QuCumber: wavefunction reconstruction with neural networks

Matthew J. S. Beach[1,2], Isaac De Vlugt[1], Anna Golubeva[1,2], Patrick Huembeli[2,3], Roger G. Melko[1,2*], Ejaaz Merali[1], Giacomo Torlai[1,2],

[1] Department of Physics and Astronomy, University of Waterloo, Ontario N2L 3G1, Canada
[2] Perimeter Institute for Theoretical Physics, Waterloo, Ontario N2L 2Y5, Canada
[3] ICFO-Institut de Ciencies Fotoniques, Barcelona Institute of Science and Technology, 08860 Castelldefels (Barcelona), Spain
* rgmelko@uwaterloo.ca

October 3, 2018

Abstract

In this post, we present QuCumber, an open-source Python package that implements a restricted Boltzmann machine to reconstruct unknown quantum states from experimental measurements. The effectiveness of modern machine learning algorithms in compressing high-dimensional data allows neural networks to discover compact representations of a quantum state, and reconstruct traditionally challenging physical properties, not directly accessible in current experiments. We show how to use QuCumber to reconstruct wavefunctions with and without a phase structure and deploy the trained network to estimate expectation values of various physical observables.

Contents

---

1 Introduction

The current advances in designing and building quantum technologies, as well as in the reliable control of synthetic quantum matter, is leading to a new wave of quantum hardware, where highly pure quantum states are routinely prepared in laboratories. Notwithstanding the presence of noise and decoherence, new exciting possibilities have been recently opened by this class of "noisy" quantum hardware. Notable examples are the quantum-assisted simulation of small molecules and quantum magnets with superconducting hardware [1,2], and the preparation of ground states breaking different spacial symmetries using a cold Rydberg atom quantum simulator [3]. With the growing number of controlled quantum degrees of freedom (e.g. qubits, atoms, etc.), reliable and scalable classical algorithms are required to aid in the analysis and verification of the quantum devices. This is necessary to extract physical observables otherwise inaccessible from experimental measurements, as well as identifying the source of noise to provide direct feedback for improving the hardware quality. However, traditional approaches for reconstructing unknown quantum states from a set of measurements – quantum state reconstruction (QSR) – suffer the exponential overhead that is typical of quantum many-body systems and thus remain suited for systems with a small number of constituents.

Recently, an alternative path to QSR was put forward, based on modern machine learning (ML) techniques [4, 5]. This approach relies on a powerful generative model called a *restricted Boltzmann machine* (RBM) [6], a stochastic neural network with two layers of binary units. The visible layer $\boldsymbol{v} = (v_1, \ldots, v_N)$ describes the physical degrees of freedom, and a hidden layer $\boldsymbol{h} = (h_1, \ldots, h_{N_h})$, used to capture high-order correlation between the visible units. Given the set of neural network parameters $\boldsymbol{\lambda}$, the RBM defines a probabilistic model described by the parametric distribution $p_{\boldsymbol{\lambda}}(\boldsymbol{v})$ (see Glossary). RBMs have been widely used in the ML community for the pre-training of deep neural networks [7] and compressing high-dimensional data into lower-dimensional representations [8]. More recently, RBMs have been adopted by the physics community in the context of representing both classical and quantum many-body states [9,10], and are currently being investigated in the context of their representational power [11] their connection with tensor network states [12] and the renormalization group [13].

In this post, we present QuCumber, a *quantum calculator used for many-body eigenstates reconstruction*. QuCumber is an open-source Python package that implements neural-network QSR of many-body wavefunctions from measurement data, directly accessible in most experimental setups. Some examples are magnetic spin projections, orbital occupation number, polarization of photons and the logical state of qubits. Given a training dataset of measurements, QuCumber discovers the most likely quantum state compatible with the measurements by finding the optimal set of parameters $\boldsymbol{\lambda}$ of an RBM. A properly trained RBM is an approximation of the unknown quantum state underlying the data and can then be used to calculate various physical observables of interest, such as entanglement entropy.

This post is organized as follows. In Section 2, we introduce the reconstruction technique for the case of positive wavefunctions, where all coefficients are (or can be transformed to be) real and positive. We discuss the required format for input data, as well as training of the RBM and the reconstruction of both diagonal and off-diagonal observables. In Section 3, we consider the more general case of a complex-valued wavefunction. We show the general strategy to extract the phase structure from raw data by performing appropriate unitary rotations on the state before measurements. We then show a practical reconstruction of an entangled state of two qubits. A glossary of useful terms and equations can be found at the end of the post in Section A.

2 Positive wavefunctions

We begin by presenting the application of QuCumber to reconstruct many-body quantum states described by wavefunctions $|\Psi\rangle$ with positive coefficients $\Psi(\mathbf{x}) = \langle \mathbf{x}|\Psi\rangle \geq 0$, where $|\mathbf{x}\rangle = |\mathrm{x}_1, \ldots, \mathrm{x}_N\rangle$ is a reference basis for the Hilbert space of N quantum degrees of freedom. The neural-network QSR requires raw data $\mathcal{D} = (\mathbf{x}_1, \mathbf{x}_2, \ldots)$ generated through projective measurements of the state $|\Psi\rangle$ in the reference basis. These measurements adhere [??] to the probability distribution given by the Born rule, $P(\mathbf{x}) = |\Psi(\mathbf{x})|^2$. Since the wavefunction is strictly positive, the quantum state is completely characterized by the measurement distribution, i.e. $\Psi(\mathbf{x}) = \sqrt{P(\mathbf{x})}$.

