FYS5429/FYS5419 report 2: A presentation and comparison between classical restricted Boltzmann machines and variational quantum Boltzmann machines

F. Pogliano[1, *]

[1]*Department of Physics, University of Oslo, N-0316 Oslo, Norway*

(Dated: May 2024)

In this report classical restricted Boltzmann machines (RBMs) and variational quantum Boltzmann machines (VarQBM) are introduced and compared, the two being quite different interpretations of the Boltzmann machine concept. While the former was used to reproduce the MNIST dataset of handwritten digits, the second was given as a task to learn the probability distribution behind the much simpler bars and stripes (BAS) dataset. While RBMs are fairly capable to reproduce the digits and to an extent accomplish the task of unlabeled classification, the VarQBM algorithm struggles to reproduce correctly the data's probability distribution, only approaching it marginally, despite the relative simplicity of the dataset and long learning times (on personal computer simulations). This shows that VarQBM, despite having intriguing qualities such as not having the necessity of evaluating a partition function, still have some way to go in order to have useful applications.

I. INTRODUCTION

Machine learning (ML) has become an important tool in our society, as it is used in both everyday tasks such as search engines, recommendation systems on music and movie streaming services or chatbots, and in more professional settings such as science, engineering or finance. It allows us to perform specific tasks without explicit instructions, and can thus help us handling problems that before would have been intractable. Machine learning models can be broadly categorized into two types: discriminative and generative. *Discriminative* ML models, normally used for supervised learning, are designed to map input data to labels or outputs. Examples of discriminative ML models are feed-forward neural networks (FFNNs) or convolutional neural networks (CNNs), where the predictions of a model are checked against the given solution, and the model's parameters are changed accordingly to fit the data better. Example tasks of discriminative ML can be regression, or supervised classification. *Generative* ML models instead aim to reproduce the underlying probability distribution of the input data. Examples of Generative ML models can be (variational) autoencoders or generative adversarial networks. These models not only are able to perform (unsupervised) classification tasks, dimensionality reduction and feature extraction, but are also capable to generate new data samples that are similar to the training data. This is done by estimating the probability distribution of the data, and then generating new data that would fit such learned distribution. In this context, energy-based models (EBMs) are an important class of generative ML models, which define the probability distribution through an energy function. Among EBMs, Boltzmann machines (e.g. the widely studied *restricted* Boltzmann machines, or RBMs) are notable examples. These consist of units that make stochastic binary decisions based on the learned distribution encoded in the values of its parameters.

Recently, a whole new computing framework, quantum computing, has become more and more advanced in terms of physical hardware, computational capabilities and possible algorithms. By exploiting the quantum mechanical properties of small physical systems (e.g. electrons), such as superposition and entanglement, quantum computers are in principle able to perform tasks that would be unfeasible on classical computers. This is of interest for the field of ML, where *quantum* machine learning (QML) aim to either improve existing algorithms by substituting specific, time-expensive steps in the learning process by quantum algorithms (*hybrid* QML), or recast whole algorithms in quantum terms. There has been a focus on quantum Boltzmann machines (QBM) in the last few years (see e.g. [1, 2]), possibly due to the intriguing fact that the physical ideas classical BM are based on, are easily translated into a quantum context.

In this report I will give an introduction to classical fully-connected (Section III) and restricted (Section III A) Boltzmann machines, as well as VarQBM (Section III B), a specific variant of QBM that has been shown to work for simple systems [2, 3]. The classical RBM was used to reproduce data from the MNIST dataset, while VarQBM was used to reproduce a much simpler dataset, the bars and stripes dataset. These two datasets are presented in Section II, while the results are presented in Section IV. A discussion then follows in Section V and finally a summary in Section VI, together with a small outlook.

II. THE DATASETS

The MNIST dataset [4] is a dataset of 70000 (normally divided into 60000 training and 10000 test) labeled handwritten digits between 0 and 9, in form of greyscale 28×28

* francesco.pogliano@fys.uio.no

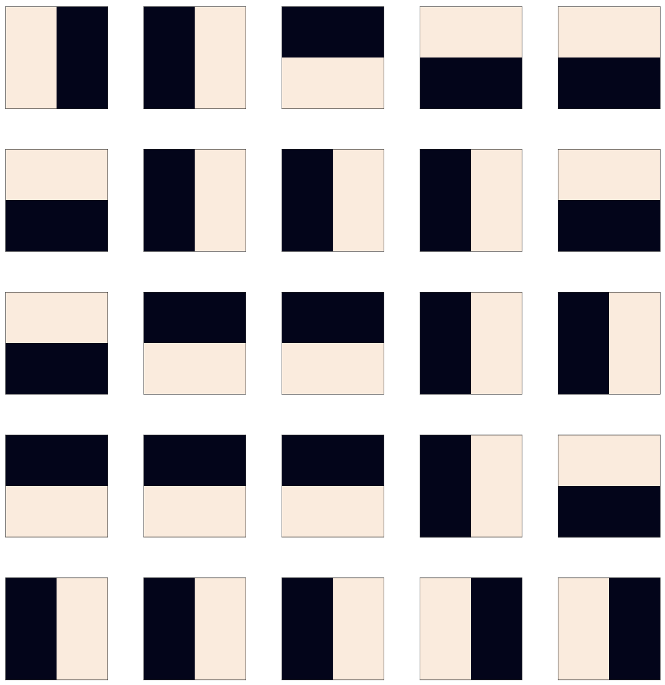

FIG. 1. Randomly generated samples of the bars and stripes (BAS) dataset, consisting of 2×2 matrices of ones and zeros.

pixel images. This is a classical dataset used for benchmarking ML algorithms, as it can be used for classification tasks when using the labeled data, but also for feature extraction, unlabeled classification and training of generative models. In Fig. 4 (left) you see examples of this dataset.

As we will see, the quantum algorithms are not yet capable to handle a lot of data, and the MNIST dataset as will be analyzed using restricted Boltzmann machines would become unfeasible for the quantum Boltzmann machine. For this reason we choose a much simpler, but in concept similar dataset: the bars and stripes (BAS) dataset, consisting of 2×2 matrices of binary values (zeros and ones), organized in such a way that ones and zeros form horizontal bands, or vertical stripes. Originally, the BAS dataset allows for all zeros and all ones matrices, but for this application, these two outcomes have been omitted. Random samples from this dataset are shown in Fig. 1.

