CS 229, Spring 2021
Applications of Quantum Computing in Machine Learning

Sasha Lazarevic (lzrvc@stanford.edu)

Quantum computing is a promising new field of computation that relies on quantum phenomena like superposition, entanglement and interference, and can provide speedup over classical algorithms in quantum simulations, optimization and machine learning applications. It is believed that large-scale fault-tolerant quantum computers will be available in 10 to 20 years, but with the recent improvements of the existing quantum computing platforms, it is becoming very valuable to understand how to use quantum computers in the context of machine learning.

1 Introduction

Quantum Computing and Machine Learning are two computational disciplines that are increasingly merging in two ways:

1) Application of machine learning for quantum computing, where ML techniques are used for the purposes of :

- design of classifiers for the estimations of quantum states
- quantum circuit optimization and design
- error mitigation methods

These techniques are very promising , especially in the current era of Noisy Intermediate-Scale Quantum (NISQ) computers, where quantum computers are still error prone, with short coherence times and limited qubit inter-connectedness. Although important, these techniques are not part of this course project.

2) Quantum-Enhanced Machine Learning, which exploits quantum phenomena to improve performance of ML algorithms in terms of computational speed and capacity of machine learning models. This is a very exciting and active area of research, and I will describe in my course project the use of Variational Quantum Algorithm, which is hybrid in nature as it consists of classical and quantum components.

When discussing quantum machine learning implementations, it is important first to denote the basic underlying quantum algorithms, which have been proven to provide different levels of speedup comparing to respective classical algorithms.

Machine learning techniques enhanced with these quantum algorithms are therefore expected to deliver a similar speedup. These are:

- Quantum Phase Estimation (QPE)
- Harrow-Hassidim-Lloyd algorithm (HHL) for solving matrix inversion problem
- Quantum Fourier Transform (QFT)
- Amplitude Amplification algorithm (aka Grover's search)
- Quantum Amplitude Estimation algorithm (QAE)
- Quantum Amplitude Estimation without Phase Estimation (QAE w/o PE)
- Quantum Circuit Born Machine (QCBM)
- Variational Quantum Algorithms (VQA) - a family of hybrid quantum-classical algorithms, which also includes Variational Quantum Eigensolver (VQE) and Quantum Approximate Optimization Algorithm (QAOA)
- Quantum Boltzmann Machine (QBM) and Variational QBM (VarQBM)

These quantum algorithms are used as building blocks in many quantum-enhanced machine learning techniques. The intention is to identify computationally intensive routines and to design a specific way how these routines can be more efficiently implemented on quantum computers.

As of 2021, we can list successful theoretical and practical implementations of quantum machine learning algorithms in small-scale environments:

- Linear regression has been implemented using QPE and HHL
- Ridge regression, where QPE and QFT (with or without Grover's search) are used to find the optimal value of the regularization parameter
- Perceptron can be implemented with Grover's search[1]
- Support Vector Machine has been implemented with HHL algorithm
- k-means clustering and k-nearest neighbors classification can be improved with Grover's search [2]
- Principal component analysis has been implemented with QPE[3] and VQA[4]

[1]Nathan Wiebe, Ashish Kapoor, Krysta M. Svore 'Quantum Perceptron Models' https://arxiv.org/pdf/1602.04799.pdf

[2]Nathan Wiebe, Ashish Kapoor, Krysta M. Svore 'Quantum Algorithms for Nearest-Neighbor Methods for Supervised and Unsupervised Learning' https://www.researchgate.net/publication/259625322$_{Quantum_Nearest-Neighbor_Algorithms_for_Machine_Learning}$

[3]Seth Lloyd, Masoud Mohseni and Patrick Reben-trost, "Quantum principal component analysis" https://www.nature.com/articles/nphys3029

[4]M. Cerezo, Kunal Sharma, Andrew Arrasmith and Patrick J. Coles 'Variational Quantum State Eigensolver' https://arxiv.org/pdf/2004.01372.pdf

- Autoencoders have been implemented with VQA[5]
- Restricted Boltzmann Machines have been implemented as QBM on quantum annealer or Var-QBM on circuit-based quantum computers
- Deep neural networks can be implemented as hybrid models with VQA
- Convolutional neural networks have been implemented with VQA[6]
- GANs training has been improved in a way that Generator or Discriminator (or both) are using associative NNs implemented with QBMs
- Synthetic data has been generated with QCBM
- Model-based reinforcement learning can be enhanced with Grover's algorithm to find rewarding action-sequences[7]
- RL Q-learning has been successfully tested with QBM, which generates values of the Q-value function
- Monte-Carlo sampling, which is a multi-purpose tool, and is used for policy evaluation in RL can be implemented with QAE

As the research with underlying foundational algorithms advances, downstream quantum machine learning algorithms also change. For example, in case of SVM the focus is shifting away from Grover's search towards HHL, and in case of RL Q-learning towards QBM.

