UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Andrew Lei

Quantum Computing Techniques for Machine Learning

Master's Thesis (30 ECTS)

Supervisor: Dr. Dirk Oliver Theis

Tartu 2020

Quantum Computing Techniques for Machine Learning

Abstract: Quantum computing has been shown to provide a considerable speed boost for a number of problems. One area that some have looked into is using quantum computing for machine learning. I implement several methods for encoding data on the Iris dataset and compare their performance. Variational input encoding as suggested by Theis and Vidal seem to provide an advantage for Havlíček encoding. There were no similar improvements for amplitude encoding, but this could be because of the implementation.

Keywords: Quantum computing, machine learning, quantum machine learning

CERCS: P170 Computer science, numerical analysis, systems, control

Kvantarvutusmeetodid masinõppimiseks

Lühikokkuvõte: On tõestatud, et kvantarvutused annavad paljudele probleemidele märkimisväärset kiiruse suurenemist. Üks valdkond, mida mõned on uurinud, on kvantarvutuse kasutamine masinõppes. Rakendan Võhumõõga andmestikus andmete kodeerimiseks mitmeid meetodeid ja võrdlen nende toimivust. Tundub, et Theise ja Vidali soovitatud variatsiooniline sisendkodeering annab Havlíčeki kodeeringule eelise. Amplituudide kodeerimise osas sarnaseid parandusi ei olnud, kuid selle põhjuseks võib olla rakendamine.

Võtmesõnad: Kvantarvutamine, masinõpe, kvantmasinõpe

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Acknowledgements

I would like to thank my advisor, Dr. Dirk Oliver Theis for his excellent guidance throughout working on this thesis.

Additionally, I would like to thank Vojtěch Havlíček and Paul Kristan Temme for kindly providing me their data and the code necessary to reproduce their paper.

Contents

1 Introduction

Quantum algorithms have presented speedups to a number of problems, including factorizing integers in sub-exponential time and searching a database in sub-linear time. As a result, an area of research has opened around the topic of quantum machine learning, which could potentially provide similar speedups in training machine learning models.

1.1 Quantum Computing

Quantum computing has received a great deal of attention lately, especially with Google announcing quantum supremacy in 2019 on a 53-qubit computer, so many laypeople have a somewhat vague notion of how quantum computing works. This conception is that quantum computers somehow perform computations on possible inputs simultaneously in parallel. This is not entirely incorrect, but it does lack some of the nuance in how the process works.

The basic unit of computation on a quantum computer is a qubit, which can have a value of 0 or 1, as on a classical computer, but also some combination of the two. This is said to be a superposition. Qubits can be transformed with a series of operations, and this series of operations is referred to as the circuit. After all of the operations have been applied, a measurement is performed. Upon measurement, the qubits will collapse into a definite state rather than a superposition, with some probability for each of the states in the superposition.

While one could view applying operations to qubits in superposition to be performing computation on multiple inputs in parallel, the problem with this viewpoint is that upon measurement, only one result would be available. The process of computation and measurement can be repeated to estimate the probability of each result, but if that were all there is to it, it would not be enough to provide an advantage over classical methods. Why not simply use random samples each time a superposition is created?

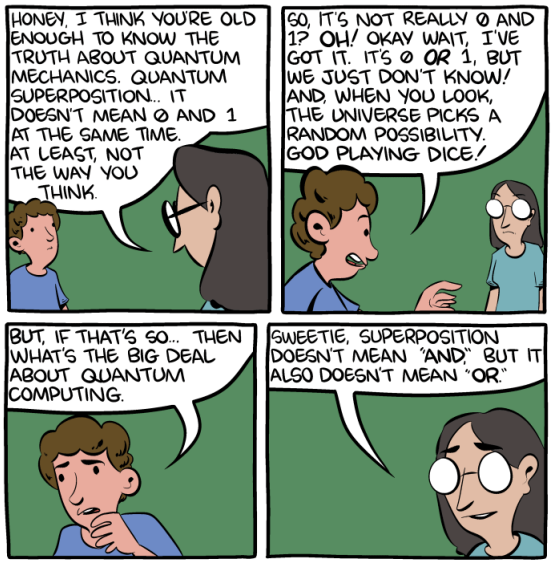

Figure 1. Quantum computing is not parallel [AW16]

The key here is that considering computation on states in superposition to merely be computation on different possibilities in parallel elides over an essential detail, which is that the different possibilities affect each other. The mechanism by which this occurs is that a state in superposition is not just a combination of probabilities of measuring 0s and 1s, but rather it is a combination of probability amplitudes, which unlike probabilities can be negative or even complex. The probabilities are the magnitude squared of these amplitudes.

If you have some background in physics, you may recall the famous double-slit experiment. To summarize the experiment, a light is projected onto a screen through one or both of two slits. When only one slit is open, the result is what you might expect: a central peak on the screen corresponding to the location of the open slit. But when both slits are open, rather than just a combination of two peaks, instead what appears is an alternating pattern of light and dark bands.

The reason for this is that in certain spots, the probability amplitude at a location from one slit might be negative, while the amplitude at the same location from the other slit might be positive. So while the magnitude squared of each of these give a positive result both times, resulting in light appearing in the location when only one slit is open, when both slits are open, the positive contribution is cancelled out by the negative contribution, resulting in destructive interference. On the other hand, at other locations, the two contributions have the same sign, making it brighter.

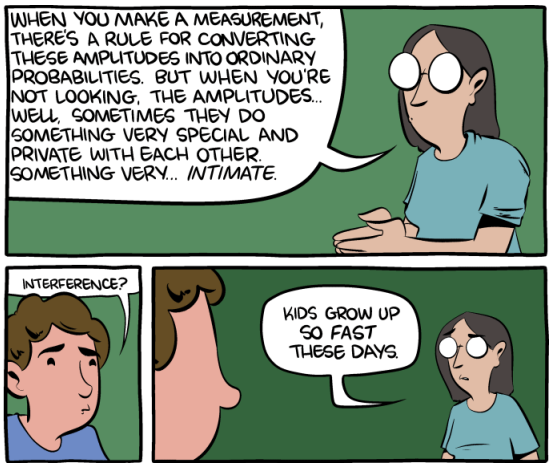

Figure 2. Probability amplitudes interfere with each other [AW16]

To see how this affects computation, consider the Hadamard gate, H, which operates on a single qubit. For $|0\rangle$ it gives $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$. This state, upon measurement, will give a measurement of $|0\rangle$ 50% of the time and $|1\rangle$ the other 50% of the time. But for $|1\rangle$, H gives $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$. If you square the amplitudes here, you will find that, again, the probability of measuring $|0\rangle$ will be 50% and likewise for $|1\rangle$.