III. BOLTZMANN MACHINES

Boltzmann machines (BMs) are generative machine learning algorithms based on the Ising model, where nodes are connected and influence each other's value through weights and biases. In the original definition of BMs, the nodes are divided into two sets, the *visible* and the *hidden* layer. The first represents the input and output layer, while the second codifies the latent features. Nodes may take the binary values of -1 and 1 (or 0 and 1) stochastically[1], depending on the probability calculated from the values of the other nodes and of the parameters of the network. Boltzmann machines are EBMs, as the probability distribution that a node i might be 1 given the network parameters Θ is $p(x_i;\Theta)$, which is expressed through an energy function $E(x_i;\Theta)$ by the Boltzmann distribution

$$p(x_i;\Theta) = \frac{e^{-E(x_i;\Theta)/k_B T_0}}{Z(\boldsymbol{X};\Theta)}, \qquad (1)$$

where $k_B T_0$ is a chosen constant usually set to 1, and $Z(\boldsymbol{X};\Theta)$ is the partition function defined as

$$Z(\boldsymbol{X};\Theta) = \int e^{-E(\boldsymbol{x};\Theta)/k_B T_0} \mathrm{d}\boldsymbol{x}, \qquad (2)$$

and serves as the normalization constant, summing/integrating over all possible configurations. Expressing probabilities in this way, except for the intriguing connection to statistical physics, makes the probabilities always positive and normalized. In the case of Boltzmann machines, the energy $E(\boldsymbol{x};\Theta)$ is described as

$$E(\boldsymbol{x};\Theta) = \sum_i a_i x_i + \sum_{ij} w_{ij} x_i x_j, \qquad (3)$$

analogous to the Ising model. From Eqs. (1) and (3) we infer that the probability is maximized when the energy is lowest. This means that the training task at hand will be to create a Hamiltonian (energy equation), formulated in terms of the parameters $\Theta : \{a_i, w_{jk}\}$, for which the training data $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2 ...\}$ represent the energy minima.

Boltzmann machines have interesting properties, as together with their connection to physical phenomena, they are also global approximators [5]. Nevertheless, they are difficult to train and have no practical uses today. This can be solved by instead employing restricted Boltzmann machines.

A. Restricted Boltzmann machines

In order to make BMs more useful, it is possible to constrain the model by reducing the connectivity between nodes, as is the case for other kinds of neural networks. in *restricted Boltzmann machines* (RBMs), the connections within the visible and the hidden layers are cut, leaving only the connections between the two sets, or layers. A schematic representation of an RBM can be seen in Fig 2.

[1] Variants of the Boltzmann machines allow the nodes to take continuous values and the probability to be Gaussian distributed, but in this report we will only consider the binary variant.

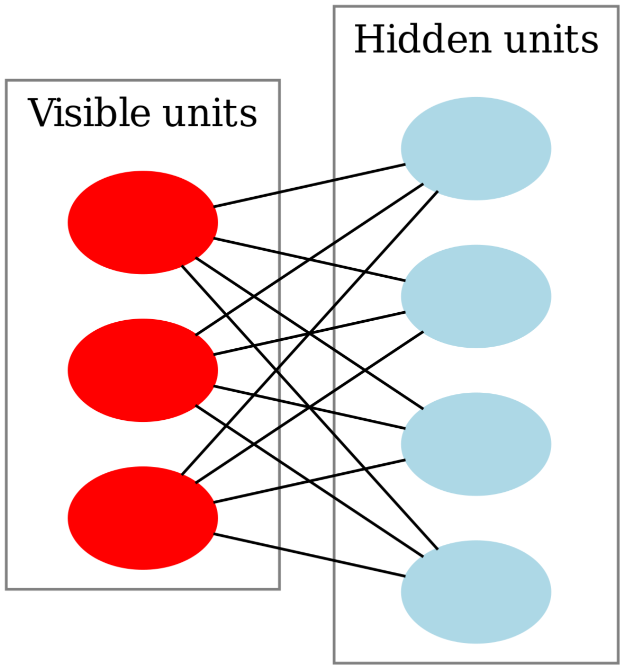

FIG. 2. Schematic representation of a RBM. By Qwertyus - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=22717044

Dividing then the nodes $\boldsymbol{x}$ into visible $\boldsymbol{v}$ and hidden $\boldsymbol{h}$ nodes, we rewrite the energy expression in Eq. (3) as

$$E(\boldsymbol{v},\boldsymbol{h};\Theta) = \sum_n a_n v_n + \sum_m b_m h_m + \sum_{nm} w_{nm} v_n h_m \\ = \boldsymbol{a}^T\boldsymbol{v} + \boldsymbol{b}^T\boldsymbol{h} + \boldsymbol{v}^T\boldsymbol{w}\boldsymbol{h}, \quad (4)$$

where $\Theta : \{a_i, b_j, w_{kl}\}$ are the network's parameters: a_i and b_j biases, and w_{kl} weights.

The objective of the training is then to find the right weight and bias values that recreate the input probability distribution when "bouncing back" against the hidden layer.

The value of one hidden node corresponds mathematically to sampling the probability of it "firing", i.e. to become 1. This probability is calculated from the values of the visible nodes and of the network's parameters. In the language of Bayesian statistics, this is written as:

$$p(\boldsymbol{h}|\boldsymbol{v},\Theta) = \frac{p(\boldsymbol{v},\boldsymbol{h})}{p(\boldsymbol{v})}, \quad (5)$$

where we used Bayes' theorem. From the lecture notes [6], we can show that[2]

$$p(\boldsymbol{v}) = \frac{1}{Z} e^{\boldsymbol{v}^T\boldsymbol{a}} \prod_m^M (1 + e^{b_m + \boldsymbol{v}^T\boldsymbol{w}_{*m}}), \quad (6)$$

and thus

$$p(\boldsymbol{h}|\boldsymbol{v},\Theta) = \prod_m^M \frac{e^{(b_m+\boldsymbol{v}^T\boldsymbol{w}_{*m})h_m}}{1 + e^{b_m+\boldsymbol{v}^T\boldsymbol{w}_{*m}}} \quad (7)$$

and similarly,

$$p(\boldsymbol{v}|\boldsymbol{h},\Theta) = \prod_n^N \frac{e^{(a_n+\boldsymbol{w}_{n*}^T\boldsymbol{h})v_n}}{1 + e^{a_n+\boldsymbol{w}_{n*}^T\boldsymbol{h}}}. \quad (8)$$

From Eqs. (7) and (8), we find that the probability for a node h_m to become 1 will be given as

$$p(h_m = 1|\boldsymbol{v}) = \frac{1}{1 + e^{-(b_m+\boldsymbol{v}^T\boldsymbol{w}_{*m})}} \\ = \sigma(b_m + \boldsymbol{v}^T\boldsymbol{w}_{*m}), \quad (9)$$

where σ represents the sigmoid function

$$\sigma(x) = 1/(1+\exp(-x)), \quad (10)$$

coincidentally a common activation function for many neural networks. Given that we only allow for two values, $p(h_m = 0|\boldsymbol{v})$ is easily calculated to be $1 - p(h_m = 1|\boldsymbol{v})$. For $p(\boldsymbol{v}|\boldsymbol{h})$, in a similar manner,

$$p(v_n = 1|\boldsymbol{h}) = \sigma(a_n + \boldsymbol{h}^T\boldsymbol{w}_{n*}) \\ p(v_n = 0|\boldsymbol{h}) = 1 - p(v_n = 1|\boldsymbol{h}). \quad (11)$$

In other words, in order to decide the value of a hidden node, we first calculate the probability of it to become either 0 or 1 by using the sigmoid function σ using the values in the visible layer $\boldsymbol{v}$ and the parameters $\boldsymbol{w}$ and $\boldsymbol{b}$ (in a similar fashion as how activation functions work in other NNs), and then sample that probability.