The positivity of the wavefunction allows a simple and natural connection between quantum states and classical probabilistic models. Specifically, QuCumber employs the probability distribution $p_{\boldsymbol{\lambda}}(\mathbf{x})$ of an RBM to capture the distribution $P(\mathbf{x})$ underlying the measurement data. Using contrastive divergence (CD) [14], QuCumber trains the RBM to discover a set of parameters $\boldsymbol{\lambda}^*$ that minimize the Kullback-Leibler (KL) divergence between the two distributions. Upon successful training ($p_{\boldsymbol{\lambda}^*}(\mathbf{x}) \sim P(\mathbf{x})$), we obtain a faithful representation of the target quantum state,

$$\psi_{\boldsymbol{\lambda}^*}(\mathbf{x}) = \sqrt{p_{\boldsymbol{\lambda}^*}(\mathbf{x})} \simeq \Psi(\mathbf{x})\,. \tag{1}$$

In the following Sections, we demonstrate QuCumber for the reconstruction of the ground-state wavefunction of the one-dimensional transverse-field Ising model (TFIM). The spin Hamiltonian is

$$\hat{H} = -J\sum_i \hat{\sigma}_i^z \hat{\sigma}_{i+1}^z - h\sum_i \hat{\sigma}_i^x\,, \tag{2}$$

where $\sigma_i^{x/z}$ are spin-1/2 Pauli matrices on site i and we assume open boundary conditions. For this example, we consider a chain with $N = 10$ spins at the quantum critical point $J = h = 1$.

2.1 Setup

Rather than experimental data, we use a set of synthetic measurements generated by sampling the exact ground-state wavefunction. Given the small size of the system, the ground state, $|\Psi\rangle$, can be found with exact diagonalization. The training dataset, $\mathcal{D}$, is generated by sampling the distribution $P(\boldsymbol{\sigma}^z) = |\Psi(\boldsymbol{\sigma}^z)|^2$, obtaining a sequence of $N_S = 10^5$ independent spin projections in the $\boldsymbol{\sigma}^z$ basis [1] . Each data point in $\mathcal{D}$ consists of an array $\boldsymbol{\sigma}_j^z = (\sigma_1^z, \ldots, \sigma_N^z)$ with shape `(N,)` and should be passed to QuCumber as a numpy array or torch tensor. For example, $\boldsymbol{\sigma}_j^z =$ `np.array([1,0,1,1,0,1,0,0,0,1])`, where we use $\sigma_j^z = 0, 1$ to represent a spin-down and spin-up state respectively. Therefore, the entire input data set is contained into an array with shape `(N_S, N)`.

Aside from the training data, QuCumber also allows us to import an example wavefunction. This can be useful for monitoring the quality of the reconstruction during training. In our case, we import the exact ground state to evaluate the fidelity between the QSR $\psi_{\boldsymbol{\lambda}^*}(\mathbf{x})$ and the exact wavefunction $\Psi(\boldsymbol{x})$. The training dataset, `train_data`, and the exact ground state, `true_psi`, are loaded with the data loading utility as follows:

```
import qucumber.utils.data as data
train_path = "tfim1d_data.txt"
psi_path = "tfim1d_psi.txt"
train_data, true_psi = data.load_data(train_path, psi_path)
```

[1] The training dataset can be download from `https://github.com/PIQuIL/QuCumber/blob/master/examples/Tutorial1_TrainPosRealWavefunction/tfim1d_data`

If `psi_path` is not provided, QuCumber will load only the training data.

Next, we initialize an RBM quantum state $\psi_{\boldsymbol{\lambda}}(\boldsymbol{\sigma}^z)$ with random weights and zero biases using the constructor `PositiveWavefunction`:

```
from qucumber.nn_states import PositiveWavefunction
state = PositiveWavefunction(num_visible=10, num_hidden=10)
```

The number of visible units (`num_visible`) must be equal to the number N of spins, while the number of hidden units (`num_hidden`) can be adjusted to systematically increase the representational power of the RBM.

The quality of the reconstruction will depend on the structure underlying the specific quantum staste and the ratio of visible to hidden units, $\alpha =$ `num_hidden`/`num_visible`. In practice, we find that $\alpha = 1$ typically leads to good approximations of positive wavefunctions [4]. However, in the general case, the value of α required for a given wavefunction reconstruction should be explored and adjusted by the user.

2.2 Training

Once an appropriate representation of the quantum state has been defined, QuCumber trains the RBM through the function `PositiveWavefunction.fit`. Several input parameters need to be provided aside from the training dataset (`train_data`). These include the number of training iterations (`epochs`), the number of samples used for the positive/negative phase of CD (`pos_batch_size`/`neg_batch_size`), the learning rate (`lr`) and the number of sampling steps in the negative phase of CD (`k`) [Can we change this to k_{CD} in the code as well?]. The last argument (`callbacks`) allows the user to pass a set of functions to be evaluated during training.

As an example of a callback, we show the `MetricEvaluator`, which evaluates a function `log_every` epochs during training. Given the small system size and the knowledge of the true ground state, we can evaluate the fidelity between the RBM state and the true ground-state wavefunction (`true_psi`). Similarly, we can calculate the KL divergence between the RBM distribution $p_{\boldsymbol{\lambda}}(\boldsymbol{x})$, and the data distribution $P(\boldsymbol{x})$, which should approach zero for a properly trained RBM. For the current example, we monitor the fidelity and KL divergence (defined in `qucumber.utils.training_statistics`):

```
from qucumber.callbacks import MetricEvaluator
import qucumber.utils.training_statistics as ts
log_every = 10
space = state.generate_hilbert_space(10)
callbacks = [
    MetricEvaluator(
        log_every,
        {"Fidelity": ts.fidelity, "KL": ts.KL},
    target_psi=true_psi,
    space=space,
    verbose=True
    )
]
```

With `verbose=True`, the program will print the epoch number and all callbacks every `log_every` epochs. Now that the metrics to monitor during training have been chosen, we can invoke the optimization with the `fit` function of `PositiveWavefunction`.

```
state.fit(
    train_data,
    epochs=400,
    pos_batch_size=100
    neg_batch_size=100,
```

```
    lr=0.01
    k=5,
    callbacks=callbacks,
)
```

Figure 1 shows the convergence of the fidelity and KL divergence during training. The convergence time will, in general, depend on the choice of hyperparameters.