So what is the difference? One interesting thing to note is that H is its own inverse, i.e., applying H twice is just the identity. To see this, let us work out the calculation for $HH|0\rangle$:

$$\begin{aligned} HH|0\rangle &= H\left(\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right) \\ &= \frac{H|0\rangle}{\sqrt{2}} + \frac{H|1\rangle}{\sqrt{2}} \\ &= \frac{|0\rangle+|1\rangle}{2} + \frac{|0\rangle-|1\rangle}{2} \\ &= |0\rangle \end{aligned}$$

The calculation for $HH|1\rangle$ is similar and left as an exercise to the reader. Note here how the positive $|1\rangle$ produced by one part of the superposition state is cancelled out by the negative $|1\rangle$ in the other part. The $|1\rangle$s destructively interfere, while the $|0\rangle$s constructively interfere.

Quantum algorithms make ample use of this property in their design, by having destructive interference in the incorrect answers and constructive interference in the correct ones. In ideal cases, this means a correct answer can be obtained with a single measurement.

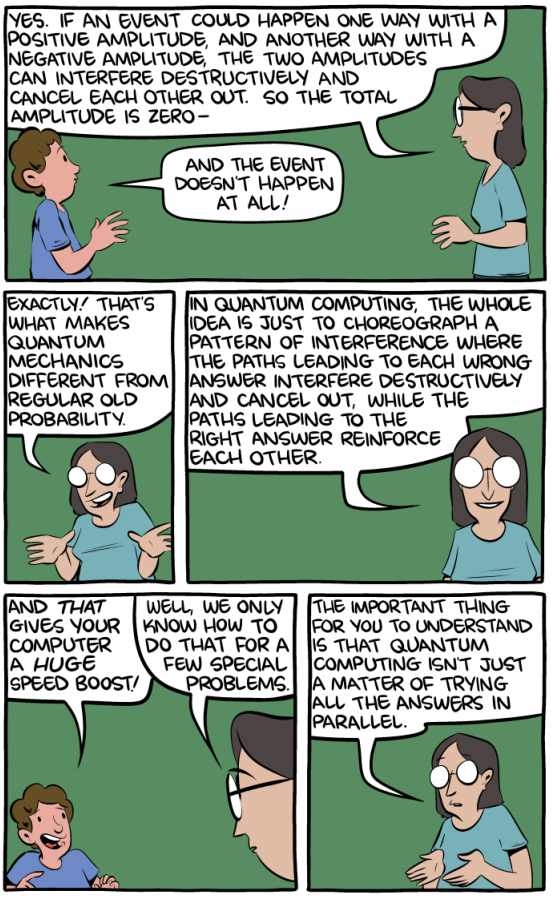

Figure 3. Quantum algorithms exploit interference to solve certain problems [AW16]

Since algorithms for which this is possible are only known for some problems, a speedup may not be possible for all problems. In particular, the time complexity class BQP, or bounded-error quantum polynomial time, is the class of quantum algorithms that runs in polynomial time with error probability of at most $\frac{1}{3}$. Just like it may be that P $\neq$ NP, it may be that BQP $\neq$ NP, and like the former, BQP $\neq$ NP is suspected.

1.1.1 Entanglement

The set of possible states for two qubits include $|00\rangle$ (i.e., both qubits are in the zero state), $|01\rangle$, $|10\rangle$, $|11\rangle$, and linear combinations of these. If we consider the state $\frac{|00\rangle+|11\rangle}{\sqrt{2}}$, we have a 50% chance of measuring $|00\rangle$ and a 50% chance of measuring $|11\rangle$. What if we only measure one of the qubits? If we measure the first qubit to be $|0\rangle$, and we know the state was in $\frac{|00\rangle+|11\rangle}{\sqrt{2}}$ before measure-

ment, we know the second qubit must be $|0\rangle$. This is an example of quantum entanglement. The qubits are entangled with each other, so if one is measured, the other must collapse as well.

In general, qubits are entangled if they cannot be separated into a tensor product of the each qubit. For example, $\frac{|00\rangle+|11\rangle}{\sqrt{2}}$ cannot be written in the form $(a\,|0\rangle + b\,|1\rangle) \otimes (c\,|0\rangle + d\,|1\rangle)$. To see this:

$$\begin{aligned}(a\,|0\rangle + b\,|1\rangle) \otimes (c\,|0\rangle + d\,|1\rangle) &= ac\,|00\rangle + ad\,|01\rangle + bc\,|10\rangle + bd\,|11\rangle \\ &= \frac{|00\rangle + |11\rangle}{\sqrt{2}}\end{aligned}$$

For this to occur we must have $ad = bc = 0$. But if $a = 0$, then $ac = 0$ and the coefficient for $|00\rangle$ is 0, and if $d = 0$, then $bd = 0$, and the coefficient for $|11\rangle$ is 0. Hence, we can conclude that it cannot be written in the above form.

This is an issue for simulating quantum computers, as it is not possible in general to perform computations on subunits of the quantum state. E.g., it is not generally possible to work with two circuits with four qubits each and then combine them for an eight qubit circuit.

1.2 Machine Learning

Like quantum computing, machine learning has garnered a great deal of attention in recent years, with applications such as self-driving cars, machine translation, and image recognition receiving not just press, but also widespread consumer use.

The subject of machine learning is quite broad, and has a great deal of overlap with the field of statistics. As a result, it is difficult to precisely state what machine learning is about. A tentative description of the task that could be expanded upon is to create a model that provides "good" predictions.

The first question here is what a "good" prediction is. This itself is a topic of research in machine learning, as it is dependent on both the task at hand and the algorithm used. In general, how good a prediction is determined by what is called a loss function, which is a function that says how incorrect an answer is.

Let us consider several examples:

- In a model for recognizing handwritten digits, for a handwritten "3", the loss function is high when it gives a 20% chance of the number being "3", and low when it gives a 90% chance of the number being "3".
- In a model for playing chess, the loss function is high when the model gives a move that results in the computer being checkmated in the next move, and low if that possibility is more distant.
- Some machine translation models give predictions for the translation word-by-word. In such a model, the loss function is high when the next word is something completely different from the word in the model translation, and low when it is the same.

A loss function can be either continuous or discrete, but since many machine learning algorithms rely on differentiating the loss function in order to minimize it, continuous loss functions tend to be preferred in many situations.

With this in mind, we can describe the task of machine learning as creating a model which makes predictions that minimizes a loss function.

The other question here is what is meant by a "creating a model". If I write a program to sort a list, am I creating a model that minimizes a loss function for a sorted list? If we consider appropriate loss functions for the task, if the program is correct, it would certainly minimize it. But this does not seem to resemble what we might think of as machine learning. Nor would writing by hand a program with a thousands or millions of branch conditions depending on whether or not some set of pixels have certain values resemble machine learning.

