Classical Machine Learning for Quantum Systems and Quantum Enhanced Machine Learning

Dariusz Lasecki

July 4, 2019

Abstract

In this paper we summarize two research publications from the area of Quantum Machine Learning: *A Neural Decoder for Topological Codes* (Torlai, Melko) and *Quantum Perceptron Models* (Wiebe, Kapoor, Svore). They serve as representatives for Classical Learning for Quantum Systems and Quantum Enhanced Learning respectively. Before explaining core ideas presented in each of the papers, we introduce essential concepts from classical Machine Learning to a reader. The first paper explains how can we use a classical Restricted Boltzmann Machine to create decoders for a 2D Toric Quantum Error Correction code. The second paper proposes two quantum approaches to the Perceptron Model and proves their advantage over classical methods. At the end of this paper we provide a summary and outlook on the field.

Chapter 1

Introduction

1.1 Motivation and overview

Quantum Information Theory provides new possibilities in terms of information processing and communication like quantum algorithms of time complexity better than their classical counterparts (e.g. the Grover's search algorithm [1]) or quantum cryptography for fundamentally secure communication (e.g. BB84 protocol [2]). It is therefore well-motivated to investigate possible improvements that could be achieved in the context of machine learning using the framework of quantum mechanics. Classical machine learning proved to be a useful approach for the wide variety of problems in classical information theory and computing (e.g. self-driving cars [3], text categorization [4]). It offers quite generic yet very powerful tools for solving problems. They can be used in a black-box manner without a need for problem-specific algorithms.

Quantum Machine Learning is the term that can be understood in several ways which also leads to some confusions in the nomenclature. It has a source in what resource do we consider quantum and which classical. We may have classical or quantum data and classical or quantum computer which operates on them. In this project we will consider two possibilities, i.e. classical machine learning for solving quantum problems (Classical Learning for Quantum Systems) and quantum methods used to operate on quantum or classical data (Quantum Enhanced Learning).

Chapter 2

Classical Learning for Quantum Systems

2.1 Overview

Classical Machine Learning has been applicable and beneficial for many domains. It is therefore not surprising that certain problems arising in quantum settings can also benefit from these tools. There are already numerous publications reporting such usefulness and they can be divided into the following classes [5]: learning about quantum systems (e.g. Hamiltonian learning [6], unknown quantum states classification [7]), controlling quantum systems (e.g. many-particle adaptive quantum metrology [8], designing quantum gates [9]), learning properties of quantum and statistical physics (e.g. critical points of phase transitions [10], expectation values of observables [11]). For a comprehensive overview see [5]. In this paper we will summarize an example which falls into the category of learning about quantum systems.

2.2 Example - A Neural Decoder for Topological Codes

We will present how to use classical neural networks in the form of a Restricted Boltzmann Machine for recognizing errors in a 2D toric code based on a detected error syndrome.
This chapter is based on the paper:
G. Torlai, R. G. Melko; A Neural Decoder for Topological Codes
Physical Review Letters, **119**, 030501, (2017).

2.2.1 2D Toric Code

The 2D toric code encodes logical qubits into a greater number of physical qubits placed on a 2D lattice with periodic boundary conditions. Therefore, we can imagine that we have a lattice of qubits placed on a torus. We assume that physical qubits are represented as edges of the lattice and their states can be modified by applying Pauli operators. The important property of the 2D toric code is that it falls into the category of stabilizer codes which means that we can divide operators acting on the system into two categories. The first category are operators which act trivially in the code subspace, i.e. elements of the code are eigenvectors of such operators. It can be proved that such operators form a group S which is called the code stabilizer. Importantly, although elements of the stabilizer group does not change the state of logical qubits, they are capable of detecting error syndromes which can then be used for performing a proper error-correcting operation. The second group of operators are those which act non-trivially on the codespace which means that they actually change the state of logical qubits. The 2D toric code, together with common operations are presented in Fig. 2.1.