The network parameters $\boldsymbol{\lambda}$, together with the callbacks, can be saved (or loaded) to a file:

```
state.save(
    "filename.pt",
    metadata={
        "fidelity": callbacks[0].Fidelity,
        "KL": callbacks[0].KL
    },
)
state.load("filename.pt")
```

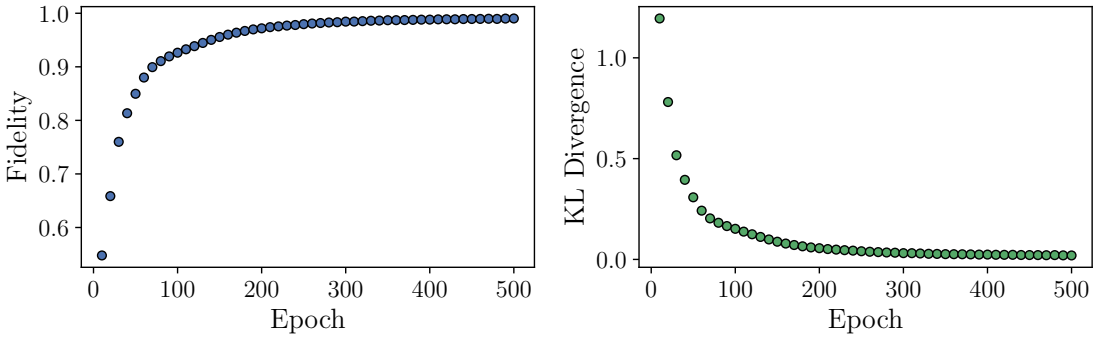

Figure 1: The fidelity (left) and the KL divergence (right) during training for the reconstruction of the ground state of the one-dimensional TFIM.

In this Section, we demonstrated to most basic aspects of QuCumber regarding training a model and verifying its accuracy. While in this example the evaluation required the knowledge of the exact ground state and the calculation of the KL divergence is feasible only for small system sizes, we point out that QuCumber is capable of carrying out the reconstruction of larger systems. In this case however, users should rely on other estimators to evaluate the training, such as physical observables (magnetization, energy, etc). In the following Section, we show how to compute diagonal and off-diagonal observables in QuCumber.

2.3 Reconstruction of physical observables

In this Section, we discuss how to calculate the average value of a generic physical observable $\hat{\mathcal{O}}$ from a trained RBM. We start with the case of observables that are diagonal in the reference basis where the RBM was trained. We then show the more general case of off-diagonal observables.

2.3.1 Diagonal observables

We begin by considering an observable with only diagonal matrix elements, $\langle \boldsymbol{\sigma} | \hat{\mathcal{O}} | \boldsymbol{\sigma}' \rangle = \mathcal{O}_{\boldsymbol{\sigma}} \delta_{\boldsymbol{\sigma}\boldsymbol{\sigma}'}$ where for convenience we take $\boldsymbol{\sigma} = \boldsymbol{\sigma}^z$ unless otherwise stated. The expectation

value of $\hat{\mathcal{O}}$ is given by

$$\langle\hat{\mathcal{O}}\rangle = \frac{1}{\sum_{\boldsymbol{\sigma}}|\psi_{\boldsymbol{\lambda}}(\boldsymbol{\sigma})|^2}\sum_{\boldsymbol{\sigma}}\mathcal{O}_{\boldsymbol{\sigma}}|\psi_{\boldsymbol{\lambda}}(\boldsymbol{\sigma})|^2\,. \tag{3}$$

In this case, the exponential sum can be approximated by a Monte Carlo average

$$\langle\hat{\mathcal{O}}\rangle \approx \frac{1}{N_{\mathrm{MC}}}\sum_{k=1}^{N_{\mathrm{MC}}}\mathcal{O}_{\boldsymbol{\sigma}_k}\,, \tag{4}$$

where the spin configurations $\boldsymbol{\sigma}_k$ are sampled from the RBM distribution $p_{\boldsymbol{\lambda}}(\boldsymbol{\sigma})$. This process is particularly efficient given the bipartite structure of the network which allows the use of block Gibbs sampling.

A simple example for the TFIM is the average magnetization per spin, $M = \sum_j\langle\hat{\sigma}_j\rangle/N$, which can be calculated directly on the spin configuration sampled by the RBM (i.e., the state of the visible layer). The visible samples are obtained with the `sample` function of the RBM state object:

```
samples = state.sample(num_samples=1000, k=10)
```

which inputs the total number of samples (`num_samples`) and the number of iterations (`k`) of a block Gibbs step. Once these samples are obtained, the magnetization can be calculated simply as

```
magnetization = samples.mul(2.0).sub(1.0).mean(1).abs().mean()
```

where we converted the binary samples of the RBM back into ±1 spins.

