J_{ki})^*(\Delta J_{kj}), \end{aligned} \tag{70}$$

[26]More precisely, it returns a PyTree with a structure similar to the PyTree that stores the parameters; each leaf gains an additional dimension of length N_s.

where we have defined the centered Jacobian $\Delta J_{ki} \equiv J_{ki} - \sum_{k=1}^{N_s} \frac{J_{ki}}{N_s}$. In matrix notation, this is equivalent to

$$G = \frac{\Delta J^\dagger}{\sqrt{N_s}} \frac{\Delta J}{\sqrt{N_s}} \,. \tag{71}$$

The Jacobian-based implementation of the quantum geometric tensor computes[27] and stores[28] the full Jacobian matrix J_{kl} for the given samples upon construction.Then, QGT–vector products $\tilde{\mathbf{v}} = G\mathbf{v}$ are computed without finding the full matrix G, in two steps:

$$\Delta \mathbf{w} = \frac{\Delta J}{\sqrt{N_s}} \mathbf{v} \,, \qquad \tilde{\mathbf{v}} = \frac{\Delta J^\dagger}{\sqrt{N_s}} \Delta \mathbf{w} = \left(\frac{\Delta J}{\sqrt{N_s}} \Delta \mathbf{w}^\dagger \right)^\dagger \,, \tag{72}$$

the final form has the advantage that the Hermitian transpose of a vector is simply its conjugate. Evaluating Eq. (72) is usually less computationally expensive than constructing the full quantum geometric tensor.

B.3 QGTOnTheFly

In some cases one might have so many parameters or samples that it is impossible to store the full Jacobian matrix in memory. In that case, we still evaluate a set of equations similar to Eq. (72), but without pre-computing the full Jacobian, only using vector–Jacobian and Jacobian–vector products.

It would be impractical to perform a vJp using the centered Jacobian; however, Eq. (23) can be rewritten as $G_{ij} = \mathbb{E}\left[O_i^* \left(O_j - \mathbb{E}\left[O_j\right]\right)\right]$, which yields

$$G = \frac{1}{N_s} J^\dagger \Delta J \,, \tag{73}$$

where we have substituted one of the two centered Jacobians with a plain Jacobian. Then, we note that the centered-Jacobian–vector product can be expressed as

$$\Delta \mathbf{w} \equiv \Delta J \, \mathbf{v} = \left(J - \frac{\mathbf{1}^T J}{N_s} \right) \mathbf{v} = \mathbf{w} - \frac{\mathbf{1}^T \mathbf{w}}{N_s} \,, \tag{74}$$

where $\mathbf{1}^T$ is a row vector all entries of which are 1, used to express averaging the columns of the Jacobian in the matrix formalism. Therefore, `QGTOnTheFly` performs the following calculations:

$$\mathbf{w} = \frac{1}{N_s} J \mathbf{v} \,, \qquad \Delta \mathbf{w} = \mathbf{w} - \langle \mathbf{w} \rangle \,, \qquad \tilde{\mathbf{v}} = J^\dagger \Delta \mathbf{w} \,, \tag{75}$$

where the first and the last step are implemented with `jax.vjp` and `jax.jvp`, respectively.

References

[1] A. Krizhevsky, I. Sutskever and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, Curran Associates Inc., Red Hook, ISBN 9781627480031 (2012).

[27] The full Jacobian can be computed row by row, performing vJps with vectors that have a single nonzero entry. In practice, as the network is evaluated independently for all samples in a batch, each of these products requires us to back-propagate the network for only one sample at a time.

[28] To speed up the evaluation of (72), we actually store $\Delta J / \sqrt{N_s}$.

[2] C. Chen, O. Li, D. Tao, A. Barnett, C. Rudin and J. K. Su, *This looks like that: Deep learning for interpretable image recognition*, Curran Associates Inc., Red Hook, ISBN 9781713807933 (2019).

[3] D. W. Otter, J. R. Medina and J. K. Kalita, *A survey of the usages of deep learning for natural language processing*, IEEE Trans. Neural Netw. Learn. Syst. **32**, 604 (2021), doi:10.1109/tnnls.2020.2979670.

[4] Y. Sun, N. B. Agostini, S. Dong and D. Kaeli, *Summarizing CPU and GPU design trends with product data*, arXiv:1911.11313.

[5] R. Frostig, M. J. Johnson and C. Leary, *Compiling machine learning programs via high-level tracing*, SysML (2018).

[6] J. Bezanson, A. Edelman, S. Karpinski and V. B. Shah, *Julia: A fresh approach to numerical computing*, SIAM Rev. **59**, 65 (2017), doi:10.1137/141000671.

[7] G. Carleo, I. Cirac, K. Cranmer, L. Daudet, M. Schuld, N. Tishby, L. Vogt-Maranto and L. Zdeborová, *Machine learning and the physical sciences*, Rev. Mod. Phys. **91**, 045002 (2019), doi:10.1103/revmodphys.91.045002.

[8] G. Carleo and M. Troyer, *Solving the quantum many-body problem with artificial neural networks*, Science **355**, 602 (2017), doi:10.1126/science.aag2302.

[9] D.-L. Deng, X. Li and S. Das Sarma, *Quantum entanglement in neural network states*, Phys. Rev. X **7**, 021021 (2017), doi:10.1103/PhysRevX.7.021021.

[10] D. Wu, L. Wang and P. Zhang, *Solving statistical mechanics using variational autoregressive networks*, Phys. Rev. Lett. **122**, 080602 (2019), doi:10.1103/PhysRevLett.122.080602.

[11] R. Kaubruegger, L. Pastori and J. C. Budich, *Chiral topological phases from artificial neural networks*, Phys. Rev. B **97**, 195136 (2018), doi:10.1103/physrevb.97.195136.

[12] I. Glasser, N. Pancotti, M. August, I. D. Rodriguez and J. I. Cirac, *Neural-network quantum states, string-bond states, and chiral topological states*, Phys. Rev. X **8**, 011006 (2018), doi:10.1103/PhysRevX.8.011006.

[13] K. Choo, G. Carleo, N. Regnault and T. Neupert, *Symmetries and many-body excitations with neural-network quantum states*, Phys. Rev. Lett. **121**, 167204 (2018), doi:10.1103/PhysRevLett.121.167204.

[14] T. Vieijra, C. Casert, J. Nys, W. De Neve, J. Haegeman, J. Ryckebusch and F. Verstraete, *Restricted Boltzmann machines for quantum states with non-Abelian or anyonic symmetries*, Phys. Rev. Lett. **124**, 097201 (2020), doi:10.1103/physrevlett.124.097201.

[15] A. Szabó and C. Castelnovo, *Neural network wave functions and the sign problem*, Phys. Rev. Res. **2**, 033075 (2020), doi:10.1103/PhysRevResearch.2.033075.

[16] T. Vieijra and J. Nys, *Many-body quantum states with exact conservation of non-Abelian and lattice symmetries through variational Monte Carlo*, Phys. Rev. B **104**, 045123 (2021), doi:10.1103/PhysRevB.104.045123.

[17] M. Bukov, M. Schmitt and M. Dupont, *Learning the ground state of a non-stoquastic quantum Hamiltonian in a rugged neural network landscape*, SciPost Phys. **10**, 147 (2021), doi:10.21468/SciPostPhys.10.6.147.

[18] S. Czischek, M. Gärttner and T. Gasenzer, *Quenches near Ising quantum criticality as a challenge for artificial neural networks*, Phys. Rev. B **98**, 024311 (2018), doi:10.1103/physrevb.98.024311.

[19] G. Fabiani and J. Mentink, *Investigating ultrafast quantum magnetism with machine learning*, SciPost Phys. **7**, 004 (2019), doi:10.21468/scipostphys.7.1.004.

[20] I. L. Gutiérrez and C. B. Mendl, *Real time evolution with neural-network quantum states*, Quantum **6**, 627 (2022), doi:10.22331/q-2022-01-20-627.

[21] M. Schmitt and M. Heyl, *Quantum many-body dynamics in two dimensions with artificial neural networks*, Phys. Rev. Lett. **125**, 100503 (2020), doi:10.1103/PhysRevLett.125.100503.

[22] G. Fabiani, M. D. Bouman and J. H. Mentink, *Supermagnonic propagation in two-dimensional antiferromagnets*, Phys. Rev. Lett. **127**, 097202 (2021), doi:10.1103/physrevlett.127.097202.

[23] D. Hofmann, G. Fabiani, J. Mentink, G. Carleo and M. Sentef, *Role of stochastic noise and generalization error in the time propagation of neural-network quantum states*, SciPost Phys. **12**, 165 (2022), doi:10.21468/SciPostPhys.12.5.165.

[24] F. Vicentini, A. Biella, N. Regnault and C. Ciuti, *Variational neural-network ansatz for steady states in open quantum systems*, Phys. Rev. Lett. **122**, 250503 (2019), doi:10.1103/PhysRevLett.122.250503.

[25] A. Nagy and V. Savona, *Variational quantum Monte Carlo method with a neural-network ansatz for open quantum systems*, Phys. Rev. Lett. **122**, 250501 (2019), doi:10.1103/PhysRevLett.122.250501.

[26] M. J. Hartmann and G. Carleo, *Neural-network approach to dissipative quantum many-body dynamics*, Phys. Rev. Lett. **122**, 250502 (2019), doi:10.1103/PhysRevLett.122.250502.

[27] C.-Y. Park and M. J. Kastoryano, *Expressive power of complex-valued restricted Boltzmann machines for solving non-stoquastic Hamiltonians*, arXiv:2012.08889.

[28] N. Astrakhantsev, T. Westerhout, A. Tiwari, K. Choo, A. Chen, M. H. Fischer, G. Carleo and T. Neupert, *Broken-symmetry ground states of the Heisenberg model on the Pyrochlore lattice*, Phys. Rev. X **11**, 041021 (2021), doi:10.1103/physrevx.11.041021.

[29] L. L. Viteritti, F. Ferrari and F. Becca, *Accuracy of restricted Boltzmann machines for the one-dimensional $J_1 - J_2$ Heisenberg model*, SciPost Phys. **166**, 166 (2022), doi:10.21468/SciPostPhys.12.5.166.

[30] Y. Nomura and M. Imada, *Dirac-type nodal spin liquid revealed by refined quantum many-body solver using neural-network wave function, correlation ratio, and level spectroscopy*, Phys. Rev. X **11**, 031034 (2021), doi:10.1103/PhysRevX.11.031034.

[31] G. Torlai, G. Mazzola, J. Carrasquilla, M. Troyer, R. Melko and G. Carleo, *Neural-network quantum state tomography*, Nat. Phys. **14**, 447 (2018), doi:10.1038/s41567-018-0048-5.

[32] B. Jónsson, B. Bauer and G. Carleo, *Neural-network states for the classical simulation of quantum computing*, arXiv:1808.05232.

[33] M. Abadi et al., *TensorFlow: Large-scale machine learning on heterogeneous systems*, arXiv:1603.04467.

[34] A. Paszke et al., *Pytorch: An imperative style, high-performance deep learning library*, arXiv:1912.01703.

[35] G. Carleo et al., *NetKet: A machine learning toolkit for many-body quantum systems*, SoftwareX **10**, 100311 (2019), doi:10.1016/j.softx.2019.100311.

[36] J. Bradbury et al., *JAX: Composable transformations of Python+NumPy programs*, http://github.com/google/jax (2018).

[37] D. Häfner and F. Vicentini, *mpi4jax: Zero-copy MPI communication of JAX arrays*, JOSS **6**, 3419 (2021), doi:10.21105/joss.03419.

[38] W. Bruinsma, *Plum: Multiple dispatch in Python*, Zenodo (2021), doi:10.5281/zenodo.5774909.

[39] M. Schmitt and M. Reh, *Jvmc: Versatile and performant variational Monte Carlo leveraging automated differentiation and GPU acceleration*, arXiv:2108.03409 .

[40] M.-D. Choi, *Completely positive linear maps on complex matrices*, Linear Algebra Appl. **10**, 285 (1975), doi:10.1016/0024-3795(75)90075-0.

[41] A. Jamiołkowski, *Linear transformations which preserve trace and positive semidefiniteness of operators*, Rep. Math. Phys. **3**, 275 (1972), doi:10.1016/0034-4877(72)90011-0.

[42] S. K. Lam, A. Pitrou and S. Seibert, *Numba: A llvm-based Python JIT compiler*, Proc. Second Workshop LLVM Compil. Infrastruct. HPC (2015), doi:10.1145/2833157.2833162.

[43] K. Kreutz-Delgado, *The complex gradient operator and the CR-calculus*, arXiv:0906.4835.

[44] J. Heek, A. Levskaya, A. Oliver, M. Ritter, B. Rondepierre, A. Steiner and M. van Zee, *Flax: A neural network library and ecosystem for JAX*, http://github.com/google/flax (2020).

[45] T. Hennigan, T. Cai, T. Norman and I. Babuschkin, *Haiku: Sonnet for JAX*, http://github.com/deepmind/dm-haiku (2020).

[46] J. Stokes, B. Chen and S. Veerapaneni, *Numerical and geometrical aspects of flow-based variational quantum Monte Carlo*, arXiv:2203.14824.

[47] R. Jastrow, *Many-body problem with strong forces*, Phys. Rev. **98**, 1479 (1955), doi:10.1103/physrev.98.1479.