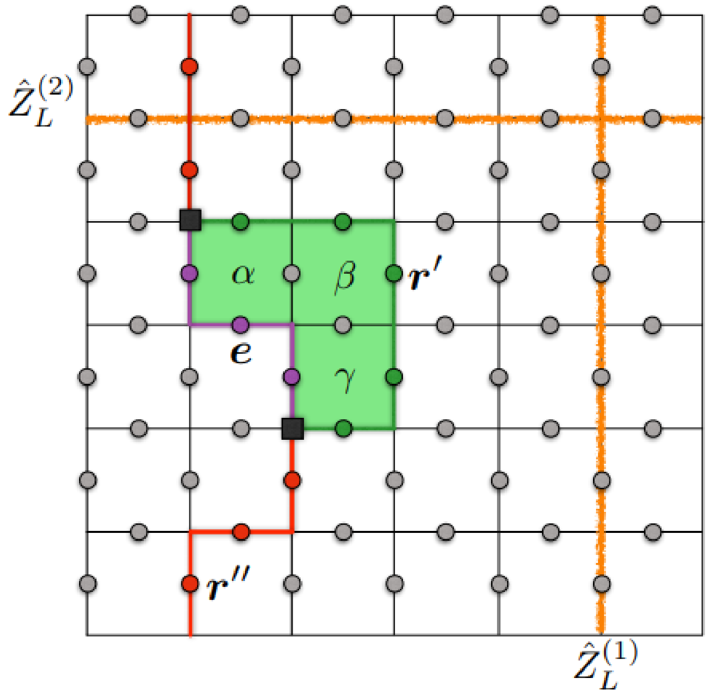

Figure 2.1: 2D Toric Code with certain operations. $Z_L^{(1)}$ and $Z_L^{(2)}$ represent logical operators. Error chain e depicted as a purple path with syndromes at its endpoints (squares). Green path r' is a recovery chain for e with the combined operator $\hat{Z}_\alpha \hat{Z}_\beta \hat{Z}_\gamma$ on the cycle $e \oplus r'$ as the recovery operation. The red recovery chain r'' is a non-trivial cycle and leads to a logical failure.[12]

2.2.2 Boltzmann Machine

A Boltzmann Machine is the type of a neural network which is stochastic recurrent. Stochastic means that some randomness is introduced into the network when it operates and recurrent means that a neural network forms a directed graph if we treat connections between the nodes as edges with weights (w_{ij}). Each node of the Boltzmann Machine can produce a binary output (s_i)

and the state of the whole network can be described by the "energy" E given by

$$E = -\left(\sum_{i<j} w_{ij} s_i s_j + \sum_i \theta_i s_i\right),$$

where θ_i is a bias of a node i in the global energy function. The negative of this parameter, i.e. $-\theta_i$ is the threshold for a node to activate.
Authors use the most common variant of a Boltzmann Machine which is called a restricted model. It means that we can see the network as consisting of several layers such that elements of each layer are not connected to each other. We can divide layers into visible ones which obtain data from certain external sources and hidden which collect data from other layers of the neural network. By drawing analogies from the Boltzmann distribution of energy, each node of a network can characterized by the probability that the node is active, given by

$$p_{i=act} = \frac{1}{1 + e^{\frac{-\Delta E_i}{T}}},$$

where

$$\Delta E_i = E_{i=off} - E_{i=act} = \sum_{i<j} w_{ij} s_j + \sum_{i>j} w_{ji} s_j + \theta_i.$$

2.2.3 Neural decoder

The authors propose the Boltzmann Machine consisting of three layers to find efficient decoders of error-syndromes for 2D toric codes. We call them the syndrome layer S ($\frac{N}{2}$ nodes), the error layer e (N nodes) and the hidden layer h (n_h nodes). The h layer is connected to other layers by symmetric edges. Each layer has an external bias called d for S, c for h and b for e. Weights of edges are represented as matrices W and U with zero diagonals. This architecture is presented in Fig. 2.2.
The neural network is trained on data consisting of pairs of error chains and corresponding syndromes which we denote $\mathcal{D} = (e, S)$. After the training, the hope is that the Boltzmann Machine will be able to model the probability distribution of errors and their syndromes well-enough and provide efficient error-recovery operations for syndromes not included in the training set.
The energy of this Boltzmann Machine is

$$E_\lambda(e, S, h) = -\sum_{ik} U_{ik} h_i S_k - \sum_{ij} W_{ij} h_i e_j - \sum_j b_j e_j - \sum_i c_i h_i - \sum_k d_k S_k,$$

and the probability distribution is

$$p_\lambda(e, S, h) = \frac{e^{-E_\lambda(e,S,h)}}{Z_\lambda},$$

where $\lambda = \{U, W, b, c, d\}$ and $Z_\lambda = \mathrm{Tr}_{\{h,S,E\}}\, e^{-E_\lambda(e,S,h)}$ is the partition function.
The probability distribution that we are interested in is $p_\lambda(e, S)$ and it emerges after summing over

the components of the hidden layer

$$p_\lambda(e,S) = \sum_h p_\lambda(e,S,h) = \frac{e^{-\mathcal{E}_\lambda(e,S)}}{Z_\lambda},$$

where $\mathcal{E}_\lambda(e,S)$ is the effective energy of the network and it can be calculated exactly, as well as probabilities $p_\lambda(e|h)$, $p_\lambda(S|h)$ and $p_\lambda(h|e,S)$. As a result, the final probability distribution produced by the neural network depends only on the parameters λ. Thus, the training of the network is basically choosing the parameters such that the resultant probability distribution is close to the probability distribution of the training data denoted $p_{data}(E,S)$. To quantify the distance between two probability distribution, the KullbackLeibler divergence is used which is given by

$$D_{KL}(p_{data}||p_\lambda) = -\frac{1}{|\mathcal{D}|}\sum_{\{e,S\}} \log p_\lambda(e,S).$$

The neural network will be driven by the minimization of D_{KL} by tuning the parameters λ what can be done by a gradient descent algorithm.