2.3.2 Off-diagonal observables

We turn now to the case of off-diagonal observables, where the expectation value assumes the following form

$$\langle\hat{\mathcal{O}}\rangle = \frac{1}{\sum_{\boldsymbol{\sigma}}|\psi_{\boldsymbol{\lambda}}(\boldsymbol{\sigma})|^2}\sum_{\boldsymbol{\sigma}\boldsymbol{\sigma}'}\psi_{\boldsymbol{\lambda}}(\boldsymbol{\sigma})\psi_{\boldsymbol{\lambda}}(\boldsymbol{\sigma}')\mathcal{O}_{\boldsymbol{\sigma}\boldsymbol{\sigma}'}\,. \tag{5}$$

This expression can once again be approximated with a Monte Carlo average

$$\langle\hat{\mathcal{O}}\rangle \approx \frac{1}{N_{\mathrm{MC}}}\sum_{k=1}^{N_{\mathrm{MC}}}\mathcal{O}^{[L]}_{\boldsymbol{\sigma}_k} \tag{6}$$

of the so-called *local estimator* of the observable:

$$\mathcal{O}^{[L]}_{\boldsymbol{\sigma}_k} = \sum_{\boldsymbol{\sigma}'}\frac{\psi_{\boldsymbol{\lambda}}(\boldsymbol{\sigma}')}{\psi_{\boldsymbol{\lambda}}(\boldsymbol{\sigma})}\mathcal{O}_{\boldsymbol{\sigma}\boldsymbol{\sigma}'}\,. \tag{7}$$

As long as the matrix representation $\mathcal{O}_{\boldsymbol{\sigma}\boldsymbol{\sigma}'}$ is sufficiently sparse in the reference basis, the summation can be evaluated efficiently. As an example, we consider the specific case of the transverse magnetization for the j-th spin, $M^x_j = \langle\hat{\sigma}^x_j\rangle$, with matrix elements

$$\langle\boldsymbol{\sigma}|\hat{\sigma}^x_j|\boldsymbol{\sigma}'\rangle = \delta_{\sigma'_j,1-\sigma_j}\prod_{i\neq j}\delta_{\sigma'_i,\sigma_j}\,. \tag{8}$$

Therefore, the expectation values reduces to the Monte Carlo average of the local observable

$$M^{x[L]}_j = \frac{\psi_{\boldsymbol{\lambda}}(\sigma_1,\ldots,1-\sigma_j,\ldots,\sigma_N)}{\psi_{\boldsymbol{\lambda}}(\sigma_1,\ldots,\sigma_j,\ldots,\sigma_N)}\,. \tag{9}$$

evaluated on spin configurations $\boldsymbol{\sigma}_k$ sampled from the RBM distribution $p_{\boldsymbol{\lambda}}(\boldsymbol{\sigma})$.

QuCumber provides an interface for sampling off-diagonal observables in the `Observable` class. Thorough examples can are available in the tutorial section in the documentation[2]. As an example, σ^x can be written as an observable class with

```
class SigmaX(Observable):

    def apply(self, nn_state, samples):
        psi = nn_state.psi(samples)
        psi_ratio_sum = torch.zeros_like(psi)

        for i in range(samples.shape[-1]):  # sum over spin sites
            flip_spin(i, samples)  # flip the spin at site i
            # compute ratio of psi_(-i) / psi and add it to the running sum
            psi_ratio = nn_state.psi(samples)
            psi_ratio = cplx.elementwise_division(psi_ratio, psi)
            psi_ratio_sum.add_(psi_ratio)
            flip_spin(i, samples)  # flip it back

        # take real part (imaginary part should be approximately zero)
        # and divide by number of spins
        return psi_ratio_sum[0].div_(samples.shape[-1])
```

The value of the observable is computed with

```
SigmaX.statistics_from_samples(state, samples)
```

Similarly, the user can define other observables like σ^z or the energy.

The reconstruction of two magnetic observables for the TFIM is shown in Fig. 2, where a different RBM was trained for each value of the transverse field h. In the left plot, we show the average longitudinal magnetization per site, which can be calculated directly from the configurations sampled by the RBM. In the right plot, we show the off-diagonal observable of transverse magnetization. In both cases, QuCumber successfully discovers a set of parameters $\boldsymbol{\lambda}^*$ that correctly approximates the ground-state wavefunction underlying the data.

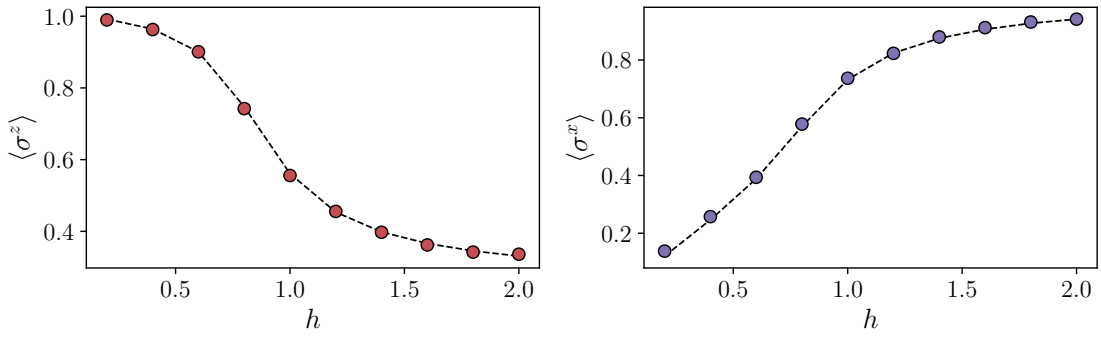

Figure 2: Reconstruction of the magnetic observables for the TFIM chain with $N = 10$ spins. We show the average longitudinal (left) and transverse (right) magnetization per site obtained by sampling a trained RBM. The dashed line denotes the results from exact diagonalization.