The training procedure, similarly to other ML models, consists in minimizing a cost function. For generative models, we usually want to maximize the probability of obtaining the training data by tweaking the parameters that describe the probability distribution. For binary-binary RBMs, the cost function can be defined as

$$\mathcal{L}(\boldsymbol{X};\Theta) = -\langle \log p(\boldsymbol{x},\Theta)\rangle_{\boldsymbol{X}} = -\log \frac{1}{K}\sum_k^K \frac{p(\boldsymbol{x}_k;\Theta)}{Z(\Theta)} \\ = \langle E(\boldsymbol{x}|\Theta)\rangle_{\boldsymbol{X}} + Z(\Theta), \quad (12)$$

[2] From this point on, we need the assumption that all stochastic variables are independent and identically distributed (iid).

where K is the number of data points in the training set, and $\langle ... \rangle_{\boldsymbol{X}}$ represent the averaging over all training data $\boldsymbol{X}$. The minus sign ensures that maximizing the probability corresponds to minimizing the cost function. There are a few reasons why we are trying to maximize the log-likelihood instead of the likelihood: one is that the derivatives become easier to handle, and the logarithm of a function behaves for our purposes in the same way as its argument (notably the minima and maxima stay in the same place, and no new such features are created). The second is that this corresponds to the Kullback-Leibler (KL) divergence, this being a measure of the difference between probability distributions, defined as

$$\mathrm{KL}(p||q) = \int_{-\infty}^{\infty} p(\boldsymbol{x}) \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} d\boldsymbol{x}. \tag{13}$$

If we replace p with the probability distribution of our data $f(\boldsymbol{x})$, and q with the probability distribution expressed by model given Θ parameters $p(\boldsymbol{x}|\Theta)$, then Eq. (13) becomes [6]

$$\begin{aligned} &\mathrm{KL}(f(\boldsymbol{x})||p(\boldsymbol{x}|\Theta)) \\ &\qquad = \langle \log f(\boldsymbol{x}) \rangle_{\boldsymbol{X}} + \langle E(\boldsymbol{x}) \rangle_{\boldsymbol{X}} + \log Z, \end{aligned} \tag{14}$$

where the first term is constant, and the last two correspond to the log-likelihood expressed in Eq. (12). Minimizing the cost function, in other words, corresponds to minimizing the KL divergence. We can find the minimum in the cost function with respect to the parameters Θ by gradient descent. The first step is then to take the derivative of the cost function:

$$\frac{\partial \mathcal{L}(\Theta)}{\partial \Theta} = \langle \frac{\partial E(\boldsymbol{x}; \Theta)}{\partial \Theta} \rangle_{\boldsymbol{X}} + \frac{\partial \mathrm{log} Z(\Theta)}{\partial \Theta}. \tag{15}$$

The two terms in Eq. (15) are called the positive and the negative phase, respectively. Intuitively, one can understand the first one as increasing the probability where we have the training data, and the second as decreasing it far from the data. While the first term is easy to compute, the second is intractable as the partition function is almost impossible to calculate in most of the cases. In order to approximate this, we use an algorithm called *contrastive divergence*. For this, we rewrite the second term as

$$\begin{aligned} \frac{\partial \mathrm{log} Z(\Theta)}{\partial \Theta} &= \frac{1}{Z(\Theta)} \frac{\partial Z(\Theta)}{\partial \Theta} \\ &= \frac{1}{Z(\Theta)} \int \frac{\partial e^{-E(\boldsymbol{x};\Theta)}}{\partial \Theta} \mathrm{d}\boldsymbol{x} \\ &= \frac{1}{Z(\Theta)} \int e^{-E(\boldsymbol{x};\Theta)} \frac{\partial \mathrm{log} e^{-E(\boldsymbol{x};\Theta)}}{\partial \Theta} \mathrm{d}\boldsymbol{x} \\ &= \int \frac{e^{-E(\boldsymbol{x};\Theta)}}{Z(\Theta)} \frac{\partial \mathrm{log} e^{-E(\boldsymbol{x};\Theta)}}{\partial \Theta} \mathrm{d}\boldsymbol{x} \\ &= \int p(\boldsymbol{x};\Theta) \frac{\partial \mathrm{log} e^{-E(\boldsymbol{x};\Theta)}}{\partial \Theta} \mathrm{d}\boldsymbol{x} \\ &= -\langle \frac{\partial E(\boldsymbol{x};\Theta)}{\partial \Theta} \rangle_{model}, \end{aligned} \tag{16}$$

meaning that the second term can be approximated by the expectation value of the model, i.e. by sampling the model many times and calculate its expectation value. By the same token, the first term in Eq. (15) can be calculated by sampling the training data and calculate its expectation value. While "sampling the data" simply means picking one point from the training set and run it through the model, and obtain

$$\begin{aligned} \langle \frac{\partial E(\boldsymbol{x};\Theta)}{\partial \boldsymbol{a}} \rangle_{\boldsymbol{X}} &= \boldsymbol{v} \\ \langle \frac{\partial E(\boldsymbol{x};\Theta)}{\partial \boldsymbol{b}} \rangle_{\boldsymbol{X}} &= \boldsymbol{h} \\ \langle \frac{\partial E(\boldsymbol{x};\Theta)}{\partial \boldsymbol{w}} \rangle_{\boldsymbol{X}} &= \boldsymbol{v} \otimes \boldsymbol{h}, \end{aligned} \tag{17}$$