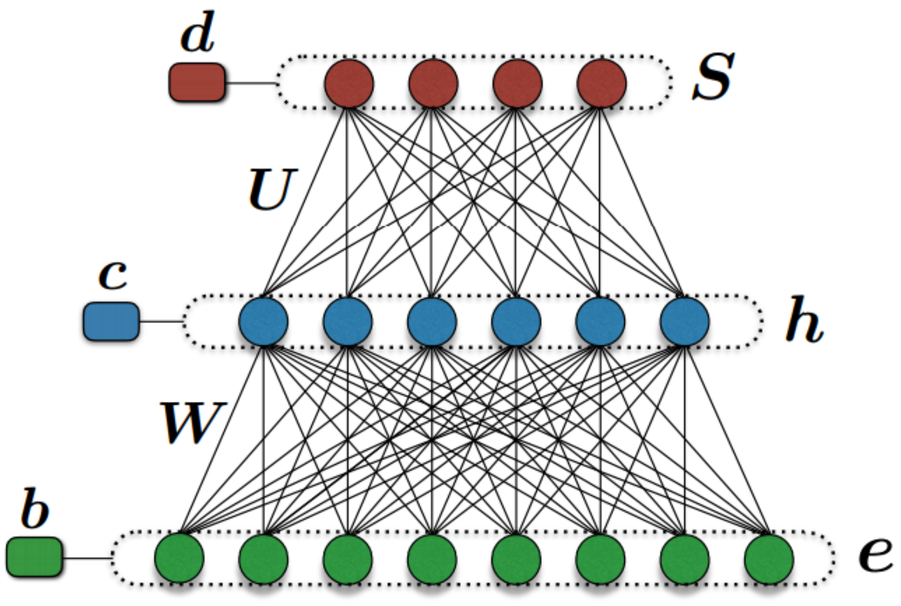

Figure 2.2: The Boltzmann Machine for a Neural Decoder [12]

2.2.4 Decoding Algorithm

Once we have trained our neural network so that it models the desired probability distribution $p_\lambda(e,S)$, we can use it to create a decoding algorithm for the error-correction purposes. Information that we get from the 2D toric code is an error syndrome S_0 which corresponds to some error e_0 which occurred and caused it. Therefore, we write $S_0 = S(e_0)$. By using our decoder, we would like to obtain some r such that $S(r) = S(e_0)$ and then hope that $r = e_0$. We should note that for reasonable accuracy we should use the neural network which was trained on a probability distribution which is similar to the actual error probability distribution in the system on which we perform the error correction. In practice it means that we should possess several Boltzmann Machines, each trained for a different error regime and we have to know the probability distribution of errors in our system. Then, we use the proper Boltzmann Machine and input data from our system into it.

The algorithm proposed by the authors is depicted in Fig. 2.3 and works as follows. In lines 1 and 2 we assume that we posses a syndrome S_0 which corresponds to some physical error e_0. In line 3 we initialize the Restricted Boltzmann Machine (RBM) with S_0 and random parameters in place of the unknown ones. Lines 4-7 represent a loop which works as long as the error matching the syndrome S_0 is found. It consists of sampling the values for the hidden layer from the probability distribution $p(h|e, S_0)$ and sampling candidate errors e from the probability distribution $p(e|h)$. Once e is found such that $S_0 = S(e)$, the algorithm terminates and we set $r = e$ as a physical error which caused the syndrome S_0.

We note that although the Boltzmann Machine is trained to reproduce $p_\lambda(e, S_0)$, we can sample from $p(h|e, S_0)$ and $p(e|h)$ by the property $p_\lambda(e, S_0) = p_\lambda(e|S_0)p(S_0)$ and by keeping S_0 fixed. The authors note that similar results can be obtained by using the Monte Carlo methods, however, they require sampling algorithms which are specific to a particular stabilizer structure or multicanonical methods which use specially modified probabilities to enforce certain behaviors. Neural decoders does not require any of these.