[2]The observables tutorial can be found at `https://qucumber.readthedocs.io/en/stable/_examples/Tutorial3_DataGeneration_CalculateObservables/tutorial_sampling_observables.html`

3 Complex wavefunctions

For positive wavefunctions, the probability distribution underlying the outcomes of projective measurement in the reference basis already contains all the information about the unknown quantum state. In this Section, we consider a target quantum state with a non-trivial sign/phase structure. This means that the wavefunction coefficients in the reference basis can be of different signs, as well as complex-valued, $\Psi(\boldsymbol{\sigma}) = \Phi(\boldsymbol{\sigma})e^{i\theta(\boldsymbol{\sigma})}$. We first need to generalize the RBM representation of the quantum state to capture generic complex wavefunctions. To this end, we introduce an additional RBM with network parameters $\boldsymbol{\mu}$ and define the total RBM state as:

$$\psi_{\boldsymbol{\lambda}\boldsymbol{\mu}}(\boldsymbol{\sigma}) = \sqrt{p_{\boldsymbol{\lambda}}(\boldsymbol{\sigma})}e^{i\phi_{\boldsymbol{\mu}}(\boldsymbol{\sigma})/2} \tag{10}$$

where we choose $\phi_{\boldsymbol{\mu}}(\boldsymbol{\sigma}) = \log(p_{\boldsymbol{\mu}}(\boldsymbol{\sigma}))$ [4]. Furthermore, the reconstruction requires a different type of measurement settings. In fact, projective measurements in the reference basis do not convey any information on the phases $\theta(\boldsymbol{\sigma})$, since $P(\boldsymbol{\sigma}) = |\Psi(\boldsymbol{\sigma})|^2 = \Phi^2(\boldsymbol{\sigma})$.

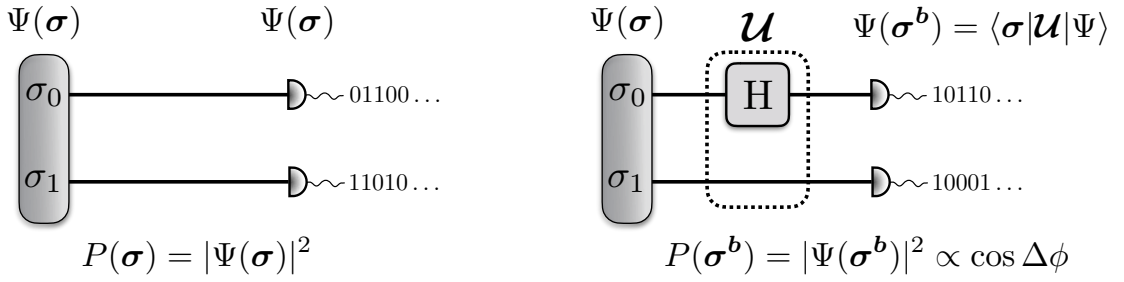

Figure 3: Unitary rotations for two qubits. (left) Measurements on the reference basis. (right) Measurement in the rotated basis. The unitary rotation (the Hadamard gate on qubit σ_0) is applied after state preparation and before the projective measurement.

The general strategy to learn a phase structure is to apply a unitary transformation $\boldsymbol{\mathcal{U}}$ to the state $|\Psi\rangle$ (before the measurements) such that the resulting measurement distribution $P'(\boldsymbol{\sigma}) = |\Psi'(\boldsymbol{\sigma})|^2$ of the rotated state $\Psi'(\boldsymbol{\sigma}) = \langle\boldsymbol{\sigma}|\,\boldsymbol{\mathcal{U}}\,|\Psi\rangle$ contains fingerprints of the phases $\theta(\boldsymbol{\sigma})$ (Fig. 3). In general, different rotations must be independently applied to gain full information on the phase structure. We make the assumption of a tensor product structure of the rotations, $\boldsymbol{\mathcal{U}} = \bigotimes_{j=1}^{N}\hat{\mathcal{U}}_j$. This is equivalent to a local change of basis from $|\boldsymbol{\sigma}\rangle$ to $\{|\boldsymbol{\sigma}^{\boldsymbol{b}}\rangle = |\sigma_1^{b_1},\ldots,\sigma_N^{b_N}\rangle\}$, where the vector $\boldsymbol{b}$ identifies the local basis b_j for each site j. The target wavefunction in the new basis is given by

$$\begin{aligned}\Psi(\boldsymbol{\sigma}^{\boldsymbol{b}}) = \langle\boldsymbol{\sigma}^{\boldsymbol{b}}|\Psi\rangle &= \sum_{\boldsymbol{\sigma}}\langle\boldsymbol{\sigma}^{\boldsymbol{b}}|\boldsymbol{\sigma}\rangle\langle\boldsymbol{\sigma}|\Psi\rangle \\ &= \sum_{\boldsymbol{\sigma}}\mathcal{U}(\boldsymbol{\sigma}^{\boldsymbol{b}},\boldsymbol{\sigma})\Psi(\boldsymbol{\sigma})\,,\end{aligned} \tag{11}$$

and the resulting measurement distribution is

$$P_{\boldsymbol{b}}(\boldsymbol{\sigma}^{\boldsymbol{b}}) = \left|\sum_{\boldsymbol{\sigma}}\mathcal{U}(\boldsymbol{\sigma}^{\boldsymbol{b}},\boldsymbol{\sigma})\Psi(\boldsymbol{\sigma})\right|^2 . \tag{12}$$