The missing ingredient here is that machine learning involves learning. A model in machine learning has parameters, which mean values that can be adjusted. If I try to fit a cubic polynomial to a set of datapoints, such a function takes the form $ax^3 + bx^2 + cx + d$. a is a parameter because it can be changed, but the 3 in the exponent is not a parameter because it is fixed.

So, with this in mind, we can describe the task of machine learning in a somewhat satisfactory manner: Optimizing the parameters of a model in order to minimize a loss function.

1.2.1 Optimization

The "learning" part of machine learning occurs through the process of optimizing the loss function. Many machine learning algorithms, most notably neural networks, employ gradient-based methods for optimization.

Recall from calculus how for a function of one parameter, f, if its derivative at x is positive, i.e. $f'(x) > 0$, then f is increasing at x. In that case, lower values of x should give lower values of $f(x)$. And conversely, if $f'(x) < 0$, then higher values of x should give lower values of $f(x)$.

If we wished to create an algorithm for finding the lowest value for f, then we could iterate by subtracting the derivative. When the derivative is positive, subtracting gives a smaller number, and when it is negative, subtracting gives a larger one.

$$x_{n+1} = x_n - \gamma f'(x)$$

γ here is some parameter that can be adjusted to affect the size of the steps taken. If it is too large, the algorithm could "overshoot", and if it is too small, convergence may taken many steps.

This is essentially a one-dimensional version of gradient descent, which instead uses the gradient, a generalized version of the derivative. More sophisticated gradient-based methods exist, but this should give a glimpse as to their fundamental character.

So evaluating the gradient is important for many machine learning algorithms. One could evaluate the gradient by approximation like so:

$$f'(x) \approx \frac{f(x+\epsilon) - f(x)}{\epsilon}$$

But there are a number of problems with this. The gradient is then approximate, and in theory one can gain more accuracy by decreasing the size of ϵ, but in practice one begins to encounter problems with numerical precision.

One alternative is symbolic differentiation. This obtains the gradient by evaluating it via the rules of calculus from the function's symbolic representation. The advantage here is that it obtains the exact value of the derivative. However, symbolic differentation has some problems with speed, expressions for derivatives that end up significantly larger than the original function expression, the complexity of symbolic algebra, and applicability limited to closed form expressions which exclude things such as control flow statements.

The other alternative, which is the preferred solution for obtaining gradients in optimization problems, is to use automatic differentiation. Automatic differentiation evaluates the gradient alongside the evaluation of the function, as every step in the evaluation of the function corresponds to a step in the evaluation of the derivative. The result is a derivative that is evaluated numerically, but one which is still exact. The quantum machine learning library used in the experiments in this thesis, PennyLane, implements automatic differentiation for quantum circuits.

One problem with gradient-based optimization methods is that they only find a local minimum for the loss function. It is possible to reach a point where all the surrounding points have a greater loss than that point, but also for another point to exist elsewhere with lower loss. A possible remedy to this is to use several random initial parameters.

1.2.2 Overfitting

If a machine learning model is judged purely based on minimizing the loss function used in the learning process, a model that would accomplish that task, yet still be a very poor model, would be to simply use a database that contained all the corresponding correct labels. Anything not in the database, it could return some arbitrary value.

In a somewhat less extreme example, it is possible to perfectly fit a degree n polynomial to a set of $n+1$ points, assuming these points all have different x coordinates. So a degree 99 polynomial could fit 100 points perfectly. However, in between points, the polynomial oscillates wildly. It seems intuitive that the true pattern would not diverge so enormously in between datapoints.

This problem can be managed by dividing the dataset into a training set and a testing set. The optimization process can then minimize the model's loss over the training set, while the model can be evaluated on the testing set. Note that the split itself does not solve the problem of overfitting, but rather it provides an indication of when overfitting occurs.

1.3 Quantum Machine Learning

The growing power of quantum computers has introduced the prospect of using quantum circuits to improve parts of the machine learning process. One of the first questions one might ask here is where might such an improvement be made.

One possibility is to use a quantum algorithm for the optimization process. This approach has been taken by D-Wave, which has produced a computer designed for quantum annealing, an optimization method. It is currently not clear whether or not their hardware provides any advantage over classical computers. For example, [MKT17] demonstrated that an apparent quantum speedup was obviated by the existence of a classical algorithm with polynomial time complexity. Quantum algorithms for optimization are not the focus of this thesis, but they do appear promising, whether or not D-Wave itself can deliver any advantages.

The other possibility, which is the subject of this thesis, is to use a quantum circuit in the machine learning model. The general pattern for how these models work is that in order to classify a point, there are three steps: Encoding, inference, and measurement.

In the encoding step, the point is first encoded into a quantum circuit by using a series of quantum gates with parameters that depend on the datapoint's features. There are several methods to perform such an encoding, and they are the main topic of this thesis.

In the inference step, a series of parametrized gates come after the gates in the encoding step. These gates perform various operations and have parameters that can be adjusted so that the model can be learned.

Finally, in the measurement step, the circuit is measured, and the measurement is converted into a label. A common way of doing this for binary classification is to measure some observable quantity (i.e., something which gives a scalar value), classify it as one label if it meets some threshold, and otherwise classify it as the other label. In this step, the loss is also calculated.

The circuit used in the model can then be used to estimate the loss by performing repeated measurements, and using a gradient obtained from automatic differentiation, adjust the parameters in the inference step in order to improve the model.

2 Background

2.1 Hilbert Spaces

Hilbert spaces are a kind of vector space that generalize the concept of an inner product (or dot product) with an additional desirable property of completeness. They are important for understanding quantum computing as quantum states are unit vectors in Hilbert spaces. We will discuss these concepts as follows.

An inner product space is a vector space with an inner product, φ, which is a function of two vectors that obey three rules

1. Conjugate symmetry: $\varphi(x,y) = \varphi(y,x)^*$
2. Linearity in the second argument: $\varphi(y, ax_1 + bx_2) = a\varphi(y,x_1) + b\varphi(y,x_2)$. Note that in mathematics, the usual convention is linearity in the first argument, but the second argument is more suited to the convention in quantum mechanics.
3. Positive definite: $x \neq 0 \implies \varphi(x,x) > 0$ and $\varphi(0,0) = 0$.

If limited to real numbers, more familiar rules appear: $\vec{x} \cdot \vec{y} = \vec{y} \cdot \vec{x}$, $\vec{y} \cdot (a\vec{x}_1 + b\vec{x}_2) = a\vec{y} \cdot \vec{x}_1 + b\vec{y} \cdot \vec{x}_2$, and $\vec{x} \neq 0 \implies \vec{x} \cdot \vec{x} > 0$.