where $\otimes$ indicates the tensor product for which e.g. for $\boldsymbol{C} = \boldsymbol{A} \otimes \boldsymbol{B}$ is $C_{ij} = A_i B_j$. The "sampling of the model" is a bit more complicated. This is done using a Markov chain Monte Carlo (MCMC) method, more specifically, a process called Gibbs sampling. The way these samples are generated, is by exploiting the principle of the random walk, or Brownian motion. We pick any starting point, and by then choosing and moving in a random direction repeatedly we will statistically end up in the macrostate of the system with the most microstates, i.e. the steady state with the maximum entropy. Once this thermal equilibrium state is reached (the chain has *burned in* [7]), we may sample from the distribution. While this is true for any MCMC method, Gibbs sampling specifically picks the direction in the space spanned by Θ by only changing one parameter at a time. Gibbs sampling is computationally inexpensive because we only change one parameter for each step, but may not perform well when certain conditions are not met: the parameters should be independent (correlations may not allow a representative sampling of the distribution) and the distribution should not be multimodal (otherwise the process may burn into one of the energy minima and not get out of it and sample the potential others). Since the nodes in RBMs are not connected within layers, *block Gibbs sampling* is allowed, meaning sampling all the hidden nodes or the visible nodes at once[3]. Usually even just one iteration of the Gibbs sampling using a data point from the training data gives a good approximation of the model's probability distribution [8]. This means that by picking a training data point $\boldsymbol{v}$ from $\boldsymbol{X}$, running it through the network and obtaining $\boldsymbol{h}$ by sampling from $\sigma(\boldsymbol{b}+\boldsymbol{w}\cdot\boldsymbol{v})$, repeating the process backwards and obtaining $\boldsymbol{v'}$ by sampling from $\sigma(\boldsymbol{a}+\boldsymbol{w}\cdot\boldsymbol{h})$, and forward again

[3] These issues may be addressed with tempering, meaning tweaking the $k_B T_0$ parameter in order to make the probability peaks less sharp and allow for better mixing during the sampling [7].

to get $\boldsymbol{h'}$, we obtain

$$\begin{aligned}\Delta\boldsymbol{a} &= \epsilon(\boldsymbol{v}-\boldsymbol{v'})\\ \Delta\boldsymbol{b} &= \epsilon(\boldsymbol{h}-\boldsymbol{h'})\\ \Delta\boldsymbol{w} &= \epsilon(\boldsymbol{v}\otimes\boldsymbol{h}^T-\boldsymbol{v'}\otimes\boldsymbol{h'}^T),\end{aligned} \tag{18}$$

where ϵ is the learning rate, a hyperparameter which may be constant, or allowed to vary e.g. using Adam. It is also reassuring that the conclusion seem "logical": the gradients for the parameters go to zero, the closer the generated output gets to the training data.

Implementation

The code for the MNIST dataset analysis can be found in `RBM.ipynb`, where a `RBM` class is used, defined in `RBM.py` using the Keras API for Tensorflow [9]. This is an adapted version of the one proposed by Babcock and Bali [8], where only the RBM part is taken, without considering Deep Belief Networks. In this particular code, the gradients are not calculated using Eqs. (18), but with autograd (calculated with the `gradient` method of `GradientTape` from Tensorflow).

B. VarQBM

The concept of quantum Boltzmann machines (QBMs) is relatively young [1], and it refers to a quantum generalization of the Boltzmann machines introduced in section III. Being still a young and dynamic field, the concept of QBMs is still not well-defined, with different versions and extensions depending to the implementation, the training, the quantum computing models and so on [3]. One implementation of QBM that involves quantum computing in both the training and the model steps, that has proved successful to solve simple systems and that will be the focus of this report, is the variational QBM (VarQBM) [2]. As for the other implementations of QBM, VarQBM "quantizes" the classic BM by encoding information onto a *Gibbs state*, where the QBM is in thermal equilibrium, and is usually represented by the density matrix

$$\rho = \frac{e^{-H/k_BT}}{\mathrm{Tr}\left(e^{-H/k_BT}\right)}, \tag{19}$$

where usually, as for classical BMs, $1k_BT = 1$. From the Gibbs state we can recover the probability P_v of measuring a specific state v by:

$$P_v^{\mathrm{QBM}} = \mathrm{Tr}(\Lambda_v\rho), \tag{20}$$

where $\Lambda_v = |v\rangle\langle v| \otimes I_h$ is a projection operator. As anticipated by the subscripts, we are usually interested in finding the probability of measuring a state represented by *visible nodes* v, independent of the values of the *hidden nodes* h. In QBM, "nodes" are qubits, and the visible ones will be the ones representing the output of the quantum circuit. The main objective of training a QBM is to find the right Gibbs state for which Eq. (20) represents the training data's probability distribution. This is done by parametrizing the Hamiltonian, and finding the right values for its parameters. As usual for quantum computing, we define the Hamiltonian in terms of tensor products of Pauli matrices

$$H_\theta = \sum_i^p \theta_i h_i, \quad h_i = \bigotimes_j^n \sigma_j, \tag{21}$$

where $\sigma_j = \{\sigma_x, \sigma_y, \sigma_z, I\}$ are the 2×2 Pauli matrices and identity matrix, and θ_i are the real parameters. What characterizes VarQBM, is the way such a Gibbs state is prepared, and it is using the variational quantum imaginary time evolution (VarQITE) method.

VarQITE

The variational quantum imaginary time evolution (VarQITE) is a method used to build the Gibbs state. It is based on the fact that, starting from a Hamiltonian such as the one in Eq. (21) and an initial state $|\psi_0\rangle$, one may write

$$|\psi_\tau\rangle = C(\tau)e^{-H\tau}|\psi_0\rangle, \tag{22}$$

where $|\psi_\tau\rangle$ is the imaginary time evolution of $|\psi_0\rangle$, and $C(\tau)$ is a normalization factor defined as

$$C(\tau) = \frac{1}{\mathrm{Tr}\left(e^{-2H\tau}|\psi_0\rangle\langle\psi_0|\right)}. \tag{23}$$

We observe here for example that when $\tau \to 0$, then $|\psi_\tau\rangle \to |\psi_0\rangle$, and for $\tau\to\infty$ the ground state of H will dominate, since it will be the component that approaches zero the slowest, and $C(\tau)$ ensures continuous normalization. While normally we could use this method to find the ground state of a Hamiltonian, in VarQBM we use it in order to build a Gibbs state by only evolving the initial state to $\tau = 1/(2k_BT)$.