Algorithm 1 Neural Decoding Strategy

```
1: $\boldsymbol{e}_0$: physical error chain
2: $\boldsymbol{S}_0 = \boldsymbol{S}(\boldsymbol{e}_0)$                         ▷ Syndrome Extraction
3: RBM $= \{\boldsymbol{e}, \boldsymbol{S} = \boldsymbol{S}_0, \boldsymbol{h}\}$              ▷ Network Initialization
4: while $\boldsymbol{S}(\boldsymbol{e}) \neq \boldsymbol{S}_0$ do                               ▷ Sampling
5:     Sample $\boldsymbol{h} \sim p(\boldsymbol{h} \,|\, \boldsymbol{e}, \boldsymbol{S}_0)$
6:     Sample $\boldsymbol{e} \sim p(\boldsymbol{e} \,|\, \boldsymbol{h})$
7: end while
8: $\boldsymbol{r} = \boldsymbol{e}$                                          ▷ Decoding
```

Figure 2.3: The Algorithm for Neural Decoding [12]

2.2.5 Results

The authors test their approach as follows. Training data sets are generated for various probability regimes of the phase-flip channel, namely $p = \{0.6, 0.5, ..., 0.15\}$. For each regime, a Boltzmann Machine is trained and then the algorithm explained in the previous section is used. It is executed on syndromes of errors chains which are generated and form a test set $\mathcal{T}_p = \{e_k\}_{k=1}^{M}$. It was empirically verified that the number of iterations of the algorithm is usually of order 10^2. Using the properties of 2D toric codes, authors are able to calculate the logical failure probability which is defined as $p_{fail} = \frac{n_{fail}}{M}$, where n_{fail} is the number of non-trivial cycles. This quantity is also calculated for the standard algorithm used for error recovery in 2D Toric Codes which is the Minimum Weight Perfect Matching (MWPM) algorithm. The MWPM algorithm finds a recovery operation which consists of the shortest sequence of edges that connect defects of the same type. The critical difference between the neural decoder and the MWPM algorithm is that the latter does not rely on the assumption of knowing the probability distribution of errors. As it can be seen in Fig. 2.4, both methods offer a comparable probability of failure up to a certain threshold value for the probability of error. From that point onwards, the standard MWPM turns out to be better.

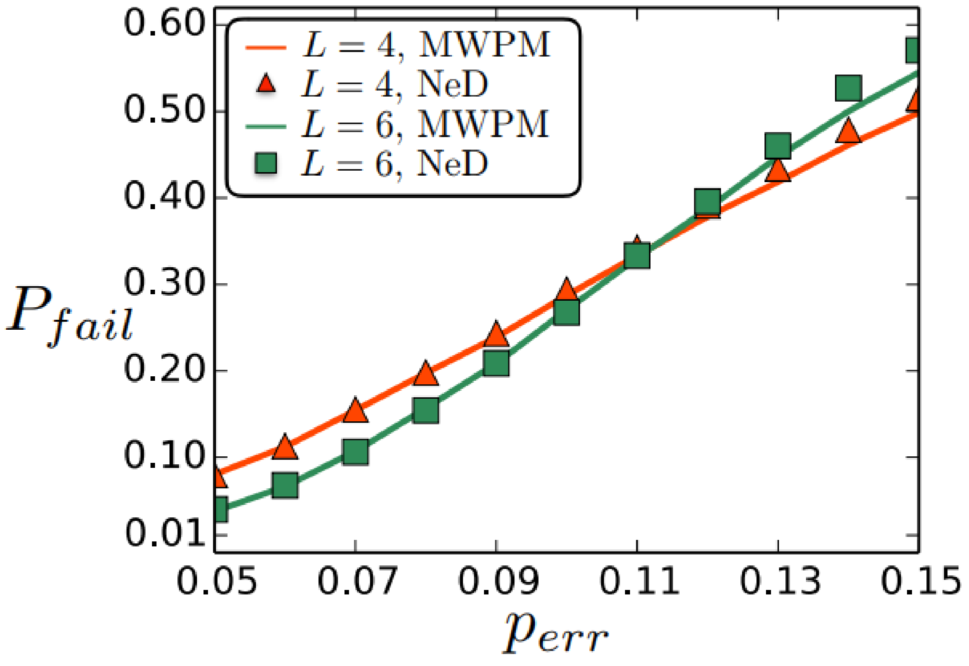

Figure 2.4: Probabilities of a logical failure as a function of elementary error probability for MWPM (lines) and the neural decoder (markers) of size $L = 4$ (red) and $L = 6$ (green). [12]