Another important element is the underlying quantum computing platform, as this determines substantially the choice of the foundational quantum algorithms. In this context, we can distinguish three types of quantum hardware:

- Adiabatic quantum computers
- Universal, fault tolerant quantum computers
- Noisy Intermediate-Stage (NISQ) Quantum computers

Scientific community agrees that Adiabatic quantum computers don't provide proof of their quantum advantage. As for the universal, fault tolerant quantum computers are concerned, the current engineering challenges won't be resolved within the next 10 years. Thus, the research community has focused on algorithms that can efficiently run on the current, NISQ computers with 50+ qubits. As of 2021, the focus is on the variational quantum algorithms that can efficiently run on these platforms.

[5] J. Romero, J. P. Olson, Alan Aspuru-Guzik 'Quantum autoencoders for efficient compression of quantum data' https://arxiv.org/pdf/1612.02806.pdf

[6] I. Cong, S. Choi, M. Lukin 'Quantum convolutional neural networks' https://www.nature.com/articles/s41567-019-0648-8

[7] Vedran Dunjko, Jacob M. Taylor and Hans J. Briegel 'Quantum-enhanced machine learning' https://arxiv.org/pdf/1610.08251.pdf

2 Basics of quantum computation

As said in the introduction, quantum computation is based on exploiting quantum phenomena like superposition, entanglement and interference.

This work does not describe their details or their mathematical modelling (also known as quantum mechanics) or the use of quantum mechanics in the information science. There is a number of great literature on this subject that is recommended to those who need in-depth understanding of these aspects. One of them is Quantum Computation and Quantum Information by Michael Nielson and Isaac Chuang.

2.1 Hardware and Simulators

As of 2021, there are two mainstream hardware implementations of quantum computers. The most popular are those based on superconducting qubits, where electrical circuits coupled with a Josephson's Junction at temperatures near absolute zero exhibit quantum behaviour and can be controlled by microwaves to induce a charged state and superposition, create entanglement, change the phase, or perform readouts. Another efficient implementation is based on trapped ions, where ions are kept in a magnetic field and are controlled with lasers. Few other technologies are being explored and the interested readers will find many resources on this subject.

Most of these platforms provide a level of abstraction where a programming framework will include a compiler and transpiler that can adapt the code to the underlying hardware and its specific architecture.

Local simulators are also an efficient starting point for learning and developing quantum algorithms. Quantum circuits of up to 15 qubits can be easily simulated on an ordinary laptop. When we approach simulations of 40 qubits and more, the requirements for hardware resources in terms of memory dramatically increase. But for the small-scale experiments, the only important difference between a simulator and a real quantum computer is noise, which needs to be mitigated through the downstream algorithm or using Bayesian inference of the posterior probability distribution of results. This project's work is done on the local simulator using IBM Qiskit python-based framework.

2.2 Quantum Gates and Quantum Circuits

Quantum operations are programmatically implemented as gates. These gates are mathematically defined as unitary matrices and can be classified as single-qubit or controlled multi-qubit operations. Combined together, they will create circuits and computational blocks. Usual way of combining the quantum gates is firstly by initializing qubits (to ket-zero or ket-one), then applying superposition and entanglement. Thus, the quantum system is ready to receive data. Next, one will

use rotational operations to encode the data or parameters as amplitudes (angles). Other parameterized and non-parameterized operations can then be performed in order to evolve the entangled quantum state through interference phenomena like phase kick-back or amplitude amplification to the state which represents the solution to our problem. In the end, we apply measurement operations to get the results from these qubits and learn the probability distribution of our solution.

3 Variational Quantum Algorithms

As mentioned already in the introduction, variational quantum algorithms are the most promising approach for the near-term quantum computation. The main advantage is that they allow iterative processing on a quantum computer, which reduces the depth of the quantum circuit. This is useful as NISQ computers, due to gate errors and short decoherence times are not able to execute correctly deep circuits beyond around a hundred of mixed single- and controlled qubit gates.