[48] D. A. Huse and V. Elser, *Simple variational wave functions for two-dimensional Heisenberg spin-1/2 antiferromagnets*, Phys. Rev. Lett. **60**, 2531 (1988), doi:10.1103/PhysRevLett.60.2531.

[49] C. Roth and A. H. MacDonald, *Group convolutional neural networks improve quantum state accuracy*, arXiv:2104.05085.

[50] O. Sharir, Y. Levine, N. Wies, G. Carleo and A. Shashua, *Deep autoregressive models for the efficient variational simulation of many-body quantum systems*, Phys. Rev. Lett. **124**, 020503 (2020), doi:10.1103/PhysRevLett.124.020503.

[51] G. Torlai and R. G. Melko, *Latent space purification via neural density operators*, Phys. Rev. Lett. **120**, 240503 (2018), doi:10.1103/PhysRevLett.120.240503.

[52] F. Becca and S. Sorella, *Quantum Monte Carlo approaches for correlated systems*, Cambridge University Press, Cambridge, UK, ISBN 9781107129931 (2017), doi:10.1017/9781316417041.

[53] W. K. Hastings, *Monte Carlo sampling methods using Markov chains and their applications*, Biometrika **57**, 97 (1970), doi:10.1093/biomet/57.1.97.

[54] M. V. Berry, *The quantum phase, five years after*, World Scientific, Singapore, ISBN 9789814507585 (1989) doi:10.1142/0613.

[55] G. Fubini, *Sulle metriche definite da una forme hermitiana*, Atti R. Ist. Veneto Sci. Lett. Arti **63**, 502 (1904).

[56] E. Study, *Kürzeste Wege im komplexen Gebiet*, Math. Ann. **60**, 321 (1905), doi:10.1007/BF01457616.

[57] J. Stokes, J. Izaac, N. Killoran and G. Carleo, *Quantum natural gradient*, Quantum **4**, 269 (2020), doi:10.22331/q-2020-05-25-269.

[58] S. Sorella, *Green function Monte Carlo with stochastic reconfiguration*, Phys. Rev. Lett. **80**, 4558 (1998), doi:10.1103/PhysRevLett.80.4558.

[59] S.-i. Amari, *Natural gradient works efficiently in learning*, Neural Comput. **10**, 251 (1998), doi:10.1162/089976698300017746.

[60] C.-Y. Park and M. J. Kastoryano, *Geometry of learning neural quantum states*, Phys. Rev. Research **2**, 023232 (2020), doi:10.1103/physrevresearch.2.023232.

[61] R. A. Horn and C. R. Johnson, *Matrix analysis*, Cambridge University Press, Cambridge, UK, ISBN 9781139020411 (2009), doi:10.1017/cbo9781139020411.

[62] I. Contreras, E. Ercolessi and M. Schiavina, *On the geometry of mixed states and the Fisher information tensor*, J. Math. Phys. **57**, 062209 (2016), doi:10.1063/1.4954328.

[63] H. Weimer, *Variational principle for steady states of dissipative quantum many-body systems*, Phys. Rev. Lett. **114**, 040402 (2015), doi:10.1103/PhysRevLett.114.040402.

[64] X. Yuan, S. Endo, Q. Zhao, Y. Li and S. C. Benjamin, *Theory of variational quantum simulation*, Quantum **3**, 191 (2019), doi:10.22331/q-2019-10-07-191.

[65] M. Hessel, D. Budden, F. Viola, M. Rosca, E. Sezener and T. Hennigan, *Optax: Composable gradient transformation and optimisation, in JAX!*, http://github.com/deepmind/optax (2020).

[66] P. Kramer and M. Saraceno, *Geometry of the time-dependent variational principle in quantum mechanics*, Springer, Berlin, Heidelberg, ISBN 9783540105794 (1981), doi:10.1007/3-540-10579-4.

[67] J. Haegeman, J. I. Cirac, T. J. Osborne, I. Pižorn, H. Verschelde and F. Verstraete, *Time-dependent variational principle for quantum lattices*, Phys. Rev. Lett. **107**, 070601 (2011), doi:10.1103/physrevlett.107.070601.

[68] G. Carleo, F. Becca, M. Schiró and M. Fabrizio, *Localization and glassy dynamics of many-body quantum systems*, Sci. Rep. **2**, 243 (2012), doi:10.1038/srep00243.

[69] H. Breuer and F. Petruccione, *The theory of open quantum systems*, Oxford University Press, Oxford, UK, ISBN 9780199213900 (2007), doi:10.1093/acprof:oso/9780199213900.001.0001.

[70] J. Stoer and R. Bulirsch, *Introduction to numerical analysis*, Springer, New York, US, ISBN 9781441930064 (2002), doi:10.1007/978-0-387-21738-3.

[71] S. d'Ascoli, L. Sagun, J. Bruna and G. Biroli, *Finding the needle in the haystack with convolutions: On the benefits of architectural bias*, arXiv:1906.06766.

[72] Y. Nomura, *Helping restricted Boltzmann machines with quantum-state representation by restoring symmetry*, J. Phys.: Condens. Matter **33**, 174003 (2021), doi:10.1088/1361-648x/abe268.

[73] T. Cohen and M. Welling, *Group equivariant convolutional networks*, arXiv:1602.07576.

[74] V. Heine, *Group theory in quantum mechanics*, Elsevier, Amsterdam, ISBN 9780080092423 (1960), doi:10.1016/C2013-0-01646-5.

[75] M. I. Aroyo, *Space-group symmetry*, International Union of Crystallography, Chester, ISBN 9780470974230 (2016), doi:10.1107/97809553602060000114.

[76] J. D. Dixon, *Computing irreducible representations of groups*, Math. Comp. **24**, 707 (1970), doi:10.1090/s0025-5718-1970-0280611-6.

[77] W. Burnside, *The theory of groups of finite order*, Cambridge University Press, Cambridge, UK, ISBN 9781139237253 (1911), doi:10.1017/CBO9781139237253.

[78] C. J. Bradley and A. P. Cracknell, *The mathematical theory of symmetry in solids: Representation theory for point groups and space groups*, Clarendon Press, Oxford, UK, ISBN 9780198519201 (1972).

[79] G. Bertaina, M. Motta, M. Rossi, E. Vitali and D. E. Galli, *One-dimensional liquid ^{4}He: Dynamical properties beyond Luttinger-liquid theory - Supplemental Material*, Phys. Rev. Lett. **116**, 135302 (2016), doi:10.1103/PhysRevLett.116.135302.

[80] G. Bertaina, M. Motta, M. Rossi, E. Vitali and D. E. Galli, *One-dimensional liquid ^{4}He: Dynamical properties beyond Luttinger-liquid theory*, Phys. Rev. Lett. **116**, 135302 (2016), doi:10.1103/PhysRevLett.116.135302.

[81] R. A. Aziz and H. H. Chen, *An accurate intermolecular potential for argon*, J. Chem. Phys. **67**, 5719 (1977), doi:10.1063/1.434827.

[82] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. Salakhutdinov and A. Smola, *Deep sets*, arXiv:1703.06114.

[83] T. Kato, *On the eigenfunctions of many-particle systems in quantum mechanics*, Commun. Pure Appl. Math. **10**, 151 (1957), doi:10.1002/cpa.3160100201.

[84] G. Pescia, J. Han, A. Lovato, J. Lu and G. Carleo, *Neural-network quantum states for periodic systems in continuous space*, Phys. Rev. Research **4**, 023138 (2022), doi:10.1103/PhysRevResearch.4.023138.

[85] F. Ferrari and F. Becca, *Gapless spin liquid and valence-bond solid in the J_1-J_2 Heisenberg model on the square lattice: Insights from singlet and triplet excitations*, Phys. Rev. B **102**, 014417 (2020), doi:10.1103/physrevb.102.014417.

[86] P. W. Anderson, *An approximate quantum theory of the antiferromagnetic ground state*, Phys. Rev. **86**, 694 (1952), doi:10.1103/PhysRev.86.694.

[87] J. R. McClean et al., *OpenFermion: The electronic structure package for quantum computers*, Quantum Sci. Technol. **5**, 034014 (2020), doi:10.1088/2058-9565/ab8ebc.

[88] J. Nys and G. Carleo, *Variational solutions to fermion-to-qubit mappings in two spatial dimensions*, arXiv:2205.00733.

[89] J. R. Johansson, P. D. Nation and F. Nori, *QuTiP: An open-source Python framework for the dynamics of open quantum systems*, Comput. Phys. Commun. **183**, 1760 (2012), doi:10.1016/j.cpc.2012.02.021.

[90] J. Johansson, P. Nation and F. Nori, *QuTiP 2: A Python framework for the dynamics of open quantum systems*, Comput. Phys. Commun. **184**, 1234 (2013), doi:10.1016/j.cpc.2012.11.019.

[91] P. E. Kloeden and E. Platen, *Numerical solution of stochastic differential equations*, Springer, Berlin, Heidelberg, ISBN 9783642081071 (1992), doi:10.1007/978-3-662-12616-5.

[92] J. Choi, J. J. Dongarra, R. Pozo and D. W. Walker, *ScaLAPACK: A scalable linear algebra library for distributed memory concurrent computers*, IEEE Comput. Soc. Press (1999), doi:10.1109/fmpc.1992.234898.

[93] T. Auckenthaler et al., *Parallel solution of partial symmetric eigenvalue problems from electronic structure calculations*, Parallel Comput. **37**, 783 (2011), doi:10.1016/j.parco.2011.05.002.

[94] M. Gates, J. Kurzak, A. Charara, A. YarKhan and J. Dongarra, *SLATE: Design of a modern distributed and accelerated linear algebra library*, ACM, 1 (2019), doi:10.1145/3295500.3356223.

[95] A. Griewank and A. Walther, *Evaluating derivatives*, Society for Industrial and Applied Mathematics, Philadelphia, US, ISBN 9780898716597 (2008), doi:10.1137/1.9780898717761.

8 Discussion and outlook

In this thesis, we have investigated several aspects of neural quantum states (NQS) as a variational approach for the study of quantum many-body systems.

In Chapter 4 and Publication [P1], we have systematically studied the influence of stochastic effects on the stability of time-dependent variational Monte Carlo (t-VMC) time propagation. To this end, we have used the two-leg antiferromagnetic Heisenberg ladder as a benchmark system. We have found that instabilities become apparent already at small system sizes and are particularly strong compared to the square-lattice version of the same model. Our results illustrate that the geometric structure of the time-dependent variational principle (TDVP), which manifests itself in the broad spectrum and thus the high condition number of the quantum Fisher matrix (QFM), plays an important role in the formation of instabilities by amplifying stochastic noise introduced by the Monte Carlo sampling. These observations are in agreement with related literature, where numerical stability was also identified as the main challenge for NQS+t-VMC simulations [78, 111, 112]. Based on our proposed validation-set TDVP error, we have been able to quantify this effect and demonstrate the importance of regularization methods for stabilizing the t-VMC dynamics. This error diagnostic also applies to other regularization methods, such as the noise-based regularization of Ref. [78], and may help to optimize other hyperparameters, such as the Monte Carlo sample size.

In Chapter 5, we have reviewed several approaches for imposing symmetry constraints on NQS ansätze and proposed a specific symmetry-projection approach for flux quantum numbers in the honeycomb Kitaev model (HKM). We have found that the projection to eigenspaces of a subset of the flux operators improves the convergence of the trial state to the correct global flux eigenspace. However, a clear improvement in variational energy is only observed once a significant fraction of flux quantum numbers is fixed. Since the projection approach has an exponentially scaling computational cost in the number of plaquettes, our results are not yet conclusive on whether reliable convergence can be achieved in a scalable fashion. Further experimentation with network or optimization hyperparameters may provide significant improvements, as has been observed in other frustrated spin systems [88, 89]. Additionally, improvements might be achieved by modifying the NQS ansatz through combination with pair-product states [73, 232] or approaching the HKM in its fermionic representation using a fermionic NQS ansatz [96, 99].

In Chapter 6 and Publication [P2], we have studied the capabilities and scaling properties of NQS based on feed-forward neural networks (FFNNs) for representing highly entangled ground states in the Sachdev-Ye-Kitaev (SYK) model using a supervised-learning approach. In this investigation, we have found that our FFNN ansatz does not learn these ground states with compression beyond the exponential state-vector representation. Thus, while NQS can represent classes of random volume-law entangled states [101, 102], our results show that this capability is not immediately applicable to volume-law ground states of sufficiently complex models. These findings contrast with efficient scaling results for lattice models that are significantly more ordered and spatially structured [86, 240]. Exploring the systems between these two extreme cases (such as spatially structured disordered models) by systematic scaling analyses is an intriguing challenge for future research and may yield important insights into the capabilities and limitations of NQS.

Finally, in Chapter 7, we have discussed the importance of open-source software for the development of computational methods, with a focus on the NetKet framework on which the numerical work of this thesis has been based and which is presented in detail in the software publications [P3, P4]. A primary change in the transition from NetKet 2 to NetKet 3 has been the move from an architecture relying on a C++ core with a Python interface to a pure-Python implementation based on the JAX framework, making use of its just-in-time (JIT) compilation and automatic differentiation capabilities. This has resulted in performance improvements compared to previous hand-optimized code and made it possible to run NQS simulations on CPU and GPU platforms with a reduced overhead in code development. The availability of automatic differentiation simplifies the implementation of custom variational states and thus encourages experimentation and rapid prototyping. Furthermore, none of these computational considerations are restricted to NQS and machine-learning-based approaches. The capabilities of NetKet can also be used to implement and optimize other variational ansätze.