2.2.6 Summary

Authors of the paper provided the proof of concept of classical machine learning applied to the problem in the quantum error correction. They explained how can we use neural networks to create decoders of error syndromes. Their algorithm matched accuracy of another standard method called Minimum Weight Perfect Matching (MWPM) for a wide range of parameters in a simple model of a phase-flip channel. It should be noted however, that the neural network approach is potentially much easier transferable to other, more complex models which gives it a clear advantage over other methods which require more tailored algorithms. When it comes to disadvantages and limitations of Boltzmann Machines, we should mention the assumption that the noise regime which introduces errors to the system is well-known. Nevertheless, the application of machine learning for quantum error correction seems to have a very bright future and, as authors claim, there are still many improvements and tricks available within the machine learning framework which can be implemented to make this approach much better and superior to other methods currently in use.

Chapter 3

Quantum Enhanced Learning

3.1 Overview

Quantum Enhanced Learning is mostly understood as using a quantum computer together with certain quantum algorithms or methods to achieve goals and solve problems that one would normally attempt to solve with classical machine learning methods. Of course, we would expect to benefit from the quantum advantage somehow and obtain results which are better under a certain set of criteria. Since many tasks in machine learning rely on searching through a space of various objects, the natural example is substituting a classical search algorithm with the quantum alternative which is the Grover Algorithm. It immediately gives us a quadratic improvement in computational complexity. However, applying quantum methods can often be performed in a more subtle and sophisticated manner which sometimes require serious reformulation and changes in the way that we think about a problem. A good example is the novel approach to Quantum Perceptron Models that uses a new machine learning algorithm which is tailored to utilize speedups that can be offered by a quantum computer. It will serve as a representative for the Quantum Enhanced Learning category and will be presented in this chapter in details. Moreover, we should also note that quantum regime may potentially lead us to routines which have no classical counterparts. By Quantum Enhanced Learning we may also refer to certain mathematical techniques that emerged from studying the quantum theory and turned out to be applicable and useful for improving classical methods and algorithms in machine learning. These include tensor networks [13] and renormalization [14].

3.2 Example - Quantum Perceptron Models

We will present two proposals for a quantum perceptron learning problem.
This chapter is based on the paper:
N. Wiebe, A. Kapoor, K. M. Svore; Quantum Perceptron Models (2016).

3.2.1 Perceptron and Classification Problem

One of the most important problems in Machine Learning is the problem of classification. Solving the problem of classification of certain objects means that we possess a function (usually called a classifier) which takes an object as an input and outputs the class that this object belongs to based on some set of criteria. One of the main approaches to this problem is by using a perceptron. Perceptron is an algorithm for supervised learning (i.e. learning based on a labeled training set of examples) of binary classifiers (functions with binary codomain). In other words it is a method of obtaining a classifier for the problem of classification into two classes. To obtain a classifier we only need to provide a perceptron with a set of examples $\{\phi_1, ..., \phi_n\} \in \mathbb{R}^D$ labeled by $\{y_1, ..., y_N\}, y_i \in \{+1, -1\}$ and the set of classification criteria need not be specified. A classifier f is defined as

$$f(x) = \begin{cases} 1 & \text{if } y_i w^T \phi_i > 0 \\ 0 & \text{otherwise} \end{cases},$$

for some hyperplane represented by a vector w. One of the powerful features of the perceptron approach is that there exist algorithms for which both computation complexity and error-rate are bounded. In the classical setting, assuming that the training set consists of unit vectors $\phi_i \in \mathbb{R}^d$ which are separated by some margin γ, the computation complexity is linear ($O(n)$), with worst-case scenario when all of the examples are processed) and the statistical efficiency can be $O(\frac{1}{\gamma^2})$ which means that at most $O(\frac{1}{\gamma^2})$ are made in classification. It turns out that both measures can be improved in the quantum setting which will be shown in the following sections. Presented methods will require introducing the concepts of feature and version spaces.
Usually, the problem of classification is considered in the feature space representation. We say that each object that we want to classify is characterized by a vector called a feature vector (ϕ_i). Each entry of the feature vector represents a certain feature of the object considered and naturally we may imagine that objects are points in a D-dimensional feature space. The problem of classification is then equivalent to choosing a hyperplane in this space which divides objects into two classes. One may also consider a dual approach, called a version space representation. A version space is the set of all possible hyperplanes which provide a perfect separation of training data. In this case, our problem can be viewed as the problem of finding a hyperplane from that set (a feasible hyperplane) by searching the space of all possible hyperplanes. In the graphical representation, we may depict data points as hyperplanes and hyperplanes as points. Then, data points

represented as hyperplanes introduce constraints on the feasible set of classifiers. Both views are presented in Fig. 3.1 with Bayes and SVM (Support Vector Machine) as examples of classifiers.