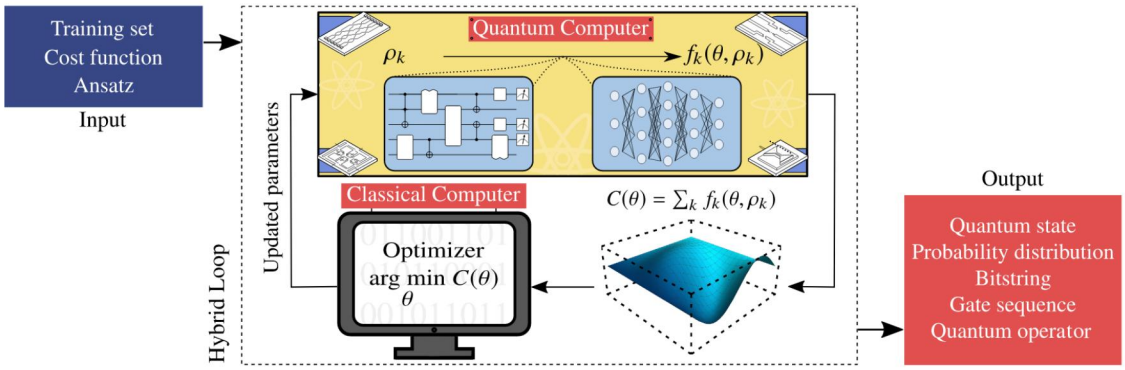

Figure 1: Figure 1. Schematic diagram of a VQA[8]

The design and implementation of a VQA includes several steps:

- Selection of the cost function $C(\theta)$ corresponding to the solution of our problem
- Selection of the ansatz, or in other words the structure of our quantum circuit
- Decisions on how to encode the trainable parameters θ
- Design and implementation of the functions that will encode feature mapping
- Decisions what the results will be used for (classification or further layers)
- Choice of the technique for the derivation of the cost function to calculate the gradient of the quantum circuit

An ansatz can be general or problem specific, it can also be fixed or dynamically adaptive[9], or hybrid where it could include classical feature mapping functions.

[8] Group of authors, 'Variational Quantum Algorithms' https://arxiv.org/pdf/2012.09265.pdf

[9] Group of authors, 'Variational Quantum Algorithms' https://arxiv.org/pdf/2012.09265.pdf

In the context of machine learning, VQA can be used for the implementation of classifiers, autoencoders, generative models or CNNs.

4 Implementation of quantum-classical neural network

4.1 Simple quantum classifier

VQA can be used to create a very simple classifier and to experiment with its gradient. My script `py_simple_clf_numpy.py` implements this classifier with a three-qubit circuit and three parameters used for the prediction. These parameters θ_1, θ_2 and θ_3 are initialized randomly and then encoded as angles using the rotational operators on the three entangled qubits. The qubits are then disentangled and the measurement of the first qubit is performed to get the probability distribution, which is the function of this entangled three-qubit system state.

The classification problem is simple: our probability distribution needs to reach the value close to $(0, 1)$ in order to predict correctly the target label (1), or $(1, 0)$ to predict (0). The circuit initially produces equally distributed probability values of $(0.5039, 0.4961)$. The small difference in these two values is due to the randomness integrated in the quantum simulator to approximate the behavior of a real quantum computer. As we are measuring the state of only one qubit, the result will consist always of two numbers.

After the calculation of the value of the quadratic loss function and its gradient, we then proceed to calculate the value of the gradient of the quantum circuit with respect to its parameters. Our prediction can be defined as:

$h(\Theta) = Q(\Theta)$ where $\Theta = (\theta_1, \theta_2, \theta_3)$ and

$Q(\Theta) = H(q_0)\, CNOT(q_0, q_1, q_3)\, RY(\Theta)\, CNOT(q_0, q_1, q_3)\, H(q_0)\, Readout(q_0)$
where q_0, q_1, q_2 are qubits

Our loss function is $J(\Theta) = \frac{1}{2}(h(\Theta) - y)^2$ and its gradient is:

$\nabla_\Theta J(\Theta) = (h(\Theta) - y))\, \nabla_{Theta} Q(\Theta)$

A common way to calculate $\nabla_\Theta Q(\Theta)$ is using a parameter shift rule[10]:

$\frac{\partial Q}{\partial \Theta} = \frac{Q(\Theta+s) - Q(\Theta-s)}{2s}$ where s is shift coefficient and typically has a value of $\frac{\pi}{2}$.