From the first two rules, we can see there is also conjugate linearity in the first argument, which is to say $\varphi(ay_1 + by_2, x) = a^*\varphi(y_1,x) + b^*\varphi(y_2,x)$. To see this:

$$\begin{aligned}
\varphi(ay_1 + by_2, x) &= \varphi(x, ay_1 + by_2)^* \\
&= (a\varphi(x,y_1))^* + (b\varphi(x,y_2))^* \\
&= a^*\varphi(x,y_1)^* + b^*\varphi(x,y_2)^* \\
&= a^*\varphi(y_1,x) + b^*\varphi(y_2,x)
\end{aligned}$$

The norm of a vector, x, represented as $\hat{\varphi}(x)$ is the square root of the inner product of the vector with itself, i.e., $\hat{\varphi}(x) = \sqrt{\varphi(x,x)}$. This is a generalization of the length of a vector.

With a vector length defined, we can define a distance between two vectors as the norm of their difference, i.e., the distance between x and y is defined as $\hat{\varphi}(x-y)$.

A Hilbert space is an inner product space that is complete, which is to say that for any convergent sequence of vectors, their limit is in the Hilbert space. Somewhat roughly, a sequence of vectors that are getting closer together could be viewed as converging to some limit.

More precisely, a Cauchy sequence of vectors is a sequence $x_1, x_2, x_3, ...$ such that for some arbitrarily small number, $\epsilon > 0$, we can always find some point in the sequence, N, beyond which any two points have a distance smaller than ϵ, or $\hat{\varphi}(x_n - x_m) < \epsilon$ for any $n, m > N$. This is a more precise way of saying that the distances between any two vectors in this sequence keeps shrinking and shrinking as you go further along.

A similar case occurs with the real numbers and the rational numbers, as not every sequence of rational numbers converges to a rational number (consider the sequence $1, 1.4, 1.41, 1.414, 1.4142, ...$ that converges to $\sqrt{2}$). The real numbers are complete, but the rational numbers are not. This property is helpful for performing calculus with these vectors, and likewise while calculus cannot be performed with rational numbers, it can with real ones.

Let us now introduce the notation used in quantum mechanics for Hilbert spaces. The notation $|x\rangle$ represents a vector, x, in the Hilbert space H and is called "ket x". The notation $\langle y|$ represents a linear functional with the type

$H \to \mathbb{C}$ and is called "bra y". Kets can be transformed into bras using the inner product:

$$\langle y| = \varphi(y, \cdot)$$

Then, the inner product between two vectors is written as $\langle y|x\rangle$. An alternative interpretation is to view kets as column vectors and bras as row vectors, where transforming one to the other is done via conjugate transpose. The inner product then follows through the normal rules of matrix multiplication. This interpretation only quite works for finite-dimensional Hilbert spaces and perhaps for countably infinite dimensions, but generally Hilbert spaces can have uncountably infinite dimensions.

In quantum mechanics, we demand that quantum states have unit length, i.e., if $|\psi\rangle$ is a quantum state, then $\langle\psi|\psi\rangle = 1$. The inner product between two states, $\langle\varphi|\psi\rangle$, has some significance as $\langle\psi|\varphi\rangle\langle\varphi|\psi\rangle$ can then be interpreted as the probability of measuring $|\varphi\rangle$ when the state is $|\psi\rangle$. With this in mind, the unit length constraint corresponds to the fact that we should find that something with state $|\psi\rangle$ has a 100% chance to be in state $|\psi\rangle$.

Do not confuse $|0\rangle$, a vector called 0, with the 0 vector, which is typically just written as 0. The former has some length, while the latter has norm zero. The naming overlap is unfortunate, but there are many cases where using $|0\rangle$ is appropriate, e.g., representing the bitstate "0" on a qubit.

2.1.1 Operators

Operators in a Hilbert space, H, are linear transformations from $H \to H$. This means for two operators, A and B, and a scalar α, $(\alpha A)|x\rangle = \alpha(A|x\rangle)$, $(A+B)|x\rangle = A|x\rangle + B|x\rangle$, and $(AB)|x\rangle = A(B|x\rangle) = AB|x\rangle$.

The effect of an operator on a bra, meanwhile, can be defined by how it affects a ket. I.e., $\langle y|A$ is defined as the functional such that for all $|x\rangle$, $(\langle y|A)|x\rangle$ evaluates as $\langle y|(A|x\rangle)$; hence we can dispose with the parentheses, and write it as $\langle y|A|x\rangle$. This is straightforward matrix multiplication in finite dimensions.

Specific operators of interest are unitary and Hermitian operators. These have to do with properties of their adjoints. In Hilbert spaces with finite dimensions, operators are just square matrices, and their adjoints are the conjugate transpose. In general, the adjoint of an operator, A, is an operator, $A^\dagger$, such that if for some arbitrary $|x\rangle$, $|y\rangle = A|x\rangle$, then $\langle y| = \langle x|A^\dagger$.

A unitary operator is an operator, U, such that $UU^\dagger = U^\dagger U = I$, the identity operator. This sort of operator is important because it is the only sort of operator that can physically exist outside of the measurement process (at least, in the Copenhagen interpretation, which may be an indication of the Copenhagen interpretation's inadequacy), and therefore the only sort of operator we can construct on a quantum circuit.

Of particular note is that a unitary operator preserves lengths. This is akin to orthogonal matrices in real vector spaces, which describe rotations and reflections. This is because the inner product of $U|\psi\rangle$ with itself is

$$\begin{aligned}\langle\psi|\, U^\dagger U\, |\psi\rangle &= \langle\psi|\, I\, |\psi\rangle \\ &= \langle\psi|\psi\rangle\end{aligned}$$

Of course, if a valid quantum state transforms into another valid quantum state, both have unit length and so the length must be preserved. This does not quite justify why all operations must be unitary, which has to do with physics, and so is out-of-scope for this discussion.

The other important type of operator is a Hermitian, or self-adjoint, operator. A Hermitian operator is an operator, A, such that $A = A^\dagger$. That is, the operator is its own adjoint. Hermitian operators in real, finite Hilbert spaces are symmetric matrices. Like symmetric matrices, they are important because of the spectral decomposition theorem, which ensures the diagonizability of the operator and so the existence of an eigendecomposition, relevant to quantum mechanics as writing a quantum state as a sum of eigenvectors (or eigenstates) is useful in many situations.