In order to reach the right Gibbs state, we need to choose both the initial state and the Hamiltonian right. To do this, the initial state is also parametrized. As explained in Zoufal et al. [2], the initial state is built by first preparing a maximally mixed state, and then evolving it through a parametrized quantum circuit. The maximally mixed state is prepared by initializing a set of $2n$ qubits at 0, dividing it into two subsets a and b, and preparing Bell states by entangling qubits from a and b in pairwise fashion. Secondly, we choose a parametrized quantum circuit $V(\boldsymbol{\omega}) = U_q(\omega_q)U_{q-1}(\omega_{q-1})...U_1(\omega_1)$ with quantum gates U_i and parameters ω_i such that $|\psi_\omega\rangle = V(\boldsymbol{\omega})|\psi_{in}\rangle$., where $|\psi_{in}\rangle$ is our maximally mixed initial state. The $|\psi_\omega\rangle$ state will be our approximation of the Gibbs state once the subsystem b is traced out

$$\rho_\omega = \mathrm{Tr}_b\left(|\psi_\omega\rangle\langle\psi_\omega|\right) \approx \frac{e^{-H/k_BT}}{\mathrm{Tr}(e^{-H/k_BT})}. \tag{24}$$

Instead of first preparing $|\psi_\omega\rangle$ and then propagating it to $|\psi_\omega(\tau = (2k_BT)^{-1})\rangle$, we project the imaginary time evolution to the parameters $\boldsymbol{\omega}$, so that $\boldsymbol{\omega} \rightarrow \boldsymbol{\omega}(\tau)$. The QITE equation in Eq. (22) is described by the differential equation

$$\frac{\mathrm{d}|\psi_\tau\rangle}{\mathrm{d}\tau} = -(H - E_\tau)|\psi_\tau\rangle, \tag{25}$$

where $E_\tau = \langle\psi_\tau|H|\psi_\tau\rangle$. Using McLahan variational principle [10], this can be expressed as a system of linear equations in the form

$$\boldsymbol{A}\frac{\mathrm{d}\boldsymbol{\omega}}{\mathrm{d}\tau} = \boldsymbol{C}, \tag{26}$$

where

$$\begin{aligned} A_{ij} &= \mathrm{Re}\left(\mathrm{Tr}\left[\frac{\partial V^\dagger\omega(\tau)}{\partial\omega(\tau)_i}\frac{\partial V\omega(\tau)}{\partial\omega(\tau)_j}\rho_{in}\right]\right) \\ C_i &= -\sum_k \theta_k \mathrm{Re}\left(\mathrm{Tr}\left[\frac{\partial V^\dagger\omega(\tau)}{\partial\omega(\tau)_i}h_i V(\omega(\tau))\rho_{in}\right]\right) \end{aligned} \tag{27}$$

and $\rho_{in} = |\psi_{in}\rangle\langle\psi_{in}|$. The $\boldsymbol{A}$ and $\boldsymbol{C}$ terms can be calculated on quantum circuits [2] and $\mathrm{d}\boldsymbol{\omega}/\mathrm{d}\tau$ used to calculate the imaginary time evolution of $\boldsymbol{\omega}$ using e.g. Euler's method

$$\boldsymbol{\omega}(\tau) = \boldsymbol{\omega}(0) + \sum_j \frac{\mathrm{d}\boldsymbol{\omega}(j\delta\tau)}{\mathrm{d}\tau}\delta\tau. \tag{28}$$

Loss function

Once the Gibbs state is prepared, its probability predictions can be compared to the training data using a loss function, and the Hamiltonian θ parameters updated using classical methods exploiting $\mathrm{d}\mathcal{L}/\mathrm{d}\theta$. As in other ML algorithms, the loss function can be chosen is customizable to the needs of the problem at hand. In QBM, there are two options: the classical and the quantum loss functions. Although VarQBM may be run using both, in Zoufal et al. [2] the choice falls on the classical loss, as it avoids certain problems with the evaluation of analytical gradients. This is defined as

$$\mathcal{L} = -\sum_v P_v^{\mathrm{data}}\log P_v^{\mathrm{QBM}}, \tag{29}$$

where P_v^{data} is the probability of finding v in the training data set, and P_v^{QBM} is defined in Eq. (20).

The gradient of the loss function with respect to the Hamiltonian parameters is then

$$\frac{\partial\mathcal{L}}{\partial\theta_i} = -\sum_v P_v^{\mathrm{data}}\frac{\partial P_v^{\mathrm{QBM}}/\partial\theta_i}{P_v^{\mathrm{QBM}}}, \tag{30}$$

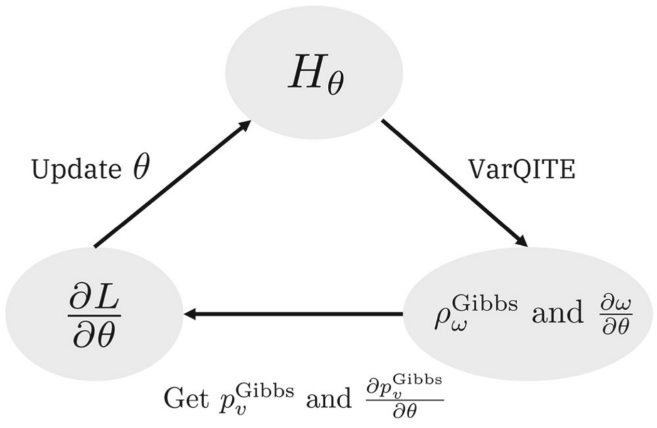
H_θ
VarQITE
$\rho_\omega^{\mathrm{Gibbs}}$ and $\frac{\partial\omega}{\partial\theta}$
Get p_v^{Gibbs} and $\frac{\partial p_v^{\mathrm{Gibbs}}}{\partial\theta}$
$\frac{\partial L}{\partial\theta}$
Update θ

FIG. 3. Scheme summarizing the VarQBM algorithm. First, a parametrized Hamiltonian is initialized. This is used to evolve a parametrized initial state to an approximation of the Gibbs state using VarQITE. This is compared to the training data probabilities, and used to find the variation of the loss function with respect to the Hamiltonian's parameters. Finally, the gradient is calculated and the parameters updated. Reused from Zoufal et al. [2]

where

$$\frac{\partial P_v^{\mathrm{QBM}}}{\partial\theta_i} = \sum_k \frac{\partial P_v^{\mathrm{QBM}}}{\partial\omega_k(\tau)}\frac{\partial\omega_k(\tau)}{\partial\theta_i}. \tag{31}$$