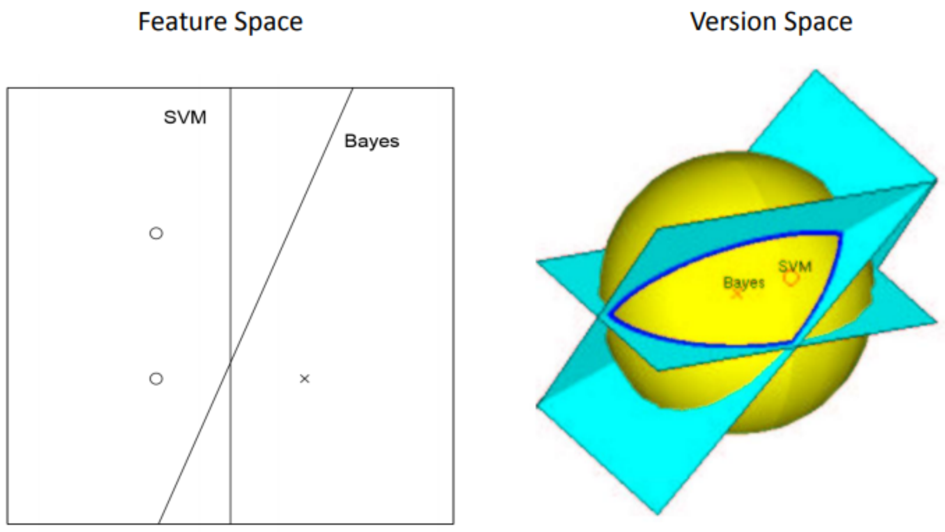

Figure 3.1: Feature and version spaces. [16]

3.2.2 Quantum Feature Space Perceptron

In this section we present a quantum model of a perceptron training proposed by the authors. It should be pointed out that it slightly differs from the classical approach in which examples are provided one by one (online). For the quantum approach we assume that examples are sampled uniformly at random from the predefined set of examples. This approach allows us to benefit more from the quantum advantage.

The training set consists of N examples represented by unit vectors $\phi_1, ..., \phi_N$. Each of them, together with a corresponding label $y_i \in \{-1, 1\}$ will be encoded into one of N basis vectors $\Phi_1, ...\Phi_N$ according to the $(B+1)$-bit representation of the tuple (ϕ_j, y_j). To encode a vector of example $\phi_j \in \mathbb{R}^D$ we would concatenate binary representations of its components and a binary value representing a label (0 for $y = -1$ and 1 for $y = 1$). The resultant string of bits would be then converted to a decimal number Q and represented as a basis vector $\Phi : [\Phi]_i = \delta_{iQ}$. Note that it means that encoded vectors are not in the same Hilbert space as corresponding training vectors. Encoded vectors live in a larger Hilbert space. By Φ_0 we will denote a unit vector representing an empty register.

We assume that a learning algorithms accesses training data by performing the following unitary operation U

$$U[u_j \otimes \Phi_0] = u_j \otimes \Phi_j$$

$$U^\dagger[u_j \otimes \Phi_j] = u_j \otimes \Phi_0,$$

where $\{u_j : j = 1, ..., N\}$ is an orthonormal basis of quantum state vectors which serve as addresses of training data in a database. It is worth noting that due to linearity of U, we can access each training vector simultaneously and obtain a superposition of states: $U \sum_{j=1}^{N} u_j \otimes \Phi_0 = \sum_j u_j \otimes \Phi_j$.

To test whether a perceptron algorithm classified a certain example correctly, we introduce a unitary operator $\mathcal{F}_w$ dependent on current weights w used by the perceptron which acts as follows

$$\mathcal{F}_w(\Phi_j) = (-1)^{f_w(\phi_j, y_j)}(\Phi_j),$$

where $f_w(\phi_j, y_j)$ is a Boolean function which takes the value of 1 if and only if the vector ϕ_j is misclassified. In other words, it applies a negative phase to the encoding of a misclassified training vector. Therefore, the full procedure can be expressed as an operator F_w defined as

$$F_w = U^\dagger(\mathbb{1} \otimes \mathcal{F}_w)U.$$

Now, we can make use of the quantum search routine (Grover's algorithm) to look for misclassified vectors by recognizing a negative phase. It can be done by using the Grover's algorithm with parameters $U_{targ} = F_w$ and $U_{init} = 2\psi\psi^\dagger - \mathbb{1}$, where $\psi = \Psi := \frac{1}{\sqrt{N}} \sum_{j=1}^{N} u_j$. This step lets us obtain a quadratic speedup when compared to the classical approach. The authors makes this statement precise in the theorem which follows.