To clarify the procedure, let us consider the simple example of a quantum state of two qubits:

$$|\Psi\rangle = \sum_{\sigma_0,\sigma_1}\Phi_{\sigma_0\sigma_1}e^{i\theta_{\sigma_0\sigma_1}}|\sigma_0\sigma_1\rangle\,, \tag{13}$$

and rotation $\boldsymbol{\mathcal{U}} = \hat{\mathrm{H}}_0 \otimes \hat{\mathcal{I}}_1$, where $\hat{\mathcal{I}}$ is the identity operator and

$$\hat{\mathrm{H}} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{14}$$

is called the *Hadamard gate*. This transformation is equivalent to rotating the qubit σ_0 from the reference (σ_0^z) basis the the σ_0^x basis. A straightforward calculation leads to the following probability distribution of the projective measurement in the new basis $|\sigma_0^x, \sigma_1\rangle$:

$$P_{\boldsymbol{b}}(\sigma_0^x, \sigma_1) = \frac{\Phi_{0\sigma_1}^2 + \Phi_{1\sigma_1}^2}{4} + \frac{1 - 2\sigma_0^x}{2} \Phi_{0\sigma_1} \Phi_{1\sigma_1} \cos(\Delta\theta) \,, \tag{15}$$

where $\Delta\theta = \theta_{0\sigma_1} - \theta_{1\sigma_1}$. Therefore, the statistics collected by measuring in this basis implicitly contains partial information on the phases. To obtain the full phases structure, additional transformations are required, one example being the rotation from the reference basis to the σ_j^y local basis, realized by the elementary gate

$$\hat{\mathrm{K}}_j = \langle \sigma_j^y | \sigma_j^z \rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -i \\ 1 & i \end{bmatrix} \,. \tag{16}$$

3.1 Setup

We now proceed to use QuCumber to reconstruct a complex-valued wavefunction. For simplicity, we restrict ourselves to two qubits and consider the general case of a quantum state with random amplitudes $\Phi_{\sigma_0\sigma_1}$ and random phases $\theta_{\sigma_0\sigma_1}$. This example is available in the online tutorial. [3] We begin by importing the required packages:

```
from qucumber.nn_states import ComplexWavefunction
import qucumber.utils.unitaries as unitaries
import qucumber.utils.cplx as cplx
```

Since we are dealing with a complex wavefunction, we load the corresponding module `ComplexWavefunction` to build the RBM quantum state $\psi_{\boldsymbol{\lambda\mu}}(\boldsymbol{\sigma})$. Furthermore, the following additional utility modules are required: the `utils.cplx` backend for complex algebra, and the `utils.unitaries` module which contains a set of elementary local rotations. By default, the set of unitaries include rotation to the σ^x and σ^y local basis implemented by the $\hat{\mathrm{H}}$ and $\hat{\mathrm{K}}$ gates respectively.

We continue by loading the data in QuCumber, which is done using the `load_data` function of the data utility:

```
train_path = "qubits_train.txt"
train_bases_path = "qubits_train_bases.txt"
psi_path = "qubits_psi.txt"
bases_path = "qubits_bases.txt"

train_samples, true_psi, train_bases, bases = data.load_data(
  train_path, psi_path, train_bases_path, bases_path
)
```

As before, we may load the true target wavefunction from `qubits_psi.txt`, which can be used to calculate the fidelity and KL divergence. In contrast with the positive case, we now have measurements performed in different bases. Therefore, the training data consists of an array of qubits projections $(\sigma_0^{b_0}, \sigma_1^{b_1})$ in `qubits_train_samples.txt`, together with the corresponding bases (b_0, b_1) where the measurement was taken, in `qubits_train_bases.txt`.

[3] The tutorial for complex wavefunction can be found at `https://qucumber.readthedocs.io/en/stable/_examples/Tutorial2_TrainComplexWavefunction/tutorial_qubits.html`

Finally, QuCumber loads the set of all the bases appearing the in training dataset, stored in `qubits_bases.txt`. This is required to properly configure the various elementary unitary rotations that need to be applied to the RBM state during the training. For this example, we generated measurements in the following bases:

$$(b_0, b_1) = (\mathrm{z}, \mathrm{z})\ ,\ (\mathrm{x}, \mathrm{z})\ ,\ (\mathrm{z}, \mathrm{x})\ ,\ (\mathrm{y}, \mathrm{z})\ ,\ (\mathrm{z}, \mathrm{y}) \tag{17}$$

Finally, before the training, we initialize the set of unitary rotations and create the RBM state object. In the case of the provided dataset, the unitaries required are the well-known $\hat{\mathrm{H}}$ and $\hat{\mathrm{K}}$ gates. The dictionary needed can be created with `unitaries.create_dict()`. By default, when `unitaries.create_dict()` is called, it will contain the identity and the $\hat{\mathrm{H}}$ and $\hat{\mathrm{K}}$ gates with the keys `Z`, `Y`, and `X` respectively. It is possible to add additional gates by specifying them as

```
U = torch.tensor([[re_part], [im_part]], dtype=torch.double)
unitary_dict = unitaries.create_dict(unitary_name="U")
```

where `re_part`, `im_part`, and `U` are to be specified by the user.

We then initialize the complex RBM object with

```
state = ComplexWavefunction(
    num_visible=2 num_hidden=2, unitary_dict=unitary_dict
)
```

The key difference between positive and complex wavefunction reconstruction is the requirement of additional measurements in different basis. Despite this, loading the data, initializing models, and training the RBMs are all very similar to the positive case.

3.2 Training

The training procedure is somewhat similar to the case of a positive wavefunction, where QuCumber optimized the parameters $\boldsymbol{\lambda}$ to minimize the KL divergence between the data and the RBM distribution. When measuring in multiple bases, the optimization now runs over the set of parameters $(\boldsymbol{\lambda}, \boldsymbol{\mu})$ and minimizes the sum of KL divergences between the data distribution $P(\boldsymbol{\sigma}^{\boldsymbol{b}})$ and the RBM distribution $|\psi_{\boldsymbol{\lambda\mu}}(\boldsymbol{\sigma}^{\boldsymbol{b}})|^2$ for each bases $\boldsymbol{b}$ appearing in the training dataset. For example, if a given training sample is measured in the basis (x, z), QuCumber applies the appropriate unitary rotation $\boldsymbol{\mathcal{U}} = \hat{\mathrm{H}}_0 \otimes \hat{\mathcal{I}}_1$ to the RBM state before collecting the gradient signal.