In summary, the results presented in this thesis have highlighted important challenges for NQS methods and we have discussed approaches to overcome those challenges. In our view, the key questions for current and future NQS research are:

1. Understanding what determines the learnability of quantum states and identifying which classes of states and parent Hamiltonians are efficiently treatable using NQS approaches, particularly concerning highly entangled states that are out of reach of established methods. Given the diversity of network architectures falling under the umbrella of the term NQS, it is unlikely that a specific answer can be given in this generality. However, for specific classes of states, answers may be obtainable. Such answers would be of great value to researchers evaluating whether NQS are the right tool for a specific physical use case.

2. Systematic investigation of the effect of network architecture and simulation hyperparameters on the reliability and accuracy of variational Monte Carlo (VMC) optimization. Results concerning the ability of NQS to learn ground states with complicated sign structures have shown that the difficulty of this problem strongly depends on details of the network architecture and training scheme [88, 89, 194, 195]. These studies have paved the way towards formulating best practices for NQS training in specific systems. Further research of this kind would greatly benefit the practical application of NQS simulation methods.

Beyond these general directions, there are many specific advances that would be valuable: NQS can be used to study composite systems, such as bosonic modes coupled to a spin or fermionic lattice model. Such composite systems are particularly relevant for the study of light-matter coupled systems [1, 2] and the flexibility of NQS could be beneficial in those scenarios. Potential ansätze for bosonic modes include standard networks such as the restricted Boltzmann machine (RBM) [91], more specifically tailored architectures such as neural coherent states [94], or potential future extensions based on Gaussian states or their generalizations [129, 130]. Similarly, further exploring fermionic NQS [100] or the combination of pair-product wave functions with NQS for spin systems [73, 81] could yield significant improvements in learning quantum spin liquid (QSL) phases.

Regarding nonequilibrium dynamics, refinement of the diagnostic proposed in Publication [P1] and its combination with other measures, such as noise-based regularization [78] and the detection of spurious modes [115], could be a solid basis for adaptive t-VMC algorithms in the future. This would be valuable for the simulation of quantum dynamics in systems that are challenging to treat using other methods, in particular Kitaev spin liquids [220, 227, 228]. Algorithms to derive spectral properties via an optimization scheme that circumvents explicit time evolution have also been proposed [77, 80] and may complement t-VMC simulation schemes.

Over the years since their introduction, NQS have shown much promise for extending the toolbox of quantum many-body simulation methods towards systems out of reach of other variational approaches. The field has moved past the proof-of-concept stage in several areas, particularly for frustrated quantum systems. The crucial tasks are now to understand what determines the learnability of quantum states, develop best practices for network architecture and simulation schemes, and implement reliable and stable optimization algorithms. If this can be achieved, the potential of NQS can be fully realized, and they can become an integral part of modern quantum simulation in the long term.

A Summary and publications

A short summary of the results in English and German and a list of publications are provided in this appendix on the following pages. The content is identical to the abstract and list of publications on the first pages of this dissertation and repeated in the appendix in fulfillment of the formal requirements of the applicable doctoral degree regulations.

Summary

Computational methods for the efficient simulation of quantum many-body systems are crucial for the study of condensed matter physics.

In this thesis, we investigate numerical properties of neural quantum states (NQS), a machine-learning-inspired variational ansatz based on using an artificial neural network to represent the quantum wave function. This representation can be used to stochastically estimate quantum expectation values, and the NQS ansatz can be trained to approximate ground states as well as real-time dynamics of quantum systems by classical optimization algorithms.

First, we investigate the stability of NQS time propagation with the time-dependent variational Monte Carlo method. Using the antiferromagnetic Heisenberg ladder as a benchmark system, we find that stochastic noise inherent to Monte Carlo sampling can be amplified by the variational equation of motion which can cause numerical instabilities. We propose an error diagnostic that can be used to quantify this effect and demonstrate the influence of regularization methods for the equation of motion on the stability of the dynamics. Subsequently, we discuss the importance of symmetries for improving NQS ground state calculations and propose a symmetry-projection scheme for the honeycomb Kitaev model. Furthermore, we present results of a systematic study of the capabilities of NQS based on feed-forward neural networks to represent highly entangled ground states in the Sachdev-Ye-Kitaev model. In this case, we find that this NQS ansatz does not learn a more efficient representation compared to the exponential scaling of the exact quantum states. This observation highlights the importance of further study to determine which properties decide whether a quantum state is amenable to an efficient approximation by neural quantum states. Finally, we present NetKet, an open-source project and software framework for numerical calculations in quantum many-body systems based on the NQS ansatz and variational Monte Carlo.

Altogether, our work highlights important challenges for the NQS approach and presents ways to help overcome those challenges and develop NQS into a reliable part of the toolbox for simulating quantum many-body physics.

Zusammenfassung

Rechenmethoden zur effizienten Simulation von Quantenvielteilchensystemen sind von essentieller Bedeutung für die Erforschung der Physik kondensierter Materie.

In dieser Arbeit werden numerische Eigenschaften von *Neural-Quantum-States* (NQS) untersucht, einem von Machine-Learning-Methoden inspirierten Variationsansatz, der auf der Darstellung der quantenmechanischen Wellenfunktion durch ein künstliches neuronales Netz basiert. Diese Darstellung kann verwendet werden, um mittels stochastischer Methoden quantenmechanische Erwartungswerte abzuschätzen. Der NQS-Ansatz kann mit klassischen Optimierungsverfahren trainiert werden, um Grundzustände sowie die Realzeitentwicklung von Quantensystemen zu approximieren.

Zunächst untersuchen wir die Stabilität von NQS-Zeitentwicklung unter Verwendung der zeitabhängigen Variational-Monte-Carlo-Methode. Wir zeigen mithilfe der antiferromagnetischen Heisenberg-Leiter als Benchmarksystem, dass stochastisches

Rauschen, welches als natürliche Konsequenz des Monte-Carlo-Verfahrens auftritt, durch die variationelle Bewegungsgleichung verstärkt werden und dadurch numerische Instabilitäten verursachen kann. Wir stellen ein Diagnoseverfahren vor, um diesen Effekt zu quantifizieren und betrachten den Einfluss von Regularisierungsmethoden für die Bewegungsgleichung auf die Stabilität der Zeitentwicklung. Anschließend diskutieren wir die Bedeutung von Symmetrien für die Verbesserung von NQS-Grundzustandsrechnungen und schlagen ein Symmetrie-Projektionsverfahren für das Honeycomb-Kitaev-Modell vor. Weiterhin zeigen wir Ergebnisse einer systematischen Untersuchung der Fähigkeit von auf Feed-Forward-Neural-Networks basierenden NQS zur Darstellung stark verschränkter Grundzustände im Sachdev-Ye-Kitaev-Modell. In diesem Fall zeigt sich, dass dieser NQS-Ansatz im Vergleich zu den exponentiell skalierenden exakten Zuständen keine effizientere Darstellung lernt. Diese Beobachtung unterstreicht die Notwendigkeit weiterer Forschung im Hinblick auf die Frage, welche Eigenschaften eines Quantenzustands dafür entscheidend sind, ob er effizient durch NQS approximiert werden kann. Schlussendlich wird NetKet vorgestellt, ein Open-Source-Projekt und Software-Framework für numerische Rechnungen in Quantenvielteilchensystemen mit Variational-Monte-Carlo und dem NQS-Ansatz.

Insgesamt zeigt diese Arbeit wichtige Herausforderungen für die Anwendung von NQS-Methoden auf und stellt Wege vor, die dabei helfen können, diese zu überwinden und so NQS zu einem zuverlässigen Werkzeug zur Simulation der Quantenvielteilchenphysik zu entwickeln.

List of publications

The following publications are part of this cumulative dissertation:

[P1] **D.H.,** Giammarco Fabiani, Johan H. Mentink, Giuseppe Carleo, Michael A. Sentef.
Role of stochastic noise and generalization error in the time propagation of neural-network quantum states.
SciPost Physics 12, 165 (2022). DOI 10.21468/SciPostPhys.12.5.165.

[P2] Giacomo Passetti, **D.H.,** Pit Neitemeier, Lukas Grunwald, Michael A. Sentef, Dante M. Kennes.
Can neural quantum states learn volume-law ground states?
Preprint (2022). DOI 10.48550/arXiv.2212.02204.

[P3] Giuseppe Carleo, Kenny Choo, **D.H.,** James E. T. Smith, Tom Westerhout, Fabien Alet, Emily J. Davis, Stavros Efthymiou, Ivan Glasser, Sheng-Hsuan Lin, Marta Mauri, Guglielmo Mazzola, Christian B. Mendl, Evert van Nieuwenburg, Ossian O'Reilly, Hugo Théveniaut, Giacomo Torlai, Filippo Vicentini, Alexander Wietek.
NetKet: A machine learning toolkit for many-body quantum systems.
SoftwareX 10, 100311 (2019). DOI 10.1016/j.softx.2019.100311.

[P4] Filippo Vicentini, **D.H.,** Attila Szabó, Dian Wu, Christopher Roth, Clemens Giuliani, Gabriel Pescia, Jannes Nys, Vladimir Vargas-Calderón, Nikita Astrakhantsev, Giuseppe Carleo.
NetKet 3: Machine Learning Toolbox for Many-Body Quantum Systems.
SciPost Physics Codebases 7 (2022). DOI 10.21468/SciPostPhysCodeb.7.

During my doctoral studies, I have contributed to the following additional publications which are not part of this cumulative dissertation:

[O1] Piotr Wrzosek, Krzysztof Wohlfeld, **D.H.,** Tomasz Sowiński, Michael A. Sentef.
Quantum walk versus classical wave: Distinguishing ground states of quantum magnets by spacetime dynamics.
Physical Review B 102, 024440 (2020). DOI `10.1103/PhysRevB.102.024440`.

[O2] **D.H.,** Michael A. Sentef.
Resonant laser excitation and time-domain imaging of chiral topological polariton edge states.
Physical Review Research 2, 033386 (2020). DOI `10.1103/PhysRevResearch.2.033386`.

[O3] Denitsa R. Baykusheva, Mona H. Kalthoff, **D.H.,** Martin Claassen, Dante M. Kennes, Michael A. Sentef, Matteo Mitrano.
Witnessing nonequilibrium entanglement dynamics in a strongly correlated fermionic chain.
Preprint (2022), accepted in *Physical Review Letters*. DOI `10.48550/arXiv.2209.02081`.

Bibliography

[1] Michael Ruggenthaler, Nicolas Tancogne-Dejean, Johannes Flick, Heiko Appel, and Angel Rubio. "From a quantum-electrodynamical light–matter description to novel spectroscopies." In: *Nature Reviews Chemistry* 2.3 (Mar. 2018). DOI: 10.1038/s41570-018-0118.

[2] Alberto de la Torre, Dante M. Kennes, Martin Claassen, Simon Gerber, James W. McIver, and Michael A. Sentef. "Colloquium: Nonthermal pathways to ultrafast control in quantum materials." In: *Reviews of Modern Physics* 93 (4 Oct. 2021), p. 041002. DOI: 10.1103/RevModPhys.93.041002.

[3] Anatoli Polkovnikov, Krishnendu Sengupta, Alessandro Silva, and Mukund Vengalattore. "Colloquium: Nonequilibrium dynamics of closed interacting quantum systems." In: *Reviews of Modern Physics* 83.3 (Aug. 2011), pp. 863–883. DOI: 10.1103/RevModPhys.83.863.

[4] Xingjiang Zhou, Wei-Sheng Lee, Masatoshi Imada, Nandini Trivedi, Philip Phillips, Hae-Young Kee, Päivi Törmä, and Mikhail Eremets. "High-temperature superconductivity." In: *Nature Reviews Physics* 3.7 (May 2021), pp. 462–465. DOI: 10.1038/s42254-021-00324-3.

[5] M. Z. Hasan and C. L. Kane. "Colloquium: Topological insulators." In: *Reviews of Modern Physics* 82.4 (Nov. 2010), pp. 3045–3067. DOI: 10.1103/RevModPhys.82.3045.

[6] Leon Balents. "Spin liquids in frustrated magnets." In: *Nature* 464.7286 (Mar. 2010), pp. 199–208. DOI: 10.1038/nature08917.

[7] Lucile Savary and Leon Balents. "Quantum spin liquids: a review." In: *Reports on Progress in Physics* 80.1 (Nov. 2016), p. 016502. DOI: 10.1088/0034-4885/80/1/016502.

[8] John Preskill. "Quantum Computing in the NISQ era and beyond." In: *Quantum* 2 (Aug. 2018), p. 79. DOI: 10.22331/q-2018-08-06-79.

[9] Antonio Acín, Immanuel Bloch, Harry Buhrman, Tommaso Calarco, Christopher Eichler, Jens Eisert, Daniel Esteve, Nicolas Gisin, Steffen J Glaser, Fedor Jelezko, Stefan Kuhr, Maciej Lewenstein, Max F Riedel, Piet O Schmidt, Rob Thew, Andreas Wallraff, Ian Walmsley, and Frank K Wilhelm. "The quantum technologies roadmap: a European community view." In: *New Journal of Physics* 20.8 (Aug. 2018), p. 080201. DOI: 10.1088/1367-2630/aad1ea.