[10] Qiskit documentation :
https://qiskit.org/textbook/ch-machine-learning/machine-learning-qiskit-pytorch.html

Numerator $Q(\Theta + s) - Q(\Theta - s)$ is a vector and we will divide this vector by its 2-norm (Euclidean distance) in order to normalize it, instead of 2s. This division is commonly used in practice and serves to stabilize the calculations. In the end, gradient $\nabla_\Theta J(\Theta)$ is used to update Θ with a fixed learning rate and the algorithm is run for 200 iterations.

We can see in Figure 2 that the cost function converges to a value close to zero, as our predicted probability for the label (1) goes up to 1.

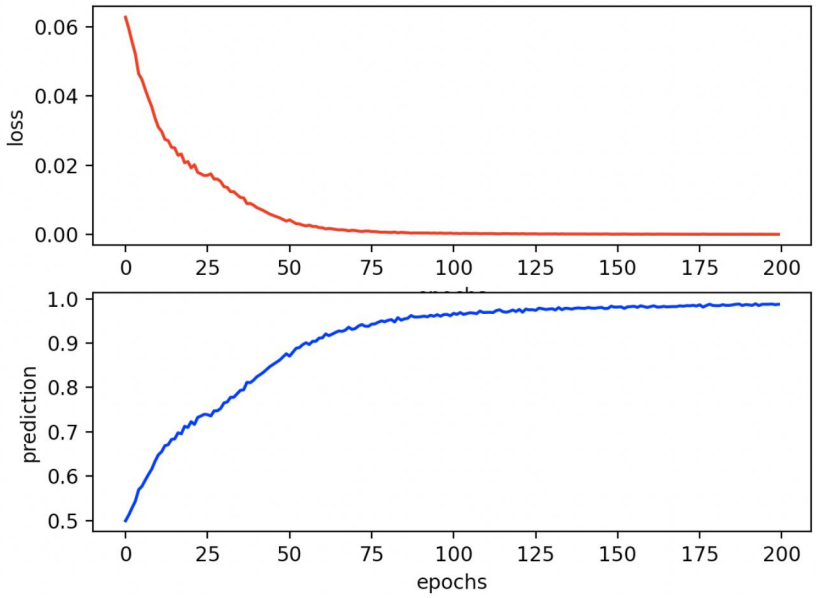

Figure 2: Loss and prediction of the simple quantum classifier

4.2 Hybrid classical-quantum classifier

We can scale and use this method to implement the quantum classifier on top of a classical neural network. The example is MNIST recognition of digits. The classical part is made of two layers and it builds upon my implementation of ps2.4 (Spring 2021).

What is new in this work is the implementation of the hybrid algorithm in numpy, where one can do low level development and experimentation of the network gradient, and monitor the inner behavior and vulnerability and stability issues of the end-to-end training.

The loss function for this case is again the quadratic loss. Quadratic loss is suitable for the probabilistic prediction situations with no outliers, like we have in MNIST dataset. The graph of the calculation of the overall gradient is described in the Figure 3:

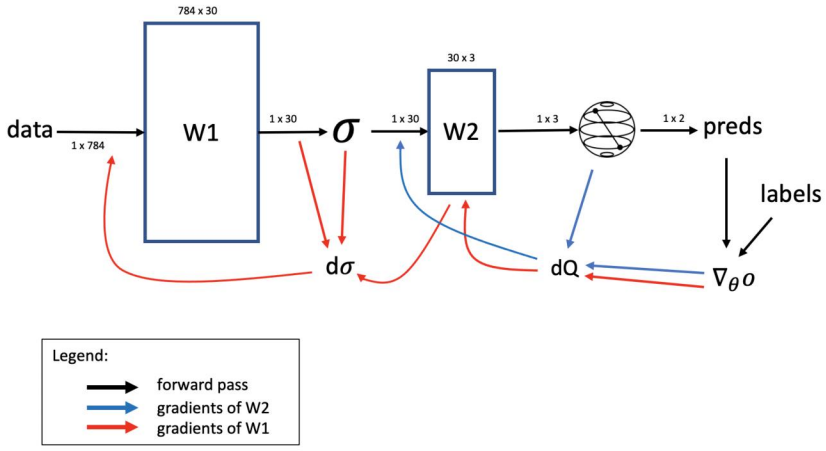

Figure 3: Gradient calculation graph

We can see from the graph that this implementation uses SGD. In case of mini- or batch gradient descent, the quantum part will have to be done in small, reduced batches limited to the capacity of the available quantum backend, or will be done sequentially. The overall gradient of the cost function can be then expressed as:

$$\nabla_\theta J = \frac{\partial C}{\partial Q}\frac{\partial Q}{\partial o}\nabla_\theta o,$$

where C denotes the classical layers, Q denotes the quantum layer (or circuit) and o denotes the output.

The setup of the quantum circuit and its' parameter initialization, which is also called quantum ansatz will look like the following:

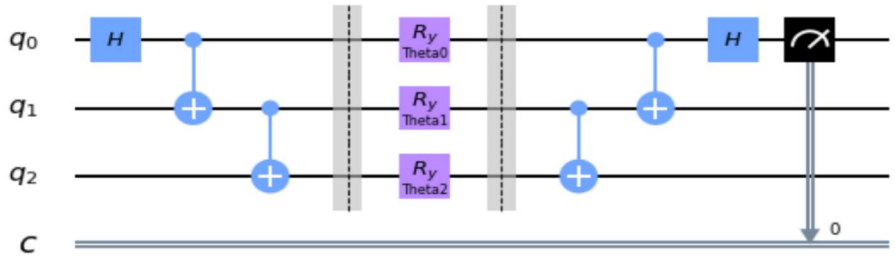

Figure 4: Quantum ansatz with three entangled qubits

For the training purposes I have selected a subset of the dataset with only two classes 0 and 1, and 600 training examples in total. For the evaluation of performance I'm using accuracy metric, which is appropriate as the data belonging to these two classes is equally distributed. The achieved test accuracy is 0.96825 and the performance of the algorithm is given in Figure 5:

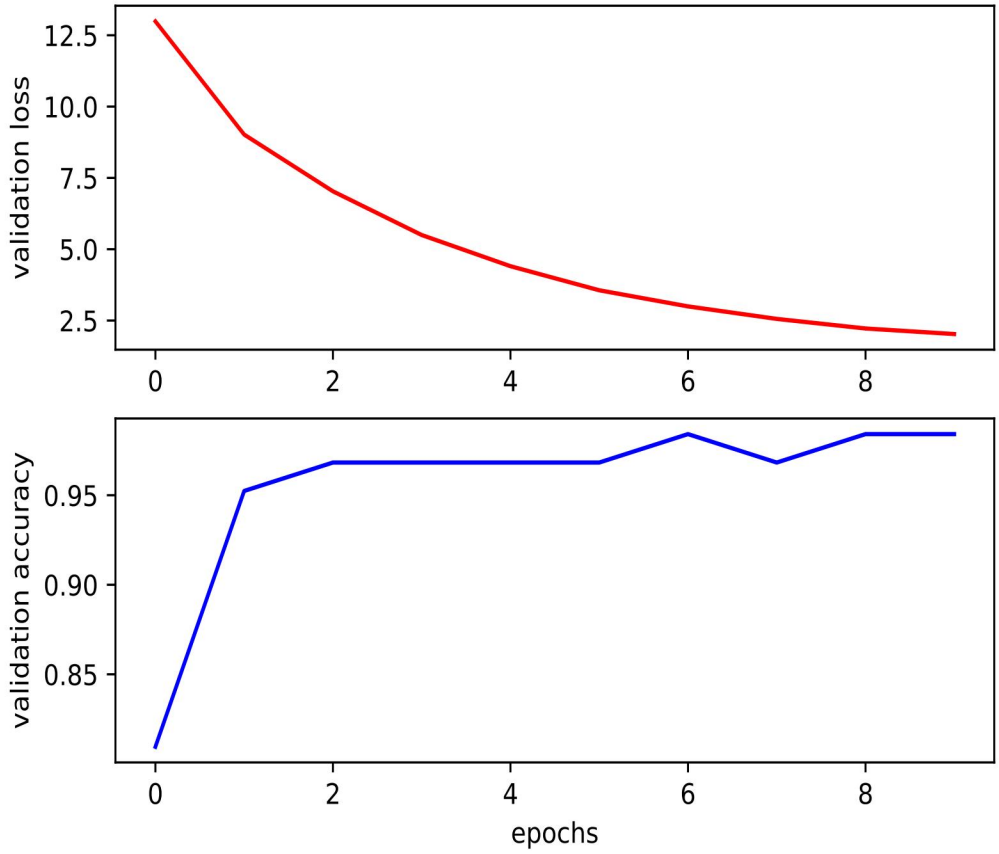

Figure 5: Performance of the hybrid classical-quantum neural network

Alternatively, another kind of ansatz with different qubit entanglement schema can be used:

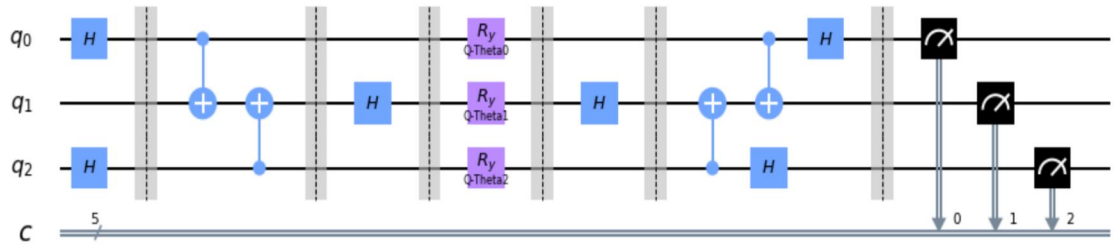

Figure 6: Alternative ansatz with three entangled qubits

Experiment with this ansatz will give the same performance and the same test accuracy: 0.96825. No post-encoding operations were used for simplicity reasons, and as that was not the objective of this project.

Benchmarking this algorithm with other similar[11] ones shows a slight difference:

algorithm	accuracy	time
quantum-classical numpy simple 2-layers NN	0.96825	2 min
quantum-classical pytorch 2-layers CNN	1	1 min
classical numpy simple 2-layers NN	0.97	7 sec

Implementation in pytorch of 2-layers CNN with quantum classifier has superior performance in terms of accuracy, but this was expected, as CNN has much higher representation capacity for the image recognition. Hybrid implementation in numpy has comparable performance to the classical neural network in terms of accuracy, and the convergence is slower and smoother, which we can notice by comparing Figure 5 with Figure 7.