Similar to the case of positive wavefunction, we generate the Hilbert space (to compute fidelity and KL divergence) and initialize the callbacks

```
state.space = nn_state.generate_hilbert_space(nv)
callbacks = [
  MetricEvaluator(
    log_every,
    {"Fidelity": ts.fidelity, "KL": ts.KL},
    target_psi=true_psi,
    bases=bases,
    verbose=True,
    space=state.space,
  )
]
```

The training is carried out by calling the `fit` function of `ComplexWavefunction`, given the set of hyper-parameters

```
state.fit(
    train_samples,
```

```
    epochs=200,
    pos_batch_size=10,
    neg_batch_size=10,
    lr=0.05,
    k=5,
    input_bases=train_bases,
    callbacks=callbacks,
)
```

In Fig. 4 we show the total KL divergence and the fidelity with the true two-qubit state during training.

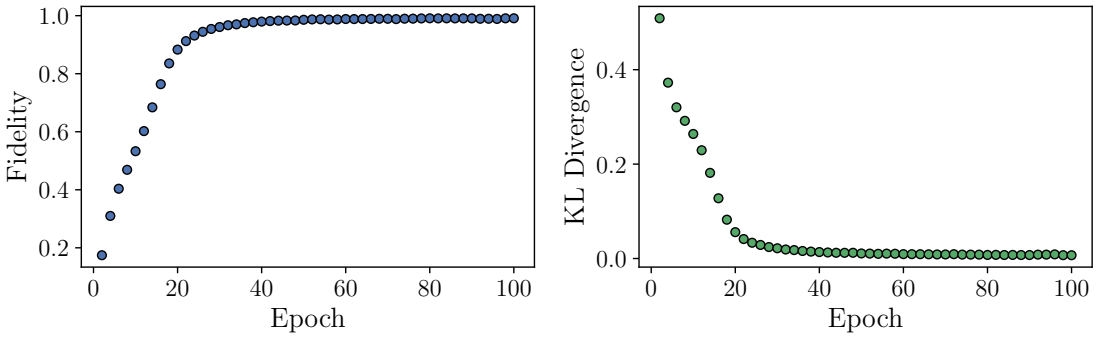

Figure 4: Training a complex RBM with QuCumber on random two-qubit data. We show the fidelity (left), and KL divergence (right), as a function of the training epochs.

After successfully training a QuCumber model, we can once again compute generic observables in the reference basis. [SOMETHING ABOUT OBSERVABLES?!?!?!?!?!???! [CONCLUDING PARAGRAPH??]

4 Conclusion

TODO

Acknowledgements

We acknowledge G. Carleo, J. Carrasquilla and L. Hayward Sierens for stimulating discussions. We thank J. Matlock and the Perimeter Institute for Theoretical Physics for the continuing support of PIQuIL.

Author contributions Authors are listed alphabetically. For an updated record of individual contributions, consult the repository at `https://github.com/PIQuIL/QuCumber/graphs/contributors`.

Funding information This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), the Canada Research Chair program, the Perimeter Institute for Theoretical Physics, and the National Science Foundation under Grant No. NSF PHY-1125915. We also gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research. Research at Perimeter Institute is supported by the Government of Canada through Industry

Canada and by the Province of Ontario through the Ministry of Research & Innovation. P. H. acknowledges support from ICFOstepstone, funded by the Marie Sklodowska-Curie Co-funding of regional, national and international programmes (GA665884) of the European Commission, as well as by the Severo Ochoa 2016-2019' program at ICFO (SEV-2015-0522), funded by the Spanish Ministry of Economy, Industry, and Competitiveness (MINECO).

A Glossary

We list an overview of terms discussed in the document and relevant for RBMs. For more detail we refer to the code documentation on `https://piquil.github.io/QuCumber/`, and References [14, 15].

- *Batch*: The subset of data selected for one iteration of training in stochastic gradient descent.
- *Biases*: For a visible unit v_j and a hidden unit h_i, the respective biases in the RBM are b_j and c_i. They act like a magnetic field term in the expression for the energy given in Eq. (18).
- *Contrastive divergence*: An approximate maximum-likelihood learning algorithm for RBMs [14].
- *Energy*: The energy of the joint configuration (v, h) of a RBM is defined as follows:

$$E_{\boldsymbol{\lambda}}(\boldsymbol{v}, \boldsymbol{h}) = -\sum_{j=1}^{n_v} b_j v_j - \sum_{i=1}^{n_h} c_i h_i - \sum_{ij} h_i W_{ij} v_j, \tag{18}$$

- *Effective energy*: Obtained from the energy by tracing out the hidden units h; often called the "free energy" in machine learning literature.

$$\mathcal{E}_{\boldsymbol{\lambda}}(\boldsymbol{v}) = -\sum_{j=1}^{n_v} b_j v_j - \sum_{i=1}^{nh} \log\left[1 + \exp\left(\sum_{j}^{n_v} W_{ij} v_j + c_i\right)\right], \tag{19}$$