[10] A. K. Fedorov, N. Gisin, S. M. Beloussov, and A. I. Lvovsky. *Quantum computing at the quantum advantage threshold: a down-to-business review.* 2022. DOI: 10.48550/arXiv.2203.17181.

[11] Daniele Sanvitto and Stéphane Kéna-Cohen. "The road towards polaritonic devices." In: *Nature Materials* 15.10 (July 2016), pp. 1061–1073. DOI: 10.1038/nmat4668.

[12] D. N. Basov, R. D. Averitt, and D. Hsieh. "Towards properties on demand in quantum materials." In: *Nature Materials* 16.11 (Oct. 2017), pp. 1077–1088. DOI: 10.1038/nmat5017.

[13] Dante M. Kennes and Angel Rubio. *A New Era of Quantum Materials Mastery and Quantum Simulators In and Out of Equilibrium.* 2022. DOI: 10.48550/arXiv.2204.11928.

[14] P. W. Anderson. "More Is Different." In: *Science* 177.4047 (Aug. 1972), pp. 393–396. DOI: 10.1126/science.177.4047.393.

[15] P. A. M. Dirac. “Quantum Mechanics of Many-Electron Systems.” In: *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character* 123.792 (Apr. 1929), pp. 714–733. ISSN: 0950-1207, 2053-9150. DOI: 10.1098/rspa.1929.0094.

[16] Richard P. Feynman. “Simulating physics with computers.” In: *International Journal of Theoretical Physics* 21.6-7 (June 1982), pp. 467–488. DOI: 10.1007/bf02650179.

[17] Dylan Herman, Cody Googin, Xiaoyuan Liu, Alexey Galda, Ilya Safro, Yue Sun, Marco Pistoia, and Yuri Alexeev. *A Survey of Quantum Computing for Finance*. 2022. DOI: 10.48550/arXiv.2201.02773.

[18] I. M. Georgescu, S. Ashhab, and Franco Nori. “Quantum simulation.” In: *Reviews of Modern Physics* 86.1 (Mar. 2014), pp. 153–185. DOI: 10.1103/RevModPhys.86.153.

[19] Antoine Browaeys and Thierry Lahaye. “Many-body physics with individually controlled Rydberg atoms.” In: *Nature Physics* 16.2 (Jan. 2020), pp. 132–142. DOI: 10.1038/s41567-019-0733-z.

[20] Pierre Pfeuty. “The one-dimensional Ising model with a transverse field.” In: *Annals of Physics* 57.1 (Mar. 1970), pp. 79–90. DOI: 10.1016/0003-4916(70)90270-8.

[21] H. Bethe. “Zur Theorie der Metalle.” In: *Zeitschrift für Physik* 71.3-4 (Mar. 1931), pp. 205–226. DOI: 10.1007/bf01341708.

[22] Murray T. Batchelor. “The Bethe ansatz after 75 years.” In: *Physics Today* 60.1 (Jan. 2007), pp. 36–40. DOI: 10.1063/1.2709557.

[23] Fedor Levkovich-Maslyuk. “The Bethe ansatz.” In: *Journal of Physics A: Mathematical and Theoretical* 49.32 (July 2016), p. 323004. DOI: 10.1088/1751-8113/49/32/323004.

[24] Alexei Kitaev. “Anyons in an exactly solved model and beyond.” In: *Annals of Physics* 321.1 (Jan. 2006), pp. 2–111. DOI: 10.1016/j.aop.2005.10.005.

[25] Riku Tuovinen. “Time-dependent quantum transport in nanosystems: a nonequilibrium Green’s function approach.” PhD thesis. University of Jyväskylä, 2016. ISBN: 978-951-39-6594-5. URL: https://jyx.jyu.fi/bitstream/handle/123456789/50127/Tuovinen-Riku_2016.pdf.

[26] P. Hohenberg and W. Kohn. “Inhomogeneous Electron Gas.” In: *Physical Review* 136.3B (Nov. 1964), B864–B871. DOI: 10.1103/PhysRev.136.b864.

[27] Erich Runge and E. K. U. Gross. “Density-Functional Theory for Time-Dependent Systems.” In: *Physical Review Letters* 52.12 (Mar. 1984), pp. 997–1000. DOI: 10.1103/PhysRevLett.52.997.

[28] Miguel A.L. Marques, Neepa T. Maitra, Fernando M.S. Nogueira, E.K.U. Gross, and Angel Rubio, eds. *Fundamentals of Time-Dependent Density Functional Theory*. Springer Berlin Heidelberg, 2012. DOI: 10.1007/978-3-642-23518-4.

[29] Alex Kamenev. *Field Theory of Non-Equilibrium Systems*. Cambridge University Press, Sept. 2011. DOI: 10.1017/cbo9781139003667.

[30] Gianluca Stefanucci and Robert van Leeuwen. *Nonequilibrium Many-Body Theory of Quantum Systems*. Cambridge University Press, Mar. 2013. DOI: 10.1017/cbo9781139023979.

[31] Yuto Ashida, Tao Shi, Mari Carmen Bañuls, J. Ignacio Cirac, and Eugene Demler. “Solving Quantum Impurity Problems in and out of Equilibrium with the Variational Approach.” In: *Physical Review Letters* 121.2 (July 2018). DOI: 10.1103/PhysRevLett.121.026805.

[32] Xiao Yuan, Suguru Endo, Qi Zhao, Ying Li, and Simon C. Benjamin. “Theory of variational quantum simulation.” In: *Quantum* 3 (Oct. 2019), p. 191. DOI: 10.22331/q-2019-10-07-191.

[33] F. Verstraete and J. I. Cirac. "Matrix product states represent ground states faithfully." In: *Physical Review B* 73.9 (Mar. 2006). DOI: 10.1103/PhysRevB.73.094423.

[34] F. Verstraete, V. Murg, and J.I. Cirac. "Matrix product states, projected entangled pair states, and variational renormalization group methods for quantum spin systems." In: *Advances in Physics* 57.2 (Mar. 2008), pp. 143–224. DOI: 10.1080/14789940801912366.

[35] Ulrich Schollwöck. "The density-matrix renormalization group in the age of matrix product states." In: *Annals of Physics* 326.1 (Jan. 2011), pp. 96–192. DOI: 10.1016/j.aop.2010.09.012.

[36] S. Bravyi. "Lagrangian representation for fermionic linear optics." In: *Quantum Information and Computation* 5.3 (May 2005), pp. 216–238. DOI: 10.26421/qic5.3-3.

[37] Christian Weedbrook, Stefano Pirandola, Raúl García-Patrón, Nicolas J. Cerf, Timothy C. Ralph, Jeffrey H. Shapiro, and Seth Lloyd. "Gaussian quantum information." In: *Reviews of Modern Physics* 84.2 (May 2012), pp. 621–669. DOI: 10.1103/RevModPhys.84.621.

[38] Bennet Windt, Alexander Jahn, Jens Eisert, and Lucas Hackl. "Local optimization on pure Gaussian state manifolds." In: *SciPost Physics* 10.3 (Mar. 2021). DOI: 10.21468/SciPostPhys.10.3.066.

[39] Giuseppe Carleo, Ignacio Cirac, Kyle Cranmer, Laurent Daudet, Maria Schuld, Naftali Tishby, Leslie Vogt-Maranto, and Lenka Zdeborová. "Machine learning and the physical sciences." In: *Reviews of Modern Physics* 91.4 (Dec. 2019). DOI: 10.1103/RevModPhys.91.045002.

[40] Juan Carrasquilla. "Machine learning for quantum matter." In: *Advances in Physics: X* 5.1 (Jan. 2020), p. 1797528. DOI: 10.1080/23746149.2020.1797528.

[41] Titus Neupert, Mark H Fischer, Eliska Greplova, Kenny Choo, and M. Michael Denner. *Introduction to Machine Learning for the Sciences.* 2021. DOI: 10.48550/arXiv.2102.04883.

[42] Anna Dawid, Julian Arnold, Borja Requena, Alexander Gresch, Marcin Płodzień, Kaelan Donatella, Kim A. Nicoli, Paolo Stornati, Rouven Koch, Miriam Büttner, Robert Okuła, Gorka Muñoz-Gil, Rodrigo A. Vargas-Hernández, Alba Cervera-Lierta, Juan Carrasquilla, Vedran Dunjko, Marylou Gabrié, Patrick Huembeli, Evert van Nieuwenburg, Filippo Vicentini, Lei Wang, Sebastian J. Wetzel, Giuseppe Carleo, Eliška Greplová, Roman Krems, Florian Marquardt, Michał Tomza, Maciej Lewenstein, and Alexandre Dauphin. *Modern applications of machine learning in quantum sciences.* 2022. DOI: 10.48550/arXiv.2204.04198.

[43] Mario Krenn, Jonas Landgraf, Thomas Foesel, and Florian Marquardt. "Artificial intelligence and machine learning for quantum technologies." In: *Physical Review A* 107.1 (Jan. 2023). DOI: 10.1103/PhysRevA.107.010101.

[44] Valentin Gebhart, Raffaele Santagati, Antonio Andrea Gentile, Erik Gauger, David Craig, Natalia Ares, Leonardo Banchi, Florian Marquardt, Luca Pezzè, and Cristian Bonato. *Learning Quantum Systems.* 2022. DOI: 10.48550/arXiv.2207.00298.

[45] Amber Boehnlein, Markus Diefenthaler, Nobuo Sato, Malachi Schram, Veronique Ziegler, Cristiano Fanelli, Morten Hjorth-Jensen, Tanja Horn, Michelle P. Kuchera, Dean Lee, Witold Nazarewicz, Peter Ostroumov, Kostas Orginos, Alan Poon, Xin-Nian Wang, Alexander Scheinker, Michael S. Smith, and Long-Gang Pang. "Colloquium: Machine learning in nuclear physics." In: *Reviews of Modern Physics* 94.3 (Sept. 2022). DOI: 10.1103/RevModPhys.94.031003.

[46] Jürgen Schmidhuber. "Deep learning in neural networks: An overview." In: *Neural Networks* 61 (Jan. 2015), pp. 85–117. DOI: 10.1016/j.neunet.2014.09.003.

[47] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning.” In: *Nature* 521.7553 (May 2015), pp. 436–444. DOI: 10.1038/nature14539.

[48] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016. ISBN: 978-0262035613.

[49] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet classification with deep convolutional neural networks.” In: *Communications of the ACM* 60.6 (May 2017), pp. 84–90. DOI: 10.1145/3065386.

[50] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep Residual Learning for Image Recognition.” In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[51] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. DOI: 10.48550/arXiv.2005.14165.

[52] John Jumper et al. “Highly accurate protein structure prediction with AlphaFold.” In: *Nature* 596.7873 (July 2021), pp. 583–589. DOI: 10.1038/s41586-021-03819-2.

[53] Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J. R. Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, David Silver, Demis Hassabis, and Pushmeet Kohli. “Discovering faster matrix multiplication algorithms with reinforcement learning.” In: *Nature* 610.7930 (Oct. 2022), pp. 47–53. DOI: 10.1038/s41586-022-05172-4.

[54] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. “Mastering the game of Go with deep neural networks and tree search.” In: *Nature* 529.7587 (Jan. 2016), pp. 484–489. DOI: 10.1038/nature16961.

[55] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play.” In: *Science* 362.6419 (Dec. 2018), pp. 1140–1144. DOI: 10.1126/science.aar6404.

[56] Oriol Vinyals et al. “Grandmaster level in StarCraft II using multi-agent reinforcement learning.” In: *Nature* 575.7782 (Oct. 2019), pp. 350–354. DOI: 10.1038/s41586-019-1724-z.

[57] Juan Carrasquilla and Roger G. Melko. “Machine learning phases of matter.” In: *Nature Physics* 13.5 (Feb. 2017), pp. 431–434. DOI: 10.1038/nphys4035.

[58] Evert P. L. van Nieuwenburg, Ye-Hua Liu, and Sebastian D. Huber. “Learning phase transitions by confusion.” In: *Nature Physics* 13.5 (Feb. 2017), pp. 435–439. DOI: 10.1038/nphys4037.

[59] Pengfei Zhang, Huitao Shen, and Hui Zhai. “Machine Learning Topological Invariants with Neural Networks.” In: *Physical Review Letters* 120.6 (Feb. 2018). DOI: 10.1103/PhysRevLett.120.066401.

[60] Benno S. Rem, Niklas Käming, Matthias Tarnowski, Luca Asteria, Nick Fläschner, Christoph Becker, Klaus Sengstock, and Christof Weitenberg. “Identifying quantum phase transitions using artificial neural networks on experimental data.” In: *Nature Physics* 15.9 (July 2019), pp. 917–920. DOI: 10.1038/s41567-019-0554-0.

[61] Annabelle Bohrdt, Christie S. Chiu, Geoffrey Ji, Muqing Xu, Daniel Greif, Markus Greiner, Eugene Demler, Fabian Grusdt, and Michael Knap. “Classifying snapshots of the doped Hubbard model with machine learning.” In: *Nature Physics* 15.9 (July 2019), pp. 921–924. DOI: 10.1038/s41567-019-0565-x.