Another important feature of a Hermitian operator is that for a Hermitian operator A, $\langle\psi|\, A\, |\psi\rangle$ is real. Note first that for any operator, B, $\langle\varphi|\, B^\dagger\, |\psi\rangle = (\langle\psi|\, B\, |\varphi\rangle)^*$ because if $|\xi\rangle = B\, |\varphi\rangle$, then $\langle\xi| = \langle\varphi|\, B^\dagger$, and we have

$$\begin{aligned}\langle\varphi|\, A^\dagger\, |\psi\rangle &= \langle\xi|\psi\rangle \\ &= (\langle\psi|\xi\rangle)^* \\ &= (\langle\psi|\, A\, |\varphi\rangle)^*\end{aligned}$$

From the conjugate symmetricity of the inner product. Now, we return to our Hermitian operator, A, where we can see

$$\begin{aligned}\langle\psi|\, A\, |\psi\rangle &= \langle\psi|\, A^\dagger\, |\psi\rangle \\ &= (\langle\psi|\, A\, |\psi\rangle)^*\end{aligned}$$

Any complex number equal to its conjugate must be a real number, so $\langle\psi|\, A\, |\psi\rangle$ must be real-valued. The converse is true, although I will not prove it here; it can be found in [Hal51, Theorem 24.2]. This property is important as physical observables in quantum mechanics have real expectation values, and are thus Hermitian.

2.1.2 Tensor Products

Hilbert spaces can be extended by combining them with a tensor product. For two Hilbert spaces, H_1 and H_2, their tensor product is $H_1 \otimes H_2$. A vector, $|\psi\rangle$ in this space has a component in H_1 and H_2. Let those be $|\psi_1\rangle$ and $|\psi_2\rangle$, respectively. Then, $|\psi\rangle$ is represented as $|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle$.

The inner product between $|\psi_1\rangle \otimes |\psi_2\rangle$ and $|\varphi_1\rangle \otimes |\varphi_2\rangle$ is defined as

$$(\langle\varphi_1| \otimes \langle\varphi_2|)(|\psi_1\rangle \otimes |\psi_2\rangle) = \langle\varphi_1|\psi_1\rangle \langle\varphi_2|\psi_2\rangle$$

Tensor products are used to represent systems of multiple qubits (which each have their own Hilbert space).

2.2 Quantum Computing

While bits on classical computers are represented as some combination of 0s and 1s, on quantum computers these are instead the basis vectors for the Hilbert space of possible states for the qubits.

More precisely, for n qubits, a computational basis state will take the form

$$\bigotimes_{i=0}^{n-1} |a_i\rangle = |a_0\rangle \otimes |a_1\rangle \otimes ... \otimes |a_{n-1}\rangle$$

Where $a_i \in \{0, 1\}$ for all i.

The inner product between two eigenstates is

$$\begin{aligned}\left(\bigotimes_{i=0}^{n-1} \langle a_i|\right)\left(\bigotimes_{i=0}^{n-1} |b_i\rangle\right) &= \bigotimes_{i=0}^{n-1} \langle a_i|b_i\rangle \\ &= \prod_{i=0}^{n-1} \delta_{a_i,b_i}\end{aligned}$$

Where δ is the Kronecker delta function.

The possible states of a quantum circuit with n qubits will be linear combinations of these eigenstates with unity norm.

2.2.1 Vector Notation

For a single qubit, it has the basis states $|0\rangle$ and $|1\rangle$. It can therefore be represented as a two-dimensional vector, $\begin{pmatrix} a_0 \\ a_1 \end{pmatrix}$ where $a_0^* a_0 + a_1^* a_1 = 1$.

In general, we can extend this, such that for an n-qubit state, we can represent it as a vector with dimension 2^n. The kth coefficient of this vector will be the coefficient of the basis state with the binary expansion of k.

Operators on n qubits can then be written as $2^n \times 2^n$ matrices.

This is part of why it quickly becomes impossible for classical computers to simulate quantum computers; assuming one bit per coefficient, a 128-qubit state vector would require 3.4×10^{26} terabytes of memory. For a 1024-qubit state vector, 1.8×10^{296} terabytes.

2.2.2 Bloch Sphere

By convention, the basis used for qubits is the Z basis, which corresponds to a measurement of spin angular momentum along the Z direction. However, one could just as well measure along the X and Y directions. Unlike classical mechanics, such a measurement will always find components along or against these directions.

The relationship the other bases can be described using the Bloch sphere.

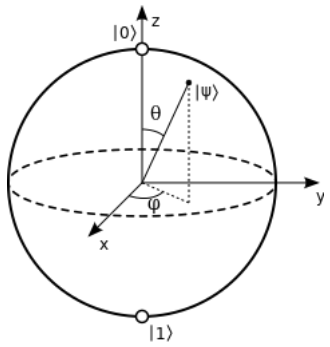

Figure 4. Bloch sphere

A state is represented as

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle \tag{2.2.2.1}$$

Where $0 \leq \theta \leq \pi$ and $0 \leq \phi < 2\pi$. Using the conventions $|+\rangle$ and $|-\rangle$ referring to the positive and negative directions of x, and $|\uparrow\rangle$ and $|\downarrow\rangle$ as the positive and negative directions of y, we find

$$\begin{aligned}
|+\rangle &= \frac{|0\rangle + |1\rangle}{\sqrt{2}} \\
|-\rangle &= \frac{|0\rangle - |1\rangle}{\sqrt{2}} \\
|\uparrow\rangle &= \frac{|0\rangle + i\,|1\rangle}{\sqrt{2}} \\
|\downarrow\rangle &= \frac{|0\rangle - i\,|1\rangle}{\sqrt{2}}
\end{aligned}$$

As well as, of course, $|0\rangle = |0\rangle$ and $|1\rangle = |1\rangle$.

Note that this definition of a state does not allow the coefficient of $|0\rangle$ to be non-real or negative. This is because the global phase of a quantum state does not matter, and it can be "factored out" so to speak by multiplying the entire state by some constant phase, such that the coefficient for $|0\rangle$ is either positive or zero.

2.2.3 Measurement

At the end of a quantum circuit, a measurement is performed. This measurement can take several forms. For example, the simplest sort of measurement would be measurement in the computational basis (by convention, the Z basis). The probability of an observation in a particular basis state is the inner product of that basis state with the quantum state under consideration times its complex conjugate.

That is, if we consider a state $|\psi\rangle$, the probability of observing it in a basis state, $|i\rangle$, is

$$\langle\psi|i\rangle \langle i|\psi\rangle = |\langle i|\psi\rangle|^2$$

The Z basis is not the only possible observable; physically, any Hermitian operator is valid (although practically, this is more limited). In general, for a family of Hermitian P_i operators such that $P_i = P_i^2$ and for all $|\psi\rangle$, $|\psi\rangle = \sum_i P_i |\psi\rangle$, the probability of measuring the observable P_i will be

$$\langle\psi| P_i |\psi\rangle$$

Note that a global phase difference will not affect the results of measurement. What this means, is that if we multiply $|\psi\rangle$ by a constant of unit magnitude, i.e., a constant of form $e^{i\phi}$, the measurement probabilities remain the same, as the probability will be

$$\begin{aligned}\langle\psi| e^{-i\phi} P_i e^{i\phi} |\psi\rangle &= e^{-i\phi} e^{i\phi} \langle\psi| P_i |\psi\rangle \\ &= \langle\psi| P_i |\psi\rangle\end{aligned}$$

Since we only interact with quantum systems via measurement, states that differ by only a global phase are effectively equivalent.