To calculate this, we need to evaluate P_v^{QBM}, $\partial P_v^{\mathrm{QBM}}/\partial\omega_k(\tau)$ and $\partial\omega_k(\tau)/\partial\theta_i$. The first, P_v^{QBM}, can be directly sampled from the approximation of the Gibbs state. One of the big differences between QBMs and classical BMs is that by the introduction of the Gibbs state, the sampling is already normalized, removing the need of calculating the partition function. The second, $\partial P_v^{\mathrm{QBM}}/\partial\omega_k(\tau)$, can be computed using quantum gradient methods (see Ref. [2] page 6 and references therein). The third, $\partial\omega_k(\tau)/\partial\theta_i$, is evaluated by computing the derivatives of Eq. (26),

$$\frac{\partial(\boldsymbol{A}\mathrm{d}\boldsymbol{\omega}/\mathrm{d}\tau)}{\partial\theta_i} = \frac{\partial\boldsymbol{C}}{\partial\theta_i} \tag{32}$$

from which we obtain

$$\frac{\partial(\mathrm{d}\boldsymbol{\omega}/\mathrm{d}\tau)}{\partial\theta_i} = \boldsymbol{A}^{-1}\frac{\partial\boldsymbol{C}}{\partial\theta_i} - \boldsymbol{A}^{-1}\frac{\partial\boldsymbol{A}}{\partial\theta_i}\frac{\mathrm{d}\boldsymbol{\omega}}{\mathrm{d}\tau}, \tag{33}$$

which allows us to find $\partial\omega_k(\tau)/\partial\theta_i$ again by Euler's method. The partial derivatives of $\boldsymbol{A}$ and $\boldsymbol{C}$ can be also be evaluated with quantum circuits [2]. A schematic representation of the VarQBM method is shown in Fig. 3.

Implementation

The analysis of the BAS dataset was carried out in `varqbm_experiment.ipynb`. This is a slightly modified version for plotting purposes of the one found in https://github.com/leevilehtonen/tfq-varqbm/tree/main, a repository linked to the work of Leevi Lehtonen [3]. In this implementation of VarQBM, TensorFlow Quantum [11] was used, together with the Google-developed, Python quantum computing simulation library Cirq [12]. The code is modular and easily understandable. As seen in the Jupyter notebook, first a parametrized Hamiltonian is created by instantiating the `Hamiltonian` class, defined in `Hamiltonian.py`. The ansatz circuit to be used in the VarQITE part of the algorithm is defined by calling `build_ansatz` from `circuit.py`, and then used as input to create the network by instantiating a `QBM` class, from `QBM.py`. When creating a `QBM` object, a `QITE` object is also created, taking care of the creation of the Gibbs state. This is defined in `qite.py`. In `dataset.py` one can find functions taking care of the dataset manipulations, and in `utils.py` the resting functions. The evaluation of $\boldsymbol{A}$ in Eq. (26) is computationally hard, and for this reason it was optimized using simultaneous perturbation stochastic approximation (SPSA) [13]. This for a faster calculation, and details can be found in Leevi Lehtonen's Master's thesis [3].

IV. RESULTS

A. RBM

The RBM model for the reproduction of the MNIST dataset was set up with 784 visible nodes (corresponding to the number of pixels in each 28×28 image) and 200 hidden nodes. The standard choice of hidden nodes in order to model the MNIST dataset is somehow set to 500 [8, 14], but no difference in training time or quality of results was observed by varying it between 150 and 500, so 200 was chosen for our training. The dataset was divided into 60000 training images, and 10000 test images. After training, the test digits are very well reproduced, especially if one takes an average of 100 runs, as this will average out the fluctuations from the stochastic choices of 0s and 1s, and give a representation of the learned probabilities. In Fig. 4 we observe 100 MNIST digits from the test set (left), together with their generated counterpart (right), for comparison. In Fig. 5 we also observe the first half of weights used for the modeling, each image showing the 28×28 weights linking each of the 100 represented hidden nodes to the visible nodes. Although these are difficult to interpret, we notice horizontal and diagonal stripes used by the model to recognize the different digits. A part for being used as a generative model, RBMs may be used for unlabeled classification. When the number of hidden nodes is less than for the visible ones, in practice we do a dimensionality reduction which allows for principal component analysis (PCA). This is indeed the case when observing Fig. 6, where the parameters mapping the first 10000 digits of the training set were mapped onto a 2D graph using SciKit-learn's TSNE embedding function [15]. Here we see how many of the different digits are grouped together, although this does not work perfectly for those handwritten digits that share many similarities (e.g. 4 and 9), given that the 2D mapping is representative of the real clustering in parameter-space.

B. VarQBM

Although we can say that the RBM was successful in learning and modeling the MNIST dataset, the same cannot be said for the VarQBM and the BAS dataset. The implementation followed closely the one done by Lehtonen [3], where an ansatz of three layers with consecutive y and z rotation gates followed by parametrized XNOT gates in chain configuration, and finally again a y and z rotation gates plus CNOT gates coupling (shown in `varqbm_experiment.ipynb`) was used for the VarQITE algorithm. The ansatz covers all eight qubits (problem plus ancillary qubits). The Hamiltonian was instead modeled as using parametrized Z and ZZ gates, the last ones linking each pair of problem qubits. Despite the training time of > 48 h on a personal computer, the (bad) results obtained by Lehtonen were confirmed, where results other than bars and stripes were generated, see Fig. 7. The results are shown as 2×2 matrices, these being representations of binary strings, wrapped in a matrix shape. The binary strings correspond to the 16 combinations four qubits can take or, in other words, the 16 different states a system of four qubits can find itself in. In this representation, the training data will be only represented by four combinations: $|1100\rangle$, $|0011\rangle$ for the two stripes, and $|1010\rangle$, $|0101\rangle$ for the two bands, and any of these will have 0.25 probability of occurring. This is shown in Fig. 8, where the expected probabilities are compared to the ones obtained by the training. Although somewhat resembling the training data (and thus showing that the model indeed tries to learn), the results are underwhelming.

V. DISCUSSION

Although fallen out of favour to more powerful generative ML algorithms such as variational autoencoders or generative adversarial networks, RBMs are still a working alternative for relatively simple datasets and lay the background to understanding energy-based models. In our application, we have seen how an RBM can quite effectively reproduce the training data of the MNIST dataset, even with a dimensionality reduction from 786 visible nodes, to 200 hidden ones. The same algorithm can be used for unlabeled classification, as the latent space allows us to carry out a principal component analysis. As it is shown in Fig. 6, this is effective when it comes digits that are quite different from each other, such as 1 (a line) and 0 (a circle), while this may become more blurry for numbers such as