[62] Niklas Käming, Anna Dawid, Korbinian Kottmann, Maciej Lewenstein, Klaus Sengstock, Alexandre Dauphin, and Christof Weitenberg. “Unsupervised machine learning of topological phase transitions from experimental data.” In: *Machine Learning: Science and Technology* 2.3 (July 2021), p. 035037. DOI: 10.1088/2632-2153/abffe7.

[63] Marin Bukov, Alexandre G. R. Day, Dries Sels, Phillip Weinberg, Anatoli Polkovnikov, and Pankaj Mehta. “Reinforcement Learning in Different Phases of Quantum Control.” In: *Physical Review X* 8.3 (Sept. 2018). DOI: 10.1103/PhysRevX.8.031086.

[64] Thomas Fösel, Murphy Yuezhen Niu, Florian Marquardt, and Li Li. *Quantum circuit optimization with deep reinforcement learning*. 2021. DOI: 10.48550/arXiv.2103.07585.

[65] Jianwei Wang, Stefano Paesani, Raffaele Santagati, Sebastian Knauer, Antonio A. Gentile, Nathan Wiebe, Maurangelo Petruzzella, Jeremy L. O’Brien, John G. Rarity, Anthony Laing, and Mark G. Thompson. “Experimental quantum Hamiltonian learning.” In: *Nature Physics* 13.6 (Mar. 2017), pp. 551–555. DOI: 10.1038/nphys4074.

[66] Agnes Valenti, Evert van Nieuwenburg, Sebastian Huber, and Eliska Greplova. “Hamiltonian learning for quantum error correction.” In: *Physical Review Research* 1.3 (Nov. 2019). DOI: 10.1103/PhysRevResearch.1.033092.

[67] Agnes Valenti, Guliuxin Jin, Julian Léonard, Sebastian D. Huber, and Eliska Greplova. “Scalable Hamiltonian learning for large-scale out-of-equilibrium quantum dynamics.” In: *Physical Review A* 105.2 (Feb. 2022). DOI: 10.1103/PhysRevA.105.023302.

[68] Mario Krenn, Manuel Erhard, and Anton Zeilinger. “Computer-inspired quantum experiments.” In: *Nature Reviews Physics* 2.11 (Sept. 2020), pp. 649–661. DOI: 10.1038/s42254-020-0230-4.

[69] Alba Cervera-Lierta, Mario Krenn, and Alán Aspuru-Guzik. “Design of quantum optical experiments with logic artificial intelligence.” In: *Quantum* 6 (Oct. 2022), p. 836. DOI: 10.22331/q-2022-10-13-836.

[70] Giuseppe Carleo and Matthias Troyer. “Solving the quantum many-body problem with artificial neural networks.” In: *Science* 355.6325 (Feb. 2017), pp. 602–606. DOI: 10.1126/science.aag2302.

[71] W. M. C. Foulkes, L. Mitas, R. J. Needs, and G. Rajagopal. “Quantum Monte Carlo simulations of solids.” In: *Reviews of Modern Physics* 73.1 (Jan. 2001), pp. 33–83. DOI: 10.1103/RevModPhys.73.33.

[72] Federico Becca and Sandro Sorella. *Quantum Monte Carlo Approaches for Correlated Systems*. Cambridge University Press, Nov. 2017. DOI: 10.1017/9781316417041.

[73] Yusuke Nomura, Andrew S. Darmawan, Youhei Yamaji, and Masatoshi Imada. “Restricted Boltzmann machine learning for solving strongly correlated quantum systems.” In: *Physical Review B* 96.20 (Nov. 2017). DOI: 10.1103/PhysRevB.96.205152.

[74] Sirui Lu, Xun Gao, and L.-M. Duan. “Efficient representation of topologically ordered states with restricted Boltzmann machines.” In: *Physical Review B* 99.15 (Apr. 2019). DOI: 10.1103/PhysRevB.99.155136.

[75] Kenny Choo, Titus Neupert, and Giuseppe Carleo. “Two-dimensional frustrated J_1-J_2 model studied with neural network quantum states.” In: *Physical Review B* 100.12 (Sept. 2019). DOI: 10.1103/PhysRevB.100.125124.

[76] Giammarco Fabiani and Johan Mentink. “Investigating Ultrafast Quantum Magnetism with Machine Learning.” In: *SciPost Physics* 7.1 (July 2019), p. 004. ISSN: 2542-4653. DOI: 10.21468/SciPostPhys.7.1.004.

[77] Douglas Hendry and Adrian E. Feiguin. “Machine learning approach to dynamical properties of quantum many-body systems.” In: *Physical Review B* 100.24 (Dec. 2019). DOI: 10.1103/PhysRevB.100.245123.

[78] Markus Schmitt and Markus Heyl. “Quantum Many-Body Dynamics in Two Dimensions with Artificial Neural Networks.” In: *Physical Review Letters* 125.10 (Sept. 2020). DOI: 10.1103/PhysRevLett.125.100503.

[79] G. Fabiani, M. D. Bouman, and J. H. Mentink. “Supermagnonic Propagation in Two-Dimensional Antiferromagnets.” In: *Physical Review Letters* 127.9 (Aug. 2021). DOI: 10.1103/PhysRevLett.127.097202.

[80] Douglas Hendry, Hongwei Chen, Phillip Weinberg, and Adrian E. Feiguin. “Chebyshev expansion of spectral functions using restricted Boltzmann machines.” In: *Physical Review B* 104.20 (Nov. 2021). DOI: 10.1103/PhysRevB.104.205130.

[81] Yusuke Nomura and Masatoshi Imada. “Dirac-Type Nodal Spin Liquid Revealed by Refined Quantum Many-Body Solver Using Neural-Network Wave Function, Correlation Ratio, and Level Spectroscopy.” In: *Physical Review X* 11.3 (Aug. 2021). DOI: 10.1103/PhysRevX.11.031034.

[82] Yusuke Nomura. “Helping restricted Boltzmann machines with quantum-state representation by restoring symmetry.” In: *Journal of Physics: Condensed Matter* 33.17 (Apr. 2021), p. 174003. DOI: 10.1088/1361-648x/abe268.

[83] Mohammadreza Noormandipour, Sun Youran, and Babak Haghighat. “Restricted Boltzmann machine representation for the groundstate and excited states of Kitaev Honeycomb model.” In: *Machine Learning: Science and Technology* 3.1 (Dec. 2021), p. 015010. DOI: 10.1088/2632-2153/ac3ddf.

[84] Eric Zou, Erik Long, and Erhai Zhao. “Learning a compass spin model with neural network quantum states.” In: *Journal of Physics: Condensed Matter* 34.12 (Jan. 2022), p. 125802. DOI: 10.1088/1361-648x/ac43ff.

[85] Agnes Valenti, Eliska Greplova, Netanel H. Lindner, and Sebastian D. Huber. “Correlation-enhanced neural networks as interpretable variational quantum states.” In: *Physical Review Research* 4.1 (Jan. 2022). DOI: 10.1103/PhysRevResearch.4.l012010.

[86] Luciano Loris Viteritti, Francesco Ferrari, and Federico Becca. “Accuracy of restricted Boltzmann machines for the one-dimensional J_1-J_2 Heisenberg model.” In: *SciPost Physics* 12.5 (May 2022). DOI: 10.21468/SciPostPhys.12.5.166.

[87] Markus Schmitt, Marek M. Rams, Jacek Dziarmaga, Markus Heyl, and Wojciech H. Zurek. “Quantum phase transition dynamics in the two-dimensional transverse-field Ising model.” In: *Science Advances* 8.37 (Sept. 2022). DOI: 10.1126/sciadv.abl6850.

[88] Christopher Roth, Attila Szabó, and Allan MacDonald. *High-accuracy variational Monte Carlo for frustrated magnets with deep neural networks.* 2022. DOI: 10.48550/arXiv.2211.07749.

[89] Moritz Reh, Markus Schmitt, and Martin Gärttner. *Optimizing Design Choices for Neural Quantum States*. 2023. DOI: 10.48550/arXiv.2301.06788.

[90] Pascal M. Vecsei, Christian Flindt, and Jose L. Lado. *Lee-Yang theory of quantum phase transitions with neural network quantum states*. 2023. DOI: 10.48550/arXiv.2301.09923.

[91] Hiroki Saito. "Solving the Bose–Hubbard Model with Machine Learning." In: *Journal of the Physical Society of Japan* 86.9 (Sept. 2017), p. 093001. DOI: 10.7566/jpsj.86.093001.

[92] Kenny Choo, Giuseppe Carleo, Nicolas Regnault, and Titus Neupert. "Symmetries and Many-Body Excitations with Neural-Network Quantum States." In: *Physical Review Letters* 121.16 (Oct. 2018). DOI: 10.1103/PhysRevLett.121.167204.

[93] Kristopher McBrian, Giuseppe Carleo, and Ehsan Khatami. "Ground state phase diagram of the one-dimensional Bose-Hubbard model from restricted Boltzmann machines." In: *Journal of Physics: Conference Series* 1290.1 (Oct. 2019), p. 012005. DOI: 10.1088/1742-6596/1290/1/012005.

[94] Wojciech Rzadkowski, Mikhail Lemeshko, and Johan H. Mentink. "Artificial neural network states for nonadditive systems." In: *Physical Review B* 106.15 (Oct. 2022). DOI: 10.1103/PhysRevB.106.155127.

[95] Ziyan Zhu, Marios Mattheakis, Weiwei Pan, and Efthimios Kaxiras. *HubbardNet: Efficient Predictions of the Bose-Hubbard Model Spectrum with Deep Neural Networks*. 2022. DOI: 10.48550/arXiv.2212.13678.

[96] Di Luo and Bryan K. Clark. "Backflow Transformations via Neural Networks for Quantum Many-Body Wave Functions." In: *Physical Review Letters* 122.22 (June 2019). DOI: 10.1103/PhysRevLett.122.226401.

[97] Kenny Choo, Antonio Mezzacapo, and Giuseppe Carleo. "Fermionic neural-network states for ab-initio electronic structure." In: *Nature Communications* 11.1 (May 2020). DOI: 10.1038/s41467-020-15724-9.

[98] James Stokes, Javier Robledo Moreno, Eftychios A. Pnevmatikakis, and Giuseppe Carleo. "Phases of two-dimensional spinless lattice fermions with first-quantized deep neural-network quantum states." In: *Physical Review B* 102.20 (Nov. 2020). DOI: 10.1103/PhysRevB.102.205122.

[99] Javier Robledo Moreno, Giuseppe Carleo, Antoine Georges, and James Stokes. "Fermionic wave functions from neural-network constrained hidden states." In: *Proceedings of the National Academy of Sciences* 119.32 (Aug. 2022). DOI: 10.1073/pnas.2122059119.

[100] Jan Hermann, James Spencer, Kenny Choo, Antonio Mezzacapo, W. M. C. Foulkes, David Pfau, Giuseppe Carleo, and Frank Noé. *Ab-initio quantum chemistry with neural-network wavefunctions*. 2022. DOI: 10.48550/arXiv.2208.12590.

[101] Dong-Ling Deng, Xiaopeng Li, and S. Das Sarma. "Quantum Entanglement in Neural Network States." In: *Physical Review X* 7 (2 May 2017), p. 021021. DOI: 10.1103/PhysRevX.7.021021.

[102] Xiao-Qi Sun, Tamra Nebabu, Xizhi Han, Michael O. Flynn, and Xiao-Liang Qi. "Entanglement features of random neural network quantum states." In: *Physical Review B* 106 (11 Sept. 2022), p. 115138. DOI: 10.1103/PhysRevB.106.115138.

[103] Ivan Glasser, Nicola Pancotti, Moritz August, Ivan D. Rodriguez, and J. Ignacio Cirac. "Neural-Network Quantum States, String-Bond States, and Chiral Topological States." In: *Physical Review X* 8.1 (Jan. 2018). DOI: 10.1103/PhysRevX.8.011006.

[104] Stephen R Clark. “Unifying neural-network quantum states and correlator product states via tensor networks.” In: *Journal of Physics A: Mathematical and Theoretical* 51.13 (Feb. 2018), p. 135301. DOI: 10.1088/1751-8121/aaaaf2.

[105] Raphael Kaubruegger, Lorenzo Pastori, and Jan Carl Budich. “Chiral topological phases from artificial neural networks.” In: *Physical Review B* 97.19 (May 2018). DOI: 10.1103/PhysRevB.97.195136.

[106] Sheng-Hsuan Lin and Frank Pollmann. “Scaling of Neural-Network Quantum States for Time Evolution.” In: *physica status solidi (b)* 259.5 (Jan. 2022), p. 2100172. DOI: 10.1002/pssb.202100172.

[107] Sören Sonnenburg, Mikio L. Braun, Cheng Soon Ong, Samy Bengio, Léon Bottou, Geoffrey Holmes, Yann LeCun, Klaus-Robert Müller, Fernando Pereira, Carl Edward Rasmussen, Gunnar Rätsch, Bernhard Schölkopf, Alexander J. Smola, Pascal Vincent, Jason Weston, and Robert C. Williamson. “The Need for Open Source Software in Machine Learning.” In: *J. Mach. Learn. Res.* 8 (2007), pp. 2443–2466. URL: https://dl.acm.org/doi/10.5555/1314498.1314577.