2.2.4 Operators

While in theory, any unitary operator, operators U such that $U^\dagger U = I$, are physically valid operations which can be performed on a quantum state, in practice we are limited to a set of possible operators for actual quantum circuits. Fortunately, there are two useful theorems. First, any unitary operator is equal (up to global phase) to some combination of CNOT and single-qubit gates [KLM10, Theorem 4.3.3]. Second, any single-qubit gate is equal (up to global phase) to some combination of at most three rotation gates [KLM10, Theorem 4.3.5].

Here are the relevant gates to this thesis:

2.2.4.1 Hadamard

This is the gate

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

This gate is useful as it has the effect of placing qubits initialized as $|0\rangle$ into superposition, specifically the state $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$, also known as $|+\rangle$.

2.2.4.2 Pauli Gates

These are the gates

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$
$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$
$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Note that X has the effect of a NOT gate.

$$X|0\rangle = |1\rangle$$
$$X|1\rangle = |0\rangle$$

So it is sometimes called the NOT gate instead. These gates are represented by their respective letters, but the X gate can sometimes appear as a circle with a cross, making more clear that it is a NOT gate.

Figure 5. NOT gate

Note that the X, Y, and Z Pauli basis states are eigenvectors of their respective Pauli gates.

2.2.4.3 Rotation Gates

There are three rotation gates corresponding to the Pauli gates. They can be obtained from taking the exponential of them. For Z this is simple:

$$
\begin{aligned}
R_Z(\theta) &= e^{-iZ\frac{\theta}{2}} \\
&= \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix}
\end{aligned}
$$

For the others, they can be found by eigendecomposition of their gates, resulting in

$$
\begin{aligned}
R_X(\theta) &= \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -i\sin\left(\frac{\theta}{2}\right) \\ -i\sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix} \\
R_Y(\theta) &= \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix}
\end{aligned}
$$

Since the X, Y, and Z Pauli basis states were eigenvectors of their respective Pauli gates, they are eigenvectors of their respective rotation gates as well. We can see that the Z rotation gate rotates the X and Y bases. A 90° rotation is

$$
\begin{aligned}
R_Z\left(\frac{\pi}{2}\right) &= \begin{pmatrix} e^{-i\frac{\pi}{4}} & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix} \\
&= \frac{1}{\sqrt{2}} \begin{pmatrix} 1-i & 0 \\ 0 & 1+i \end{pmatrix}
\end{aligned}
$$

Then,

$$
\begin{aligned}
R_Z\left(\frac{\pi}{2}\right)|+\rangle &= \frac{(1-i)|0\rangle + (1+i)|1\rangle}{\sqrt{2}} \\
&= \frac{(1-i)}{\sqrt{2}}\left(|0\rangle + \frac{1+i}{1-i}|1\rangle\right) \\
&= \frac{(1-i)}{\sqrt{2}}\left(|0\rangle + \frac{2i}{2}|1\rangle\right) \\
&= \frac{(1-i)}{\sqrt{2}}\left(|0\rangle + i|1\rangle\right) \\
&= \frac{(1-i)}{\sqrt{2}}|\uparrow\rangle
\end{aligned}
$$

Which is a 90° rotation from $|+\rangle$ up to a global phase difference.

2.2.4.4 Controlled Gates

These are gates that only operate if a certain qubit (called the control) is set to either zero or one. The gate that performs the operation performs it on a qubit called the target. The circuit notation for this is a line from the gate on the

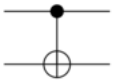

Figure 6. CNOT gate

target qubit to the controlled qubit, with a filled circle on the controlled qubit. For example, here is a controlled NOT (CNOT) gate:

This is a NOT gate that only performs the action of NOT if the controlled qubit is set to one. The mathematical operation of this is

$$|0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes X = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

In general, a controlled gate, U, can be written in the same form, $|0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes U$ (switching the 0s and the 1s if controlling on 0 instead). It is also possible to control on multiple qubits in an analogous manner, although these operations would have to be constructed from simpler gates. I will discuss a special case, the uniformly controlled gate, in a subsequent section on amplitude encoding.

Controlled gates are useful for entangling qubits with each other.

2.2.4.5 Single- and Two-qubit Gates in Multi-Qubit Circuits

Generally, only single- and two-qubit gates are necessary for universality (specifically, CNOT gates and rotation gates). Their action on a multi-qubit circuit can be determined by taking their tensor product with the identity on qubits they do not operate on. So, e.g., an $R_X(\theta)$ on the third qubit on a five-qubit circuit can be written as

$$I \otimes I \otimes R_X(\theta) \otimes I \otimes I$$

2.3 Machine Learning

In the supervised learning problem, we have some feature space, X, some label space Y, and a number of points in $X \times Y$. We would like to find some function, $f : X \to Y$, such that a loss function, $L : Y \times Y \to \mathbb{R}^{\geq 0}$, is minimized. f is called the model. Models typically have parameters from some parameter space, W, in which case, f can be seen as a function $f : W \times X \to Y$, and the task becomes to find $\theta \in W$ such that L is minimized.

In practice, we split the points we have in $X \times Y$ into training and testing points in order to determine whether by training on the former, the model can generalize to the latter.

There are two algorithms in particular that bear discussing here, support vector machines and neural networks (or multi-layer perceptrons).

2.3.1 Support Vector Machines

The principle behind SVM is to find a hyperplane that separates one class from another. The goal is to find the maximum-margin hyperplane, that is, the hyperplane dividing the classes such that the distance from the closest point is as far as possible. The side of the hyperplane a point, $\vec{x}_i$ is on can be found from the parameters of the hyperplane, $\vec{w}$ and b, by evaluating $\vec{w} \cdot \vec{x}_i - b$; if it is positive, it is on one side, if negative, the other.

This is called the hard-margin SVM, which has the shortcoming that sometimes points of the two classes are mixed, so no hyperplane can divide them. A soft-margin SVM instead minimizes a hinge loss function, $\max\left(0, 1 - y_i\left(\vec{w} \cdot \vec{x}_i - b\right)\right)$, where y_i is the label of point i. This function is zero when points appear on the correct side of the hyperplane and grows as points on the wrong side are farther away from the hyperplane.

One of the shortcomings of SVM is that this is linear, which presents a difficulty in non-linear problems. E.g., if we sample points in $[-1, 1]^2$ and label them based on whether they are within the unit circle, SVM cannot find any 2-hyperplane (line) that divides the classes. The solution to this is to replace the dot product with a nonlinear kernel function.