Theorem 1. *Given a training set that consists of unit vectors $\Phi_1, ..., \Phi_N$ that are separated by a margin γ in feature space, the number of applications of F_w needed to infer a perceptron model, w, such that $P(\exists j : f_w(\phi_j) = 1) \leq \epsilon$ using a quantum computer is N_{quant} where*

$$\Omega(\sqrt{N}) \ni N_{\text{quant}} \in O\left(\frac{\sqrt{N}}{\gamma^2}\log(\frac{1}{\epsilon\gamma^2})\right).$$

3.2.3 Quantum Version Space Perceptron

Similarly as in the previous section, we will be formulating our task as a search problem. In the Quantum Version Space Perceptron we will be searching for a feasible separating hyperplane. Initially, we sample K candidate hyperplanes $w_1, ..., w_k$ from a spherical Gaussian distribution $\mathcal{N}(0, 1)$. We choose K to be large enough so that we have a very high chance of having at least one feasible hyperplane among the generated ones. To make this statement precise, the authors propose the following theorem.

Theorem 2. *Given a training set that consists of d-dimensional unit vectors $\Phi_1, ..., \Phi_n$ with labels $y_1, ..., y_N$ that are separated by a margin of γ in feature space, then a D-dimensional vector w sampled from $\mathcal{N}(0, 1)$ perfectly separates data with probability $\Theta(\gamma)$.*

As a consequence, if we also use the amplitude amplification method to improve the chance of finding a vector in the version space, our final quantum algorithm will on average require $O(\frac{1}{\sqrt{\gamma}})$.

We can use the Grover's search algorithm to find a feasible hyperplane as follows. Assuming that $K = 2^m$ and that we already sampled perceptron vectors (hyperplanes) $w_1, ...w_K$ we denote encodings of binary representations of those vectors by $W_1, ..., W_K$ and an empty register by W_0. We will generate vectors by a unitary operator V defined as

$$V(u_j \otimes W_0) = (u_j \otimes W_j).$$

By applying V and a Hadamard operator we can create a state Φ which is the uniform superposition of our random vectors and can be written as

$$\Psi := \frac{1}{\sqrt{K}} \sum_{k=1}^{K} u_k \otimes W_k.$$

The state Ψ defined above serves as an initial state for the Grover's search algorithm. Similarly as in the previous section, we also define an operator $\hat{\mathcal{F}}_{\phi,y}$ which acts on a vector in the version space as

$$\hat{\mathcal{F}}_{\phi,y}(W_j) = (-1)^{1+f_{w_j}(\phi,y)}(W_j).$$

However, it may happen that a vector is outside the version space and then it cannot be rotated by $\hat{\mathcal{F}}_{\phi,y}$ towards the boundary of the version space. It is caused by the fact that $\hat{\mathcal{F}}_{\phi,y}$ checks only one of the half-space inequalities which put constraints on feasible hyperplanes. To fix this problem, we can define another operation $\hat{G}$ which is based on $\hat{\mathcal{F}}_{\phi,y}$ and serves as a reflection about the version space.

$$\hat{G}(u_j \otimes W_0) = (-1)^{1+(f_{w_j}(\phi_1,y_1) \vee ... \vee f_{w_j}(\phi_N,y_N))}(u_j \otimes W_0)$$

The operation $\hat{G}$ requires $O(N)$ elementary quantum gates and queries to $\hat{\mathcal{F}}_{\phi}$.
The operations defined above allow us to amplify the margin between the two classes from initial γ to $\sqrt{\gamma}$ which is defined precisely in the theorem that follows.

Theorem 3. *Given a training set that consists of unit vectors $\Phi_1, ..., \Phi_N$ that are separated by a margin γ in feature space, the number of queries to $\hat{\mathcal{F}}_{\phi,y}$ needed to infer a perceptron model with probability at least $1-\epsilon$, w, such that w is in the version space using a quantum computer is N_{quant} where*

$$N_{\text{quant}} \in O\left(\frac{N}{\sqrt{\gamma}} \log^{3/2}\left(\frac{1}{\epsilon}\right)\right).$$

To compare, the classical algorithm requires $O(N \log(1/\epsilon)/\gamma)$ operations. Therefore, we obtain a better complexity in the quantum case provided that $\frac{1}{\gamma} \gg \log(1/\epsilon)$. The conclusion is that using the quantum algorithm does not only improve the computational complexity but also decreases the number of the training vector that need to be queried in the learning process (the perceptron becomes more "perceptive").