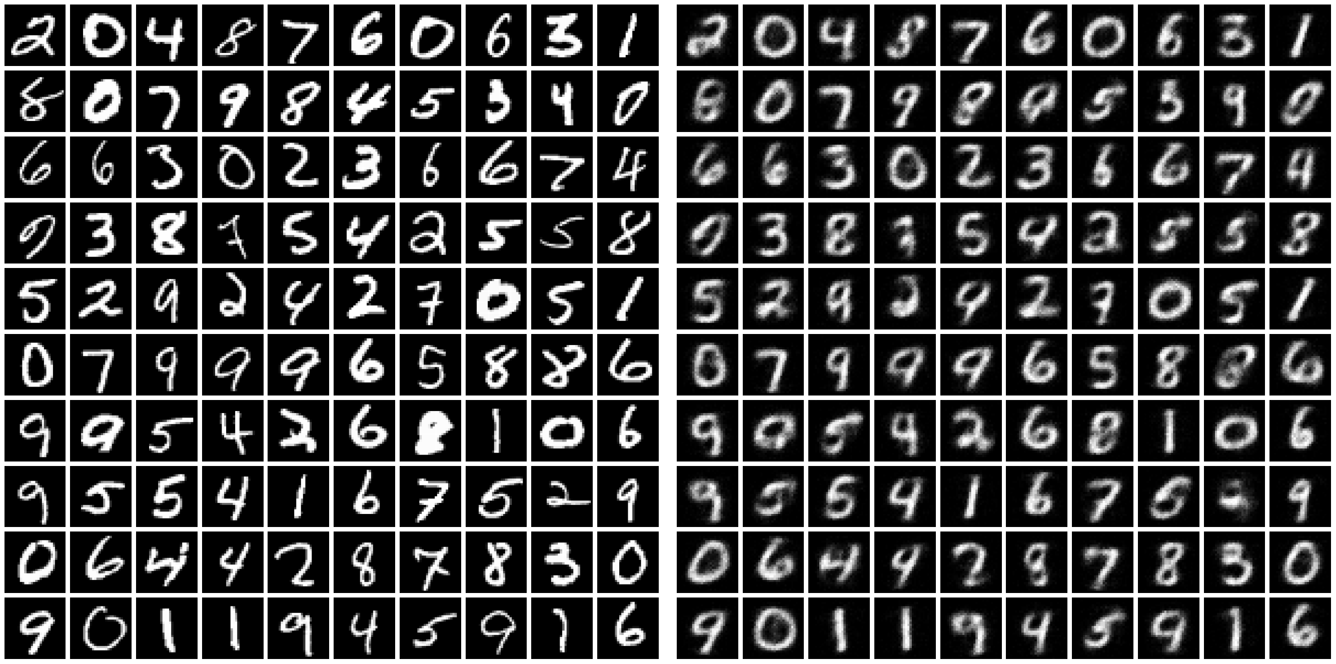

FIG. 4. One hundred samples of the test set used for the MNIST RBM analysis to the leftm and the respective generated images. The images on the right were generated by averaging the solutions of 100 runs.

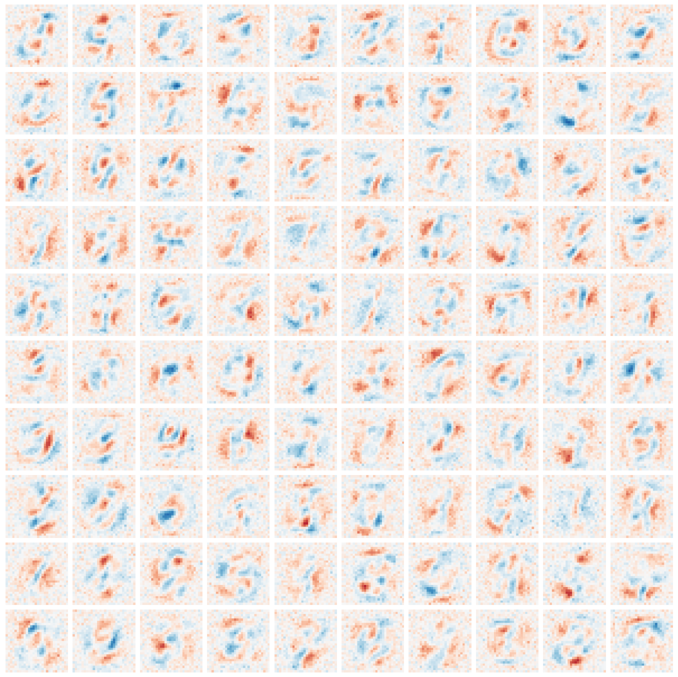

FIG. 5. A heatmap of the values of the weights for the first 100 hidden nodes of the RBM, wrapped as 28×28 matrices.

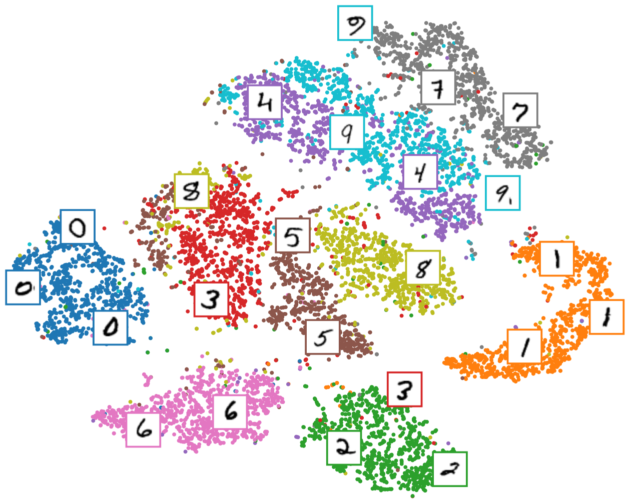

FIG. 6. The TSNE embedding of the training set, showing a 2D mapping of the PCA. We see how many digits are well separated by others, showing some success in the unlabeled classification task, although the job becomes difficult for similar digits such as 4 and 9, given that this 2D mapping is representative of the parameter-space.

4 and 9, and somewhat for 5, 3 and 8. The quantum counterpart of the BMs considered in this report is the VarQBM algorithm as presented in Zoufal et al. [2], which outlines a fully connected BM where qubits take the place of nodes, and first a thermal state is generated using variational quantum imaginary time evolution, incorporating all the information of the system in its Hamiltonian. Despite being attractive from a theoretical and practical point of view, since the algorithm may be run on present NISQ (noisy intermediate-scale quantum)

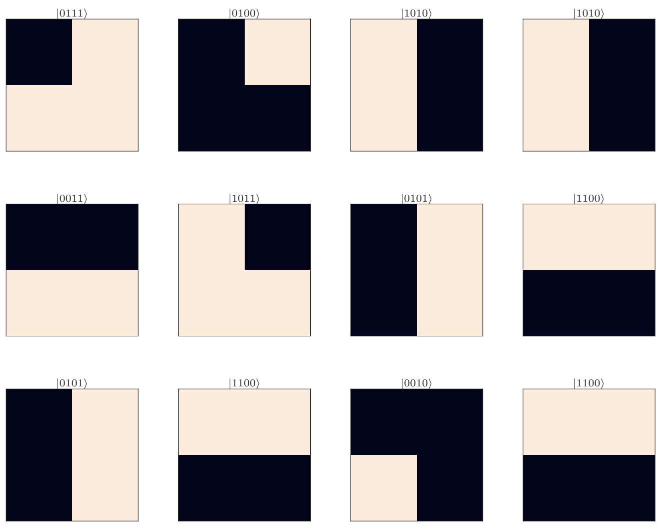

FIG. 7. Twelve random generated images from the learned VarQBM model. Of these, four are not bars or stripes, making the error rate for this specific (small) sampling about 33%.