For example, the polynomial kernel $k\left(\vec{x}_i, \vec{x}_j\right) = \left(\vec{x}_i \cdot \vec{x}_j\right)^d$ can be used. For $d = 2$, if we use $\vec{w} = (1, 1)$, then we find that for points within the unit circle, their kernel with $\vec{w}$ is less than one, and for points outside, it is greater than one. This is then clearly linearly separable.

2.3.2 Neural Networks

The most basic kind of neural network is the multi-layer perceptron which is built from simple linear classifiers called perceptrons. These are quite similar to SVM:

$$f\left(\vec{x}\right) = \begin{cases} 1 & \vec{w} \cdot \vec{x} + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

However, these perceptrons had some issues, so modern neural networks, instead of performing this piecewise function, instead use a continuous activation function:

$$f\left(\vec{x}\right) = a\left(\vec{w} \cdot \vec{x} + b\right)$$

In particular, a popular activation function is the sigmoid activation function, $\operatorname{sig}(x) = \frac{1}{1+e^{-x}}$. Note how $\lim_{x \to \infty} \operatorname{sig}(x) = 1$ and $\lim_{x \to -\infty} \operatorname{sig}(x) = 0$.

While a single perceptron is a linear classifier, if multiple perceptrons are used, and their output is fed into more perceptrons, and so on, the result can perform non-linear classification. Rather than a single vector, $\vec{w}$, we then have a matrix of weights, W, and our bias scalar, b, becomes a vector of biases, $\vec{b}$. If the dimension of $\vec{x}$ is n and we use the m perceptrons in a the first layer, the matrix W for that layer would have dimension $m \times n$, and the dimension of $\vec{b}$ is m.

The output of a first layer of perceptrons would appear as

$$\text{sig}\left(W_0\vec{x} + \vec{b}_0\right)$$

If we apply another layer of perceptrons, then we have

$$\text{sig}\left(W_1\text{sig}\left(W_0\vec{x} + \vec{b}_0\right) + \vec{b}_1\right)$$

And so on. Note that the dimensions of these weight matrices do not have to be the same, resulting in varying sizes for each layer.

According to the universal approximation theorem, from this single extra layer, any function can be approximated. However, in practice, more layers have been found to produce better results.

There are more complicated kinds of neural networks, but they follow a similar principle.

2.3.3 Training Models

We return to the task of minimizing $L : Y \times Y \to \mathbb{R}^{\geq 0}$ given a model and training set. The model, with parameters in space W, is a function from $f : W \times X \to Y$. So, the total loss for some set of weights is

$$l\left(\vec{\theta}\right) = \sum_{(x_i, y_i)} L\left(f\left(\vec{\theta}, \vec{x}_i\right), y_i\right)$$

This total loss is a function $l : W \to \mathbb{R}^{\geq 0}$. This problem is then solvable with the techniques of calculus by finding the gradient of l and applying some gradient descent optimization method.

Modern machine learning software include automatic differentiation, so that the gradients are automatically evaluated when computations are performed, so that such methods may easily be used.

2.4 Quantum Machine Learning

There are several steps in a quantum machine learning algorithm. First, the data must be encoded into the circuit. Second, parametrized gates operate on the encoded data, performing the inference part of the algorithm. Finally, the data is read and converted into a label.

2.4.1 Encoding Data

For each datapoint $\vec{x}$, the process of encoding data is to construct a series of gates that will transform the initial state, $\left|\vec{0}\right\rangle$, into some state that depends on $\vec{x}$, $|\Phi(\vec{x})\rangle$. For example, one such method is amplitude encoding, in which the resulting state would be $|\Phi(\vec{x})\rangle = \sum_i \sqrt{x_i}\,|i\rangle$; in this scheme, the data is encoded into the basis state amplitudes of the state.

According to [HCT+19] and [SP19, pg.193], classifiers based on quantum circuits should only provide a quantum advantage if $\langle\Phi(\vec{x})|\Phi(\vec{y})\rangle$ is difficult to estimate classically. [HCT+19] then provide their own encoding scheme, discussed in Section 4.1.

In [VT19], Vidal and Theis propose making the encoding process itself depend on learned parameters as well.

2.4.2 Inference

Inference in quantum machine learning amounts to the unitary operation, W, such that the loss, which uses the measurement result of an observable of $W|\Phi(\vec{x})\rangle$, is minimized. Mathematically, W is an $2^n \times 2^n$ matrix with complex elements which is unitary, where n is the number of qubits; it is parametrized with 2^{2n} free parameters.

However, in practice, such a general matrix cannot be directly learned, since a real quantum computer has to employ combinations of a limited set of gates rather than arbitrary unitary operations. So instead what is used are layers of rotation gates separated by entangling layers composed of controlled Pauli gates.

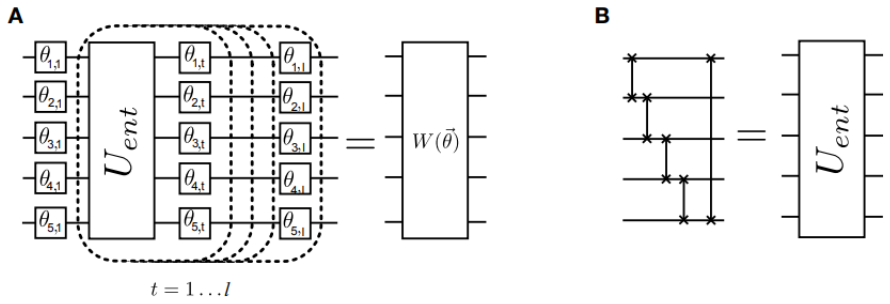

Figure 7. Circuit portion for inference from [HCT+19]. Havlíček, et al. use (a) Y and Z rotations on each qubit separated by (b) entangling layers that perform a circle of CZ gates (note that in the two-qubit case, the entangling layer does not perform a circle, and is just a CZ gate between the first and second qubits; two of them would cancel each other out). On the other hand, the PennyLane demonstration at [Xan] uses X, Y, and Z rotations separated by entangling layers of CNOT gates.

The free parameters are the rotation angles in the layers of rotation gates. The resulting state after application of the inference operator is then $W(\vec{\theta})\left|\Phi\left(\vec{x}\right)\right\rangle$, where $\vec{\theta}$ is the learned parameters.

Note that unlike neural networks, the parametrized layers are not separated by non-linearities; from the moment the data is encoded into the circuit until the circuit is measured, the entire process is linear. While the encoding process and measurement present non-linearities, this portion in between is entirely linear, and additional parametrized layers do not add any non-linearity. This could present a weakness, since in classical neural networks, even though one hidden layer is sufficient to model any function, multiple hidden layers perform better in practice.