[108] Max Langenkamp and Daniel N. Yue. “How Open Source Machine Learning Software Shapes AI.” In: *Proceedings of the 2022 AAAI/ACM Conference on AI, Ethics, and Society*. ACM, July 2022. DOI: 10.1145/3514094.3534167.

[109] Matthew J. S. Beach, Isaac De Vlugt, Anna Golubeva, Patrick Huembeli, Bohdan Kulchytskyy, Xiuzhe Luo, Roger G. Melko, Ejaaz Merali, and Giacomo Torlai. “QuCumber: wavefunction reconstruction with neural networks.” In: *SciPost Physics* 7 (1 2019), p. 9. DOI: 10.21468/SciPostPhys.7.1.009.

[110] Markus Schmitt and Moritz Reh. “jVMC: Versatile and performant variational Monte Carlo leveraging automated differentiation and GPU acceleration.” In: *SciPost Physics Codebases* 2 (2022). DOI: 10.21468/SciPostPhysCodeb.2.

[111] Stefanie Czischek, Martin Gärttner, and Thomas Gasenzer. “Quenches near Ising quantum criticality as a challenge for artificial neural networks.” In: *Physical Review B* 98.2 (July 2018). DOI: 10.1103/PhysRevB.98.024311.

[112] Irene López Gutiérrez and Christian B. Mendl. “Real time evolution with neural-network quantum states.” In: *Quantum* 6 (Jan. 2022), p. 627. DOI: 10.22331/q-2022-01-20-627.

[113] Kaelan Donatella, Zakari Denis, Alexandre Le Boité, and Cristiano Ciuti. *Dynamics with autoregressive neural quantum states: application to critical quench dynamics*. 2022. DOI: 10.48550/arXiv.2209.03241.

[114] Robert J. Webber and Michael Lindsey. “Rayleigh-Gauss-Newton optimization with enhanced sampling for variational Monte Carlo.” In: *Physical Review Research* 4.3 (Aug. 2022). DOI: 10.1103/PhysRevResearch.4.033099.

[115] Huan Zhang, Robert J. Webber, Michael Lindsey, Timothy C. Berkelbach, and Jonathan Weare. *Understanding and eliminating spurious modes in variational Monte Carlo using collective variables*. 2022. DOI: 10.48550/arXiv.2211.09767.

[116] Giuseppe Carleo, Yusuke Nomura, and Masatoshi Imada. “Constructing exact representations of quantum many-body systems with deep neural networks.” In: *Nature Communications* 9.1 (Dec. 2018). DOI: 10.1038/s41467-018-07520-3.

[117] Jing Chen, Song Cheng, Haidong Xie, Lei Wang, and Tao Xiang. “Equivalence of restricted Boltzmann machines and tensor network states.” In: *Physical Review B* 97.8 (Feb. 2018). DOI: 10.1103/PhysRevB.97.085104.

[118] Yoav Levine, Or Sharir, Nadav Cohen, and Amnon Shashua. "Quantum Entanglement in Deep Learning Architectures." In: *Physical Review Letters* 122.6 (Feb. 2019). DOI: 10.1103/PhysRevLett.122.065301.

[119] Yichen Huang and Joel E. Moore. "Neural Network Representation of Tensor Network and Chiral States." In: *Physical Review Letters* 127.17 (Oct. 2021). DOI: 10.1103/PhysRevLett.127.170601.

[120] Or Sharir, Amnon Shashua, and Giuseppe Carleo. "Neural tensor contractions and the expressive power of deep neural quantum states." In: *Physical Review B* 106.20 (Nov. 2022). DOI: 10.1103/PhysRevB.106.205136.

[121] Abhay Ashtekar and Troy A. Schilling. *Geometrical Formulation of Quantum Mechanics.* 1997. DOI: 10.48550/arXiv.gr-qc/9706069.

[122] Alexander Wietek and Andreas M. Läuchli. "Sublattice coding algorithm and distributed memory parallelization for large-scale exact diagonalizations of quantum many-body systems." In: *Physical Review E* 98.3 (Sept. 2018). DOI: 10.1103/PhysRevE.98.033309.

[123] Edwin Pednault, John A. Gunnels, Giacomo Nannicini, Lior Horesh, Thomas Magerlein, Edgar Solomonik, Erik W. Draeger, Eric T. Holland, and Robert Wisnieff. *Pareto-Efficient Quantum Circuit Simulation Using Tensor Contraction Deferral.* 2017. DOI: 10.48550/arXiv.1710.05867.

[124] Hans De Raedt, Fengping Jin, Dennis Willsch, Madita Willsch, Naoki Yoshioka, Nobuyasu Ito, Shengjun Yuan, and Kristel Michielsen. "Massively Parallel Quantum Computer Simulator, Eleven Years Later." In: *Computer Physics Communications* 237 (Apr. 2019), pp. 47–61. ISSN: 00104655. DOI: 10.1016/j.cpc.2018.11.005.

[125] Edwin Pednault, John A. Gunnels, Giacomo Nannicini, Lior Horesh, and Robert Wisnieff. *Leveraging Secondary Storage to Simulate Deep 54-Qubit Sycamore Circuits.* 2019. DOI: 10.48550/arXiv.1910.09534.

[126] John Preskill. *Quantum computing and the entanglement frontier.* 2012. DOI: 10.48550/arXiv.1203.5813.

[127] J. Ignacio Cirac, David Pérez-García, Norbert Schuch, and Frank Verstraete. "Matrix product states and projected entangled pair states: Concepts, symmetries, theorems." In: *Reviews of Modern Physics* 93.4 (Dec. 2021). DOI: 10.1103/RevModPhys.93.045003.

[128] Sevag Gharibian, Yichen Huang, Zeph Landau, and Seung Woo Shin. "Quantum Hamiltonian Complexity." In: *Foundations and Trends® in Theoretical Computer Science* 10.3 (2015), pp. 159–282. DOI: 10.1561/0400000066.

[129] Lucas Hackl, Tommaso Guaita, Tao Shi, Jutho Haegeman, Eugene Demler, and J. Ignacio Cirac. "Geometry of variational methods: dynamics of closed quantum systems." In: *SciPost Physics* 9 (2020), p. 048. DOI: 10.21468/SciPostPhys.9.4.048.

[130] Tao Shi, Eugene Demler, and J. Ignacio Cirac. "Variational study of fermionic and bosonic systems with non-Gaussian states: Theory and applications." In: *Annals of Physics* 390 (Mar. 2018), pp. 245–302. DOI: 10.1016/j.aop.2017.11.014.

[131] Tommaso Guaita, Lucas Hackl, Tao Shi, Eugene Demler, and J. Ignacio Cirac. "Generalization of group-theoretic coherent states for variational calculations." In: *Physical Review Research* 3.2 (May 2021). DOI: 10.1103/PhysRevResearch.3.023090.

[132] Kota Ido, Takahiro Ohgoe, and Masatoshi Imada. "Time-dependent many-variable variational Monte Carlo method for nonequilibrium strongly correlated electron systems." In: *Physical Review B* 92.24 (Dec. 2015). DOI: 10.1103/PhysRevB.92.245106.

[133] Takahiro Misawa, Satoshi Morita, Kazuyoshi Yoshimi, Mitsuaki Kawamura, Yuichi Motoyama, Kota Ido, Takahiro Ohgoe, Masatoshi Imada, and Takeo Kato. "mVMC—Open-source software for many-variable variational Monte Carlo method." In: *Computer Physics Communications* 235 (Feb. 2019), pp. 447–462. DOI: 10.1016/j.cpc.2018.08.014.

[134] Johannes Hauschild and Frank Pollmann. "Efficient numerical simulations with Tensor Networks: Tensor Network Python (TeNPy)." In: *SciPost Physics Lecture Notes* (Oct. 2018). DOI: 10.21468/SciPostPhysLectNotes.5.

[135] Warren S. McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity." In: *The Bulletin of Mathematical Biophysics* 5.4 (Dec. 1943), pp. 115–133. DOI: 10.1007/bf02478259.

[136] S. C. Kleene. "Representation of Events in Nerve Nets and Finite Automata." In: *Automata Studies. (AM-34)*. Princeton University Press, Dec. 1956, pp. 3–42. DOI: 10.1515/9781400882618-002.

[137] F. Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain." In: *Psychological Review* 65.6 (1958), pp. 386–408. DOI: 10.1037/h0042519.

[138] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer New York, 2009. DOI: 10.1007/978-0-387-84858-7.

[139] Wolfgang Maass. "Networks of spiking neurons: The third generation of neural network models." In: *Neural Networks* 10.9 (Dec. 1997), pp. 1659–1671. DOI: 10.1016/s0893-6080(97)00011-7.

[140] G. Cybenko. "Approximation by superpositions of a sigmoidal function." In: *Mathematics of Control, Signals, and Systems* 2.4 (Dec. 1989), pp. 303–314. DOI: 10.1007/bf02551274.

[141] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks." In: *Neural Networks* 4.2 (1991), pp. 251–257. DOI: 10.1016/0893-6080(91)90009-t.

[142] Allan Pinkus. "Approximation theory of the MLP model in neural networks." In: *Acta Numerica* 8 (Jan. 1999), pp. 143–195. DOI: 10.1017/s0962492900002919.

[143] Patrick Kidger and Terry Lyons. "Universal Approximation with Deep Narrow Networks." In: *Proceedings of Thirty Third Conference on Learning Theory*. Ed. by Jacob Abernethy and Shivani Agarwal. Vol. 125. Proceedings of Machine Learning Research. PMLR, Sept. 2020, pp. 2306–2327. URL: https://proceedings.mlr.press/v125/kidger20a.html.

[144] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep Sparse Rectifier Neural Networks." In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Nov. 2011, pp. 315–323. URL: https://proceedings.mlr.press/v15/glorot11a.html.

[145] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. *Self-Normalizing Neural Networks*. 2017. DOI: 10.48550/arXiv.1706.02515.

[146] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. "Searching for Activation Functions." In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net, 2018. URL: https://openreview.net/forum?id=Hkuq2EkPf.

[147] Bing Xu, Ruitong Huang, and Mu Li. *Revise Saturated Activation Functions*. 2016. DOI: 10.48550/arXiv.1602.05980.

[148] Steffen Eger, Paul Youssef, and Iryna Gurevych. "Is it Time to Swish? Comparing Deep Learning Activation Functions Across NLP tasks." In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*. Ed. by Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii. Association for Computational Linguistics, 2018, pp. 4415–4424. DOI: 10.18653/v1/d18-1472.

[149] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*. 2015. DOI: 10.48550/arXiv.1511.07289.

[150] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. "Understanding deep learning (still) requires rethinking generalization." In: *Communications of the ACM* 64.3 (2021), pp. 107–115. DOI: 10.1145/3446776.

[151] Joshua Bassey, Lijun Qian, and Xiangfang Li. *A Survey of Complex-Valued Neural Networks*. 2021. DOI: 10.48550/arXiv.2101.12249.

[152] Bjarni Jónsson, Bela Bauer, and Giuseppe Carleo. *Neural-Network States for the Classical Simulation of Quantum Computing*. Aug. 2018. DOI: 10.48550/arXiv.1808.05232.

[153] Giacomo Torlai, Guglielmo Mazzola, Juan Carrasquilla, Matthias Troyer, Roger Melko, and Giuseppe Carleo. "Neural-network quantum state tomography." In: *Nature Physics* 14.5 (Feb. 2018), pp. 447–450. DOI: 10.1038/s41567-018-0048-5.

[154] Juan Carrasquilla and Giacomo Torlai. "How To Use Neural Networks To Investigate Quantum Many-Body Physics." In: *PRX Quantum* 2.4 (Nov. 2021). DOI: 10.1103/PRXQuantum.2.040201.

[155] Stefanie Czischek, M. Schuyler Moss, Matthew Radzihovsky, Ejaaz Merali, and Roger G. Melko. "Data-enhanced variational Monte Carlo simulations for Rydberg atom arrays." In: *Physical Review B* 105.20 (May 2022). DOI: 10.1103/PhysRevB.105.205108.

[156] Alexandra Nagy and Vincenzo Savona. "Variational Quantum Monte Carlo Method with a Neural-Network Ansatz for Open Quantum Systems." In: *Physical Review Letters* 122.25 (June 2019). DOI: 10.1103/PhysRevLett.122.250501.

[157] Michael J. Hartmann and Giuseppe Carleo. "Neural-Network Approach to Dissipative Quantum Many-Body Dynamics." In: *Physical Review Letters* 122.25 (June 2019). DOI: 10.1103/PhysRevLett.122.250502.

[158] Filippo Vicentini, Alberto Biella, Nicolas Regnault, and Cristiano Ciuti. "Variational Neural-Network Ansatz for Steady States in Open Quantum Systems." In: *Physical Review Letters* 122.25 (June 2019). DOI: 10.1103/PhysRevLett.122.250503.

[159] Nobuyuki Yoshioka and Ryusuke Hamazaki. "Constructing neural stationary states for open quantum many-body systems." In: *Physical Review B* 99.21 (June 2019). DOI: 10.1103/PhysRevB.99.214306.

[160] P. Smolensky. "Information Processing in Dynamical Systems: Foundations of Harmony Theory." In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. Cambridge, MA, USA: MIT Press, 1986, pp. 194–281. ISBN: 026268053X.