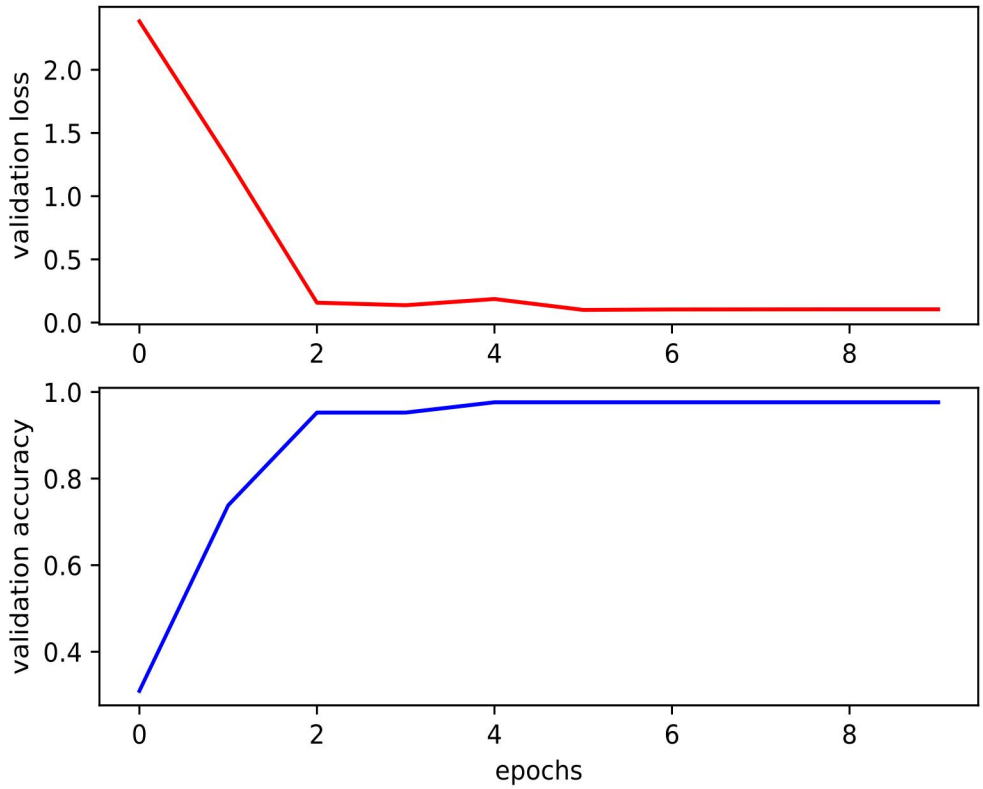

Figure 7: Performance of the classical neural network

[11] Implementation of MNIST with PyTorch and Qiskit:
https://github.com/liangqiyao990210/Quantum-Deep-Learning

In terms of the calculation time, it is clear that quantum circuit that will be run for 1'000 shots for the forward pass, and in addition for 3 x 1'000 shots for the calculation of $Q(\Theta + s)$, and 3 x 1'000 shots for the calculation of $Q(\Theta - s)$, and this for each datapoint increases the computational burden.

4.3 What are the conclusions?

It follows from this project that amplitudes of the quantum state (or in other words rotational angles of entangled qubits) can be used as a technique to encode the data. We also see that the results of quantum circuits are probability distributions that can be controlled, and this is suitable for the use in machine learning context. We also conclude that quantum circuits can be used as classifiers on top of the classical neural network, and even though this technique is of a trivial practical value, it still demonstrates how to build and train variational quantum algorithms, where one part is executed on a quantum machine and one part on a classical computer. We have also seen that there are different ways how quantum ansatz can be implemented and some can behave identically in case of small entanglement schemas with several qubits and simple or non-existent subsequent operations.

The right question here is: in which cases and in what architectures can quantum circuits be of a practical use in the context of deep learning?

More work is needed to provide a precise answer, but the intuition says that developing a hybrid ansatz by modeling complex non-linear functions with very large state spaces to be used inside a neural network can provide the most benefits in terms of increasing the representational power and/or provide computational speedup. As a new generation of quantum computers with more than 1'000 qubits is expected to arrive in 2023, the design of such complex models will become a very important research topic.

As about the increased computational burden and calculation time, it is expected that for complicated and large networks, the execution time of the quantum circuits will have less impact. A single-qubit gate on a superconducting quantum computer nowadays takes typically 2-4 ns, and the controlled two-qubit gates take 20-30 ns. This, for the shallow circuits of the depth up to 20 gates per qubit, and multiplied with 1'000 shots will represent around 300-500 μs. In the given example, if we have to perform 1 forward + 6 backward passes, this would represent 3.5 ms * dataset size / qubits. But certainly this impact would have to be carefully evaluated and modeled for each particular engineering case.

5 Closing comments

This project has achieved:

- Comprehensive overview of the quantum machine learning algorithms and indication of their underlying principles
- Description of the quantum computing principles used in the implementation of hybrid models and variational quantum algorithms
- Implementation of the quantum classifier in numpy, both as a standalone quantum classifier, and as part of the hybrid quantum-classical network; analysis of the performance, benchmarking with other similar non-quantum algorithms and discussion of the results

Challenges:

- End-to-end training requires careful tuning of the architecture and hyperparameters and control of the properties of the training dataset. Overall cost function has a non-convex and very irregular shape, and quantum circuit is very sensitive to the input values, which means that having very small values close to zero as the output of sigmoid activation function exploded the gradients of the quantum layer, and the training was unsuccessful. It took long time to find out the reason of this problem, to adapt the data to this case, and to determine the right place of the quantum layer in the overall architecture.
- Tanh activation function has better properties if used immediately before the quantum layer, as it doesn't squeeze the probabilities close to zero.

Potential next steps to continue experimentation:

- Scaling up the size of the hybrid network to distinguish among more than two classes and work in mini-batches instead of SGD.
- Implement quantum layers as hidden layers, not just use as a classifier.
- Exploring alternative ansatze and gradient calculation methods.
- Experimentation with more complex circuits where quantum state is evolved with additional operators, so to represent complex non-linear functions.
- Testing impact of the noise when running on real quantum hardware.
- Implementation of the hybrid neural network entirely in a higher-level programming framework.

END

This report was created by Sasha Lazarevic