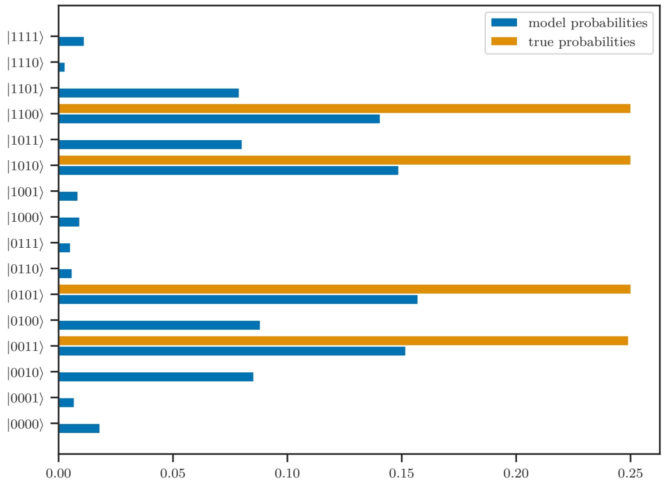

FIG. 8. Learned probabilities for the VarQBM model, compared to the theoretical ones. As we see, even if the probabilities are somewhat in the right ballpark, the probability distribution for spurious solutions is still fairly large.

hardware, predictions for simple datasets as the BAS are underwhelming. Although doing a relatively good job in learning Bell and GHZ states [2], the algorithm still struggles with learning more multimodal datasets. The usefulness of this model may also rely on the possibility of reducing the dimensionality of input data (e.g. by folding, pre-processing), as one needs two qubits for every input data point.

VI. SUMMARY AND OUTLOOK

In the context of generative learning, Boltzmann machines are an interesting algorithm from a physical point of view as it is an energy-based system, modeled after the Ising model. In this report we introduced and compared two interpretations of Boltzmann machines, a machine learning concept used for generative learning. The classical restricted Boltzmann machine is a stochastic model formed by nodes divided into visible and hidden, connected by weights and biases which represent the model's parameters. The model tries to learn the probability distribution of the input data by minimizing a loss function formulated as the Kullback-Leibler divergence, or the difference between the model and the data probability distributions. The quantum version of BMs described here is VarQBM. We talk in this case of fully-connected BM (and not *restricted*), where the aim is to generate a quantum Gibbs state: a superposition of the 2^N possible states using N qubits, each of which represents a (possibly labeled) outcome and their measurement probabilities represent the probability distribution of the model. While the classical RBM was shown to reproduce the MNIST dataset fairly well even with 200 hidden nodes (less than the 500 normally used in the literature), VarQBM still struggles on relatively simple datasets such as the BAS, where even with extensive training the model probability distribution only slightly approaches the one provided by the data. Even though the algorithms was shown in other works that it is capable to reproduce Bell states and GHZ states [2], more multimodal datasets still present a struggle.

A part for a general improvement of VarQBM towards being capable to analyze bigger and more complex datasets, it would be interesting to compare the training time of a VarQBM simulation on a personal computer against quantum computers, to see if there is already an appreciable difference. VarQBMs, although limited as of today, present intriguing qualities such as not having to estimate a partition function, which could potentially make them very useful in the context of generative ML.

[1] M. H. Amin, E. Andriyash, J. Rolfe, B. Kulchytskyy, and R. Melko, Quantum boltzmann machine, Phys. Rev. X **8**, 021050 (2018).

[2] C. Zoufal, A. Lucchi, and S. Woerner, Variational quantum boltzmann machines, Quantum Machine Intelligence **3**, 7 (2021).

[3] L. Lehtonen, engAnalysis and implementation of quantum boltzmann machines (2021).

[4] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE **86**, 2278 (1998).

[5] N. Le Roux and Y. Bengio, Deep Belief Networks Are

Compact Universal Approximators, Neural Computation **22**, 2192 (2010), https://direct.mit.edu/neco/article-pdf/22/8/2192/842688/neco.2010.08-09-1081.pdf.

[6] M. Hjorth-Jensen, *Lecture notes, course FYS5429* (Department of Physics, University of Oslo and Department of Physics and Astronomy and National Superconducting Cyclotron Laboratory, Michigan State University, 2024).

[7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, 2016) `http://www.deeplearningbook.org`.

[8] J. Babcock and R. Bali, eng*Generative AI with Python and TensorFlow 2: Create Images, Text, and Music with VAEs, GANs, LSTMs, Transformer Models*, 1st ed. (Packt Publishing, Limited, Birmingham, 2021).

[9] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems (2015), software available from tensorflow.org.

[10] A. McLachlan, A variational solution of the time-dependent schrodinger equation, Molecular Physics **8**, 39 (1964), https://doi.org/10.1080/00268976400100041.

[11] M. Broughton, G. Verdon, T. McCourt, A. J. Martinez, J. H. Yoo, S. V. Isakov, P. Massey, R. Halavati, M. Y. Niu, A. Zlokapa, E. Peters, O. Lockwood, A. Skolik, S. Jerbi, V. Dunjko, M. Leib, M. Streif, D. V. Dollen, H. Chen, S. Cao, R. Wiersema, H.-Y. Huang, J. R. McClean, R. Babbush, S. Boixo, D. Bacon, A. K. Ho, H. Neven, and M. Mohseni, Tensorflow quantum: A software framework for quantum machine learning (2021), arXiv:2003.02989 [quant-ph].

[12] C. Developers, Cirq (2023).

[13] J. Spall, Multivariate stochastic approximation using a simultaneous perturbation gradient approximation, IEEE Transactions on Automatic Control **37**, 332 (1992).

[14] G. E. Hinton, S. Osindero, and Y.-W. Teh, A fast learning algorithm for deep belief nets., Neural computation. **18**, 1527 (20067).

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research **12**, 2825 (2011).