[161] G. E. Hinton and R. R. Salakhutdinov. "Reducing the Dimensionality of Data with Neural Networks." In: *Science* 313.5786 (July 2006), pp. 504–507. DOI: 10.1126/science.1127647.

[162] Geoffrey E. Hinton. "A Practical Guide to Training Restricted Boltzmann Machines." In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012, pp. 599–619. DOI: 10.1007/978-3-642-35289-8_32.

[163] Roger G. Melko, Giuseppe Carleo, Juan Carrasquilla, and J. Ignacio Cirac. "Restricted Boltzmann machines in quantum physics." In: *Nature Physics* 15.9 (June 2019), pp. 887–892. DOI: 10.1038/s41567-019-0545-1.

[164] Patrick Huembeli, Juan Miguel Arrazola, Nathan Killoran, Masoud Mohseni, and Peter Wittek. "The physics of energy-based models." In: *Quantum Machine Intelligence* 4.1 (Jan. 2022). DOI: 10.1007/s42484-021-00057-7.

[165] Nicolas Le Roux and Yoshua Bengio. "Representational Power of Restricted Boltzmann Machines and Deep Belief Networks." In: *Neural Computation* 20.6 (June 2008), pp. 1631–1649. DOI: 10.1162/neco.2008.04-07-510.

[166] J. Eisert, M. Cramer, and M. B. Plenio. "Colloquium: Area laws for the entanglement entropy." In: *Reviews of Modern Physics* 82 (1 Feb. 2010), pp. 277–306. DOI: 10.1103/RevModPhys.82.277.

[167] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, R. Howard, Wayne Hubbard, and Lawrence Jackel. "Handwritten Digit Recognition with a Back-Propagation Network." In: *Advances in Neural Information Processing Systems*. Ed. by D. Touretzky. Vol. 2. Morgan-Kaufmann, 1989. URL: https://proceedings.neurips.cc/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf.

[168] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[169] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Li Wang, Gang Wang, Jianfei Cai, and Tsuhan Chen. *Recent Advances in Convolutional Neural Networks*. 2015. DOI: 10.48550/arXiv.1512.07108.

[170] Taco Cohen and Max Welling. "Group Equivariant Convolutional Networks." In: *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*. Ed. by Maria-Florina Balcan and Kilian Q. Weinberger. Vol. 48. JMLR Workshop and Conference Proceedings. JMLR.org, 2016, pp. 2990–2999. URL: http://proceedings.mlr.press/v48/cohenc16.html.

[171] Christopher Roth and Allan H. MacDonald. *Group Convolutional Neural Networks Improve Quantum State Accuracy*. 2021. DOI: 10.48550/arXiv.2104.05085.

[172] Bryan Clark. *Variational Monte Carlo Notes for Boulder Summer School 2010*. July 21, 2010. URL: https://boulderschool.yale.edu/sites/default/files/files/VMC_Final_Notes.pdf.

[173] W. L. McMillan. "Ground State of Liquid He^4." In: *Physical Review* 138.2A (Apr. 1965), A442–A451. DOI: 10.1103/PhysRev.138.a442.

[174] Max Born. "Zur Quantenmechanik der Stossvorgänge." In: *Zeitschrift für Physik* 37.12 (Dec. 1926), pp. 863–867. DOI: 10.1007/bf01397477.

[175] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. "Equation of State Calculations by Fast Computing Machines." In: *The Journal of Chemical Physics* 21.6 (June 1953), pp. 1087–1092. DOI: 10.1063/1.1699114.

[176] W. K. Hastings. "Monte Carlo sampling methods using Markov chains and their applications." In: *Biometrika* 57.1 (Apr. 1970), pp. 97–109. DOI: 10.1093/biomet/57.1.97.

[177] Or Sharir, Yoav Levine, Noam Wies, Giuseppe Carleo, and Amnon Shashua. "Deep Autoregressive Models for the Efficient Variational Simulation of Many-Body Quantum Systems." In: *Physical Review Letters* 124.2 (Jan. 2020). DOI: 10.1103/PhysRevLett.124.020503.

[178] Mohamed Hibat-Allah, Martin Ganahl, Lauren E. Hayward, Roger G. Melko, and Juan Carrasquilla. "Recurrent neural network wave functions." In: *Physical Review Research* 2.2 (June 2020). DOI: 10.1103/PhysRevResearch.2.023358.

[179] Jannes Nys and Giuseppe Carleo. "Variational solutions to fermion-to-qubit mappings in two spatial dimensions." In: *Quantum* 6 (Oct. 2022), p. 833. DOI: 10.22331/q-2022-10-13-833.

[180] Sebastian Ruder. *An overview of gradient descent optimization algorithms.* 2016. DOI: 10.48550/arXiv.1609.04747.

[181] Alex Graves. *Generating Sequences With Recurrent Neural Networks.* 2013. DOI: 10.48550/arXiv.1308.0850.

[182] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization.* 2014. DOI: 10.48550/arXiv.1412.6980.

[183] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." In: *Journal of Machine Learning Research* 12.61 (2011), pp. 2121–2159. URL: http://jmlr.org/papers/v12/duchi11a.html.

[184] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. "The Marginal Value of Adaptive Gradient Methods in Machine Learning." In: *Advances in Neural Information Processing Systems.* Ed. by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/81b3833e2504647f9d794f7d7b9bf341-Paper.pdf.

[185] Yingxue Zhou, Belhal Karimi, Jinxing Yu, Zhiqiang Xu, and Ping Li. "Towards Better Generalization of Adaptive Gradient Methods." In: *Advances in Neural Information Processing Systems.* Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 810–821. URL: https://proceedings.neurips.cc/paper/2020/file/08fb104b0f2f838f3ce2d2b3741a12c2-Paper.pdf.

[186] Sandro Sorella. "Green Function Monte Carlo with Stochastic Reconfiguration." In: *Physical Review Letters* 80.20 (May 1998), pp. 4558–4561. DOI: 10.1103/PhysRevLett.80.4558.

[187] Sandro Sorella. "Generalized Lanczos algorithm for variational quantum Monte Carlo." In: *Physical Review B* 64.2 (June 2001). DOI: 10.1103/PhysRevB.64.024512.

[188] Shun-ichi Amari. "Natural Gradient Works Efficiently in Learning." In: *Neural Computation* 10.2 (Feb. 1998), pp. 251–276. ISSN: 0899-7667, 1530-888X. DOI: 10.1162/089976698300017746.

[189] Shun-ichi Amari. *Information Geometry and Its Applications.* Applied Mathematical Sciences volume 194. OCLC: ocn930997330. Japan: Springer, 2016. ISBN: 978-4-431-55977-1.

[190] James Stokes, Brian Chen, and Shravan Veerapaneni. *Numerical and geometrical aspects of flow-based variational quantum Monte Carlo.* 2022. DOI: 10.48550/arXiv.2203.14824.

[191] S. Bravyi, D.P. DiVincenzo, R. Oliveira, and B.M. Terhal. "The complexity of stoquastic local Hamiltonian problems." In: *Quantum Information and Computation* 8.5 (May 2008), pp. 361–385. DOI: 10.26421/qic8.5-1.

[192] Matthias Troyer and Uwe-Jens Wiese. "Computational Complexity and Fundamental Limitations to Fermionic Quantum Monte Carlo Simulations." In: *Physical Review Letters* 94.17 (May 2005). DOI: 10.1103/PhysRevLett.94.170201.

[193] Gaopei Pan and Zi Yang Meng. *Sign Problem in Quantum Monte Carlo Simulation.* 2022. DOI: 10.48550/arXiv.2204.08777.

[194] Tom Westerhout, Nikita Astrakhantsev, Konstantin S. Tikhonov, Mikhail I. Katsnelson, and Andrey A. Bagrov. “Generalization properties of neural network approximations to frustrated magnet ground states.” In: *Nature Communications* 11.1 (Mar. 2020). DOI: 10.1038/s41467-020-15402-w.

[195] Attila Szabó and Claudio Castelnovo. “Neural network wave functions and the sign problem.” In: *Physical Review Research* 2.3 (July 2020). DOI: 10.1103/PhysRevResearch.2.033075.

[196] Ao Chen, Kenny Choo, Nikita Astrakhantsev, and Titus Neupert. “Neural network evolution strategy for solving quantum sign structures.” In: *Physical Review Research* 4.2 (May 2022). DOI: 10.1103/PhysRevResearch.4.l022026.

[197] Tom Westerhout, Mikhail I. Katsnelson, and Andrey A. Bagrov. *Unveiling ground state sign structures of frustrated quantum systems via non-glassy Ising models*. 2022. DOI: 10.48550/arXiv.2207.10675.

[198] Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng, eds. *Handbook of Markov Chain Monte Carlo*. Chapman and Hall/CRC, May 2011. ISBN: 9780429138508. DOI: 10.1201/b10905.

[199] Michael Betancourt. *A Conceptual Introduction to Hamiltonian Monte Carlo*. 2017. DOI: 10.48550/arXiv.1701.02434.

[200] Tom Vieijra, Corneel Casert, Jannes Nys, Wesley De Neve, Jutho Haegeman, Jan Ryckebusch, and Frank Verstraete. “Restricted Boltzmann Machines for Quantum States with Non-Abelian or Anyonic Symmetries.” In: *Physical Review Letters* 124.9 (Mar. 2020). DOI: 10.1103/PhysRevLett.124.097201.

[201] Vinay Ambegaokar and Matthias Troyer. “Estimating errors reliably in Monte Carlo simulations of the Ehrenfest model.” In: *American Journal of Physics* 78.2 (Feb. 2010), pp. 150–157. DOI: 10.1119/1.3247985.

[202] Andrew Gelman, John B. Carlin, Hal S. Stern, David B. Dunson, Aki Vehtari, and Donald B. Rubin. *Bayesian Data Analysis*. 3rd ed. Chapman and Hall/CRC, 2013. ISBN: 9781439840955.

[203] Aki Vehtari, Andrew Gelman, Daniel Simpson, Bob Carpenter, and Paul-Christian Bürkner. “Rank-Normalization, Folding, and Localization: An Improved Rˆ for Assessing Convergence of MCMC (with Discussion).” In: *Bayesian Analysis* 16.2 (June 2021). DOI: 10.1214/20-ba1221.

[204] Jutho Haegeman, J. Ignacio Cirac, Tobias J. Osborne, Iztok Pižorn, Henri Verschelde, and Frank Verstraete. “Time-Dependent Variational Principle for Quantum Lattices.” In: *Physical Review Letters* 107.7 (Aug. 10, 2011). ISSN: 0031-9007, 1079-7114. DOI: 10.1103/PhysRevLett.107.070601.

[205] Jutho Haegeman, J. Ignacio Cirac, Tobias J. Osborne, Iztok Pižorn, Henri Verschelde, and Frank Verstraete. *Supplementary Material for “Time-Dependent Variational Principle for Quantum Lattices”*. May 26, 2011. DOI: 10.48550/arXiv.1103.0936.

[206] Wikimedia Commons. *File:Tangentialvektor.svg — Wikimedia Commons, the free media repository*. 2020. URL: https://commons.wikimedia.org/w/index.php?title=File:Tangentialvektor.svg&oldid=513180893.

[207] W. Wirtinger. “Zur formalen Theorie der Funktionen von mehr komplexen Veränderlichen.” In: *Mathematische Annalen* 97.1 (Dec. 1927), pp. 357–375. DOI: 10.1007/bf01447872.

[208] James Stokes, Josh Izaac, Nathan Killoran, and Giuseppe Carleo. “Quantum Natural Gradient.” In: *Quantum* 4 (May 2020), p. 269. DOI: 10.22331/q-2020-05-25-269.

[209] Ken Kreutz-Delgado. *The Complex Gradient Operator and the CR-Calculus*. 2009. DOI: 10.48550/arXiv.0906.4835.

[210] F. Strocchi. "Complex Coordinates and Quantum Mechanics." In: *Reviews of Modern Physics* 38.1 (Jan. 1966), pp. 36–40. DOI: 10.1103/RevModPhys.38.36.

[211] A.D. McLachlan. "A variational solution of the time-dependent Schrodinger equation." In: *Molecular Physics* 8.1 (Jan. 1964), pp. 39–44. DOI: 10.1080/00268976400100041.

[212] Peter Kramer and Marcos Saraceno, eds. *Geometry of the Time-Dependent Variational Principle in Quantum Mechanics*. Springer Berlin Heidelberg, 1981. DOI: 10.1007/3-540-10579-4.

[213] J. Broeckhove, L. Lathouwers, E. Kesteloot, and P. Van Leuven. "On the equivalence of time-dependent variational principles." In: *Chemical Physics Letters* 149.5-6 (Sept. 1988), pp. 547–550. DOI: 10.1016/0009-2614(88)80380-4.

[214] Marin Bukov, Markus Schmitt, and Maxime Dupont. "Learning the ground state of a non-stoquastic quantum Hamiltonian in a rugged neural network landscape." In: *SciPost Physics* 10.6 (June 2021). DOI: 10.21468/SciPostPhys.10.6.147.

[215] Giuseppe Carleo, Federico Becca, Marco Schiró, and Michele Fabrizio. "Localization and Glassy Dynamics Of Many-Body Quantum Systems." In: *Scientific Reports* 2.1 (Feb. 2012). DOI: 10.1038/srep00243.