- *Epoch*: A single pass through an entire training set.
- *Hidden units*: There are n_h units in the hidden layer of the RBM, denoted by the vector $\boldsymbol{h} = (h_1, ..., h_{n_h})$. The number of hidden units can be adjusted to increase the representational capacity of the RBM.
- *Hyper-parameter:* A set of training parameters which do not appear in the training cost function. Examples include the learning rate, number of hidden units, batch size, or number of training epochs
- *Joint distribution*: The RBM assigns a probability to each joint configuration (v, h) according to the Boltzmann distribution:

$$p_{\boldsymbol{\lambda}}(\boldsymbol{v}, \boldsymbol{h}) = \frac{1}{Z_{\boldsymbol{\lambda}}} e^{-E_{\boldsymbol{\lambda}}(\boldsymbol{v}, \boldsymbol{h})}, \tag{20}$$

- *KL divergence*: The Kullback-Leibler divergence, or relative entropy, is a measure for the "distance" between two probability distributions P and Q, defined as:

$$\mathrm{KL}(Q \,||\, P) = \sum_{\boldsymbol{v}} Q(\boldsymbol{v}) \log \frac{Q(\boldsymbol{v})}{P(\boldsymbol{v})} \tag{21}$$

  The KL divergence of identical distributions is zero, while for orthogonal distributions it is infinite.

- *Learning rate*: The step-size used in the gradient descent algorithm for the optimization of the network parameters.

- *Marginal distribution*: Obtained by marginalizing the joint distribution, e.g.

$$p_{\boldsymbol{\lambda}}(\boldsymbol{v}) = \frac{1}{Z_{\boldsymbol{\lambda}}} \sum_{\boldsymbol{h}} e^{-E_{\boldsymbol{\lambda}}(\boldsymbol{v},\boldsymbol{h})} = \frac{1}{Z_{\boldsymbol{\lambda}}} e^{-\mathcal{E}_{\boldsymbol{\lambda}}(\boldsymbol{v})}. \tag{22}$$

- *QuCumber*: A quantum calculator used for many-body eigenstate reconstruction.

- *Parameters*: The set of weights and biases $\boldsymbol{\lambda} = \{\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{c}\}$ characterizing the RBM energy function.

- *Partition function*: The normalizing constant of the Boltzmann distribution. It is obtained by summing over all possible pairs of visible and hidden vectors:

$$Z_{\boldsymbol{\lambda}} = \sum_{\boldsymbol{v},\boldsymbol{h}} e^{-E_{\boldsymbol{\lambda}}(\boldsymbol{v},\boldsymbol{h})}. \tag{23}$$

- *Restricted Boltzmann Machine*: A two-layer network with bidirectionally connected stochastic processing units. "Restricted" refers to the connections (or weights) between the visible and hidden units: each visible unit is connected with each hidden unit, but there are no intra-layer connections.

- *Visible units*: There are n_v units in the visible layer of the RBM, denoted by the vector $v = (v_1, ..., v_{n_v})$. These units correspond to the physical degree of freedom, and n_v is fixed by the number thereof.

- *Weights*: W_{ij} is the symmetric connection or interaction between the visible unit v_j and the hidden unit h_i.

References

[1] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow and J. M. Gambetta, *Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets*, Nature **549**, 242 EP (2017).

[2] N. Moll, P. Barkoutsos, L. S. Bishop, J. M. Chow, A. Cross, D. J. Egger, S. Filipp, A. Fuhrer, J. M. Gambetta, M. Ganzhorn, A. Kandala, A. Mezzacapo *et al.*, *Quantum optimization using variational algorithms on near-term quantum devices*, Quantum Science and Technology **3**(3), 030503 (2018).

[3] H. Bernien, S. Schwartz, A. Keesling, H. Levine, A. Omran, H. Pichler, S. Choi, A. S. Zibrov, M. Endres, M. Greiner, V. Vuletić and M. D. Lukin, *Probing many-body dynamics on a 51-atom quantum simulator*, Nature **551**, 579 EP (2017).

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

[5] 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.

[6] P. Smolensky, *Information Processing in Dynamical Systems: Foundations of Harmony Theory*, MIT Press, Cambridge, MA, USA, ISBN 0-262-68053-X (1986).

[7] G. Hinton, S. Osindero and Y. Teh, *A fast learning algorithm for deep belief nets*, Neural computation **18**(7), 1527 (2006).

[8] G. E. Hinton and R. R. Salakhutdinov, *Reducing the dimensionality of data with neural networks*, Science **313**(5786), 504 (2006), doi:10.1126/science.1127647, `http://science.sciencemag.org/content/313/5786/504.full.pdf`.

[9] G. Torlai and R. G. Melko, *Learning thermodynamics with boltzmann machines*, Phys. Rev. B **94**, 165134 (2016), doi:10.1103/PhysRevB.94.165134.

[10] G. Carleo and M. Troyer, *Solving the quantum many-body problem with artificial neural networks*, Science **355**(6325), 602 (2017), doi:10.1126/science.aag2302, `http://science.sciencemag.org/content/355/6325/602.full.pdf`.

[11] X. Gao and L.-M. Duan, *Efficient representation of quantum many-body states with deep neural networks*, Nature Communications **8**(1), 662 (2017), doi:10.1038/s41467-017-00705-2.

[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] M. Koch-Janusz and Z. Ringel, *Mutual information, neural networks and the renormalization group*, Nature Physics **14**(6), 578 (2018), doi:10.1038/s41567-018-0081-4.

[14] G. E. Hinton, *Training products of experts by minimizing contrastive divergence*, Neural computation **14**(8), 1771 (2002), doi:10.1162/089976602760128018.

[15] G. E. Hinton, *A practical guide to training restricted boltzmann machines*, In *Neural networks: Tricks of the trade*, pp. 599–619. Springer, doi:10.1007/978-3-642-35289-8_32 (2012).