3.2.4 Summary

The authors proposed two ways of incorporating quantum routines into the model of a perceptron. Each of the methods presented yields speedups when compared to their classical counterparts and each in a different way. In case of a Feature Space Quantum Perceptron, we obtained a quadratic speedup in terms of the size of the training data, whereas the Version Space Quantum Perceptron offers a quadratic reduction in the scaling of the training time. Since the perceptron training is the core of classical machine learning, both results are important and show that there is much potential in the quantum approach to Machine Learning. The authors point out that apart from their two proposed model, there may still exist many interesting approaches to Quantum Perceptrons. Some of them may not have classical counterparts and provide unexpected results.

Chapter 4

Summary

We presented two instances of Quantum Machine Learning. The first of them uses a classical computer together with classical machine learning methods to solve a problem relevant to quantum error correction. Its performance is comparable to the most commonly used algorithm for the wide range of parameters and then drops after passing a certain threshold. However, the machine learning approach described in this paper is only the proof of concept. It does not utilize more sophisticated tricks that are usually used in other machine learning applications. It suggests a natural direction for further research in which we could investigate how much improvement could be made if we tuned the model more. Apart from that, the neural decoder works with the different set of assumptions than a standard method. In particular, the knowledge of the error-regime that generates defects in the 2D toric code is required. Therefore, we may wonder if this model may become superior to the standard method in certain applications where such knowledge is available. The second instance of Quantum Machine Learning explained in this paper is the model of a perceptron that can be implemented on a quantum computer and operate on both classical and quantum data. It offers two approaches to port the concept of a perceptron learning to the quantum setting. Both of them make use of the Grover's search algorithm which offers a quadratic speedup over the classical search. As a result, the first method, the Quantum Feature Space Perceptron, benefits from it by offering a quadratic speedup in the complexity of a perceptron training process when compared to the classical learning algorithm. The second model, namely the Quantum Version Space Perceptron, exposed a different type of improvement which is the quadratic reduction of the number of interactions with the training set, while keeping the same computational complexity. Having two ways to look at the perceptron training, it would be interesting to investigate other possible models in the future work.

To sum up, we see that Quantum Machine Learning seems to be a very promising direction of research. It uses tools from both classical machine learning and from Quantum Information Theory in various configurations and settings to solve a wide range of problems in physics, engineering and computer science.

Bibliography

[1] L.K. Grover, A fast quantum mechanical algorithm for database search, Proceedings, 28th Annual ACM Symposium on the Theory of Computing, (1996) p. 212

[2] C. H. Bennett and G. Brassard, Quantum cryptography: Public key distribution and coin tossing, In Proceedings of IEEE International Conference on Computers, Systems and Signal Processing, volume 175, page 8. New York, (1984).

[3] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao and K. Zieba, End to End Learning for Self-Driving Cars, (2016)

[4] F. Sebastiani, Machine learning in automated text categorization, ACM computing surveys (CSUR) 34, no. 1: 1-47 (2002).

[5] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe and S. Lloyd, Quantum Machine Learning Nature, 549, 195-202 (2017).

[6] C. E. Granade, C. Ferrie, N. Wiebe, and D. G. Cory, Robust online Hamiltonian learning, New J. Phys., 14(10):103013 (2012).

[7] M. Sasaki, A. Carlini, and R. Jozsa. Quantum template matching, Phys. Rev. A, 64:022317 (2001).

[8] A. Hentschel and B. C. Sanders, Machine learning for precise quantum measurement, Phys. Rev. Lett., 104:063603 (2010).

[9] E. Zahedinejad, J. Ghosh, and B. C. Sanders, Designing high-fidelity single-shot three-qubit gates: A machine learning approach, arXiv:1511.08862 (2015)

[10] J. Carrasquilla and R. G. Melko, Machine learning phases of matter, arXiv:1605.01735 (2016).

[11] G. Torlai and R. G. Melko, Learning thermodynamics with Boltzmann machines, arXiv:1606.02718 (2016).

[12] G. Torlai and R. G. Melko, A Neural Decoder for Topological Codes Physical Review Letters, 119, 030501 (2017).

[13] A. Anandkumar, R. Ge, D. Hsu, S. M. Kakade, and M. Telgarsky, Tensor Decompositions for Learning Latent Variable Models, pages 19 38, Springer International Publishing, Cham (2015)

[14] C. Beny, Deep learning and the renormalization group, arXiv:1301.3124 (2013)

[15] N. Wiebe, A. Kapoor, K. M. Svore, Quantum Perceptron Models (2016).

[16] Minka, Thomas P., A family of algorithms for approximate Bayesian inference, PhD thesis, Massachusetts Institute of Technology (2001).