[216] Josef Stoer and Roland Bulrisch. *Numerische Mathematik 2*. 5th ed. Springer-Verlag, 2005. ISBN: 978-3-540-23777-8. DOI: 10.1007/b137272.

[217] Lev D. Landau and Evgeny M. Lifshitz. *Quantum Mechanics: Non-relativistic Theory*. Course of Theoretical Physics. Pergamon Press, 1977. ISBN: 978-0-08-020940-1.

[218] Alexander Wietek. *Topological States of Matter in Frustrated Quantum Magnetism*. 2022. DOI: 10.48550/arXiv.2210.03511.

[219] Tom Vieijra and Jannes Nys. "Many-body quantum states with exact conservation of non-Abelian and lattice symmetries through variational Monte Carlo." In: *Physical Review B* 104.4 (July 2021). DOI: 10.1103/PhysRevB.104.045123.

[220] M. Hermanns, I. Kimchi, and J. Knolle. "Physics of the Kitaev Model: Fractionalization, Dynamic Correlations, and Material Connections." In: *Annual Review of Condensed Matter Physics* 9.1 (Mar. 2018), pp. 17–33. DOI: 10.1146/annurev-conmatphys-033117-053934.

[221] Frank Wilczek. "Magnetic Flux, Angular Momentum, and Statistics." In: *Physical Review Letters* 48 (17 Apr. 1982), pp. 1144–1146. DOI: 10.1103/PhysRevLett.48.1144.

[222] Frank Wilczek. "Remarks on Dyons." In: *Physical Review Letters* 48 (17 Apr. 1982), pp. 1146–1149. DOI: 10.1103/PhysRevLett.48.1146.

[223] A.Yu. Kitaev. "Fault-tolerant quantum computation by anyons." In: *Annals of Physics* 303.1 (Jan. 2003), pp. 2–30. DOI: 10.1016/s0003-4916(02)00018-0.

[224] Jiří Chaloupka, George Jackeli, and Giniyat Khaliullin. "Kitaev-Heisenberg Model on a Honeycomb Lattice: Possible Exotic Phases in Iridium Oxides A_2IrO_3." In: *Physical Review Letters* 105.2 (July 2010). DOI: 10.1103/PhysRevLett.105.027204.

[225] Jiří Chaloupka, George Jackeli, and Giniyat Khaliullin. "Zigzag Magnetic Order in the Iridium Oxide Na_2IrO_3." In: *Physical Review Letters* 110.9 (Feb. 2013). DOI: 10.1103/PhysRevLett.110.097204.

[226] Jeffrey G. Rau, Eric Kin-Ho Lee, and Hae-Young Kee. “Generic Spin Model for the Honeycomb Iridates beyond the Kitaev Limit.” In: *Physical Review Letters* 112.7 (Feb. 2014). DOI: 10.1103/PhysRevLett.112.077204.

[227] Adithya Sriram and Martin Claassen. “Light-induced control of magnetic phases in Kitaev quantum magnets.” In: *Physical Review Research* 4.3 (Sept. 2022). DOI: 10.1103/PhysRevResearch.4.l032036.

[228] Emil Vinas Boström, Adithya Sriram, Martin Claassen, and Angel Rubio. *Controlling the magnetic state of the proximate quantum spin liquid α-RuCl$_3$ with an optical cavity.* 2022. DOI: 10.48550/arXiv.2211.07247.

[229] Elliott H. Lieb. “Flux Phase of the Half-Filled Band.” In: *Physical Review Letters* 73.16 (Oct. 1994), pp. 2158–2161. DOI: 10.1103/PhysRevLett.73.2158.

[230] Fabian Zschocke and Matthias Vojta. “Physical states and finite-size effects in Kitaev’s honeycomb model: Bond disorder, spin excitations, and NMR line shape.” In: *Physical Review B* 92.1 (July 2015). DOI: 10.1103/PhysRevB.92.014403.

[231] Fabian Zschocke. “Kitaev Honeycomb Model: Majorana Fermion Representation and Disorder.” PhD thesis. Technische Universität Dresden, 2016. URL: http://nbn-resolving.de/urn:nbn:de:bsz:14-qucosa-205839.

[232] Moyuru Kurita, Youhei Yamaji, Satoshi Morita, and Masatoshi Imada. “Variational Monte Carlo method in the presence of spin-orbit interaction and its application to Kitaev and Kitaev-Heisenberg models.” In: *Physical Review B* 92.3 (July 2015). DOI: 10.1103/PhysRevB.92.035122.

[233] G. Kells, A. T. Bolukbasi, V. Lahtinen, J. K. Slingerland, J. K. Pachos, and J. Vala. “Topological Degeneracy and Vortex Manipulation in Kitaev’s Honeycomb Model.” In: *Physical Review Letters* 101.24 (Dec. 2008). DOI: 10.1103/PhysRevLett.101.240404.

[234] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. *JAX: composable transformations of Python+NumPy programs.* Version 0.3.13. 2018. URL: http://github.com/google/jax.

[235] Roy Frostig, Matthew James Johnson, and Chris Leary. “Compiling machine learning programs via high-level tracing.” In: *SysML Conference 2018.* Feb. 16, 2018. URL: https://mlsys.org/Conferences/doc/2018/146.pdf (visited on 10/19/2022).

[236] Subir Sachdev and Jinwu Ye. “Gapless spin-fluid ground state in a random quantum Heisenberg magnet.” In: *Physical Review Letters* 70.21 (May 1993), pp. 3339–3342. DOI: 10.1103/PhysRevLett.70.3339.

[237] Debanjan Chowdhury, Antoine Georges, Olivier Parcollet, and Subir Sachdev. “Sachdev-Ye-Kitaev Models and beyond: Window into Non-Fermi Liquids.” In: *Reviews of Modern Physics* 94.3 (Sept. 2022), p. 035004. DOI: 10.1103/RevModPhys.94.035004.

[238] Arijit Haldar, Omid Tavakol, and Thomas Scaffidi. “Variational wave functions for Sachdev-Ye-Kitaev models.” In: *Physical Review Research* 3 (2 Apr. 2021), p. 023020. DOI: 10.1103/PhysRevResearch.3.023020.

[239] Joonho Kim, Jaedeok Kim, and Dario Rosa. “Universal effectiveness of high-depth circuits in variational eigenproblems.” In: *Physical Review Research* 3 (2 June 2021), p. 023203. DOI: 10.1103/PhysRevResearch.3.023203.

[240] Dan Sehayek, Anna Golubeva, Michael S. Albergo, Bohdan Kulchytskyy, Giacomo Torlai, and Roger G. Melko. "Learnability scaling of quantum states: Restricted Boltzmann machines." In: *Physical Review B* 100.19 (Nov. 2019). DOI: 10.1103/PhysRevB.100.195125.

[241] Victoria Stodden. "The Legal Framework for Reproducible Scientific Research: Licensing and Copyright." In: *Computing in Science & Engineering* 11.1 (Jan. 2009), pp. 35–40. DOI: 10.1109/mcse.2009.19.

[242] *The Open Source Definition.* Mar. 22, 2007. URL: https://opensource.org/osd.

[243] Katie Malone. "Doing Data Science on the Shoulders of Giants: The Value of Open Source Software for the Data Science Community." In: *Harvard Data Science Review* (Apr. 2020). DOI: 10.1162/99608f92.268cc8e4.

[244] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. "TensorFlow: A System for Large-Scale Machine Learning." In: *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016.* Ed. by Kimberly Keeton and Timothy Roscoe. USENIX Association, 2016, pp. 265–283. URL: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi.

[245] TensorFlow Developers. *TensorFlow.* 2022. DOI: 10.5281/ZENODO.6574269.

[246] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In: *Advances in Neural Information Processing Systems 32.* Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[247] J. Daniel Gezelter. "Open Source and Open Data Should Be Standard Practices." In: *The Journal of Physical Chemistry Letters* 6.7 (Apr. 2015), pp. 1168–1169. DOI: 10.1021/acs.jpclett.5b00285.

[248] Mark Fingerhuth, Tomáš Babej, and Peter Wittek. "Open source software in quantum computing." In: *PLOS ONE* 13.12 (Dec. 2018). Ed. by Leonie Anna Mueck, e0208561. DOI: 10.1371/journal.pone.0208561.

[249] Susi Lehtola and Antti J. Karttunen. "Free and open source software for computational chemistry education." In: *WIREs Computational Molecular Science* 12.5 (Mar. 2022). DOI: 10.1002/wcms.1610.

[250] B Bauer et al. "The ALPS project release 2.0: open source software for strongly correlated systems." In: *Journal of Statistical Mechanics: Theory and Experiment* 2011.05 (May 2011), P05001. DOI: 10.1088/1742-5468/2011/05/p05001.

[251] Markus Wallerberger, Sergei Iskakov, Alexander Gaenko, Joseph Kleinhenz, Igor Krivenko, Ryan Levy, Jia Li, Hiroshi Shinaoka, Synge Todo, Tianran Chen, Xi Chen, James P. F. LeBlanc, Joseph E. Paki, Hanna Terletska, Matthias Troyer, and Emanuel Gull. *Updated Core Libraries of the ALPS Project.* 2018. DOI: 10.48550/arXiv.1811.08331.

[252] Olivier Parcollet, Michel Ferrero, Thomas Ayral, Hartmut Hafermann, Igor Krivenko, Laura Messio, and Priyanka Seth. "TRIQS: A toolbox for research on interacting quantum systems." In: *Computer Physics Communications* 196 (Nov. 2015), pp. 398–415. DOI: 10.1016/j.cpc.2015.04.023.

[253] Matthew Fishman, Steven R. White, and E. Miles Stoudenmire. "The ITensor Software Library for Tensor Network Calculations." In: *SciPost Physics Codebases* (2022), p. 4. DOI: 10.21468/SciPostPhysCodeb.4.

[254] *NetKet: Machine Learning for Many-Body Quantum Systems.* Apr. 30, 2018. URL: https://web.archive.org/web/20180428152223/http://www.netket.org/.

[255] Wenzel Jakob, Jason Rhinelander, and Dean Moldovan. *pybind11 – Seamless operability between C++11 and Python.* 2017. URL: https://github.com/pybind/pybind11.

[256] Phillip Weinberg and Marin Bukov. "QuSpin: a Python package for dynamics and exact diagonalisation of quantum many body systems part I: spin chains." In: *SciPost Physics* 2.1 (Feb. 2017). DOI: 10.21468/SciPostPhys.2.1.003.

[257] Hans Fangohr, Thomas Kluyver, and Massimo DiPierro. "Jupyter in Computational Science." In: *Computing in Science & Engineering* 23.2 (Mar. 1, 2021), pp. 5–6. ISSN: 1521-9615, 1558-366X. DOI: 10.1109/MCSE.2021.3059494.

[258] Brian E. Granger and Fernando Perez. "Jupyter: Thinking and Storytelling With Code and Data." In: *Computing in Science & Engineering* 23.2 (Mar. 1, 2021), pp. 7–14. ISSN: 1521-9615, 1558-366X. DOI: 10.1109/MCSE.2021.3059263.

[259] Atilim Gunes Baydin and Barak A. Pearlmutter. *Automatic Differentiation of Algorithms for Machine Learning.* 2014. DOI: 10.48550/arXiv.1404.7456.

[260] *XLA: Optimizing Compiler for Machine Learning.* Aug. 18, 2022. URL: https://www.tensorflow.org/xla.

[261] Lisandro Dalcin and Yao-Lung L. Fang. "mpi4py: Status Update After 12 Years of Development." In: *Computing in Science & Engineering* 23.4 (July 2021), pp. 47–54. DOI: 10.1109/mcse.2021.3083216.

[262] Dion Häfner and Filippo Vicentini. "mpi4jax: Zero-copy MPI communication of JAX arrays." In: *Journal of Open Source Software* 6.65 (2021), p. 3419. DOI: 10.21105/joss.03419.

[263] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. "Automatic Differentiation in Machine Learning: a Survey." In: *J. Mach. Learn. Res.* 18 (2017), 153:1–153:43. URL: http://jmlr.org/papers/v18/17-468.html.

List of acronyms

CNN convolutional neural network

DMRG density matrix renormalization group

ED exact diagonalization

FFNN feed-forward neural network

GCNN group-convolutional neural network

HKM honeycomb Kitaev model
HPC high-performance computing

irrep irreducible representation

JIT just-in-time (compilation)

MC-CLT Markov chain central limit theorem
MCMC Markov chain Monte Carlo
MCSE Monte Carlo standard error
MPI message passing interface
MPS matrix-product state

NAQS neural autoregressive quantum state
NDO neural density operator
NGD natural gradient descent
NQS neural quantum state

QFM quantum Fisher matrix
QGT quantum geometric tensor
QMC quantum Monte Carlo
QSL quantum spin liquid
QST quantum state tomography

RBM restricted Boltzmann machine

SR stochastic reconfiguration
SYK Sachdev-Ye-Kitaev (model)

t-VMC time-dependent variational Monte Carlo
TDSE time-dependent Schrödinger equation
TDVP time-dependent variational principle
TNS tensor-network state

VMC variational Monte Carlo
VP variational principle

XLA accelerated linear algebra (compiler)