2.4.3 Measurement

After all the operations in the circuit have been performed, some observable is measured. For example, [HCT+19] measure $Z_1 Z_2$ (in a two-qubit circuit), while the example used by Pennylane measures just Z_1 (again, in a two-qubit circuit).

Each measurement provides one of a number of discrete values, but we require the expected value for an observable, so measurements are made repeatedly in order to obtain the mean.

The end result is that the expected value of a measurement from the entire circuit is $\left\langle\Phi\left(\vec{x}\right)\right| W(\vec{\theta})^{\dagger}\mathbf{f}W(\vec{\theta})\left|\Phi\left(\vec{x}\right)\right\rangle$, where $\mathbf{f}$ represents the observable measured.

The measurement is then used to produce a label and a value for the loss which is used to optimize the parameters used in the inference step (and, in Vidal and Theis's case, in the encoding step as well).

3 Dataset

3.1 Havlíček Dataset

I reproduced the results by Havlíček, et al. with the dataset created in [HCT+19]. The methodology for generating this dataset will be described in Section 4.1.2.1.

3.2 Iris Dataset

The main dataset I used was the Iris dataset, a dataset consisting of three species of the Iris flower, with 50 samples for each species. There are four features (all in centimeters): Sepal length, sepal width, petal length, and petal width.

This provides two important differences from the Havlíček dataset. First, the generalization from two dimensions to four has to be made for Havlíček encoding. Second, classification for the Havlíček dataset used two classes, while a method for multi-class classification has to be used here.

3.2.1 Preprocessing

[HCT+19] drew datapoints from the range $(0, 2\pi]^2$, hence it was appropriate to normalize to this range for Havlíček encoding.

I used two approaches for normalizing for Havlíček encoding: Normalizing to the largest feature value in the dataset, and normalizing to the largest value for each feature.

For the experiments involving amplitude encoding, since amplitude encoding requires the sum of the squares of the features to add up to one, each datapoint was normalized so that this condition would be met.

The labels required some processing themselves. Since the original models were designed for binary classification, it was necessary to convert multi-class labels into labels suitable for binary classification, using one-hot encoding, or something similar, which would have multi-dimensional binary labels.

A naïve approach with two bias parameters would be to try something such as $(-1, -1)$ for label 1, $(-1, 1)$ for label 2, and $(1, -1)$ for label 3.

The problem with this approach is that for two biases, we require that $2\hat{y} > 1 - b_1$ for the first to be assigned a positive label, and $2\hat{y} > 1 - b_2$ for the second to be assigned a positive label. But either $b_1 \geq b_2$ or $b_2 \geq b_1$. Hence, either anything that is positive for the first label dimension is positive for the second as well, or vice versa.

To get around this, the encoding instead uses $(-1, -1)$ for label 1, $(-1, 1)$ for label 2, and $(1, 1)$ for label 3.

4 Experiments

For this thesis, I used PennyLane, a Python library for quantum machine learning that implements automatic differentiation [BIS+18].

4.1 Base Havlíček Model

4.1.1 Model

The base Havlíček model is an extension of the model used in [HCT+19]. Havlíček, et al. encode data as follows:

Where $\vec{x}$ is an n-element vector, with n being the same as the number of qubits.

The encoding consists of a layer of Hadamard gates on each qubit, followed by an operator, $U_{\Phi(\vec{x})}$, then another layer of Hadamards, and $U_{\Phi(\vec{x})}$ again.

```
def ital_U_gate(x, n):
    for i in range(n):
        qml.Hadamard(wires=i)

    U_gate(x, n)

    for i in range(n):
        qml.Hadamard(wires=i)
```

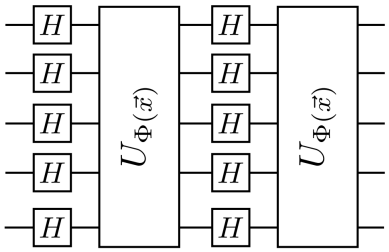

Figure 8. Encoding by [HCT+19]

```
    U_gate(x, n)
```

The $U_{\Phi(\vec{x})}$ gate consists of ϕ gates, which can operate on one or two qubits. The single-qubit ϕ operation is just a Z rotation by x_i on the ith qubit.

```
def U_gate(x, n):
    for i in range(n):
        qml.RZ(2 * x[i], wires=i)

    for i in range(n):
        for j in range(i):
            phi_gate2(x[j], x[i], j, i)
```

Each qubit gets a Z rotation with the corresponding feature of the datapoint being encoded. Then, each pairwise combination of qubits (and features) have the two-qubit ϕ gate applied to them. The j index ranges from 0 to i in order to not double-count.

```
def phi_gate2(x1, x2, w1, w2):
    qml.CNOT(wires=[w1,w2])
    phi = 2 * (np.pi - x1) * (np.pi - x2)
    qml.RZ(phi, wires=w2)
    qml.CNOT(wires=[w1,w2])
```

The pairwise ϕ gates, consist of a CNOT gate, a Z rotation by $2(\pi - x_i)(\pi - x_j)$ on the target of the CNOT gate, and another CNOT gate. Whether the ith gate is the target or the control does not matter; the effect of this series of gate either way.

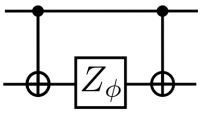

Figure 9. Pairwise component of Havlíček encoding, from [HCT+19]

After encoding, there are several layers, with each layer consisting of a Y and a Z rotation and an entangling layer.

```
def W_gate(theta_layers):
    l, n, _ = theta_layers.shape

    for layer in theta_layers[:-1]:
        for i, theta in enumerate(layer):
            qml.RY(theta[0], wires=i)
            qml.RZ(theta[1], wires=i)

        U_ent_gate(n)

    for i, theta in enumerate(theta_layers[-1]):
        qml.RY(theta[0], wires=i)
        qml.RZ(theta[1], wires=i)
```

The entangling layer consists of a circular pattern of CZ gates, except for the two-qubit case, where this would result in two CZ gates in a row, cancelling each other out.

```
def U_ent_gate(n):
    if n == 2: qml.CZ(wires=[0,1])
    else:
        for i in range(n):
            qml.CZ(wires=[i, (i+1)%n])
```

Finally, $\frac{1}{2}\left(1 + Z_1 Z_2 Z_3 Z_4\right)$ is measured, by adding the probabilities of an even parity state. The result from this is $\hat{y}$.

This is used in the loss function

$$\text{sig}\left(\frac{\sqrt{R}\left(\frac{1}{2} - \left(\hat{y} - \frac{yb}{2}\right)\right)}{\sqrt{2\left(1-\hat{y}\right)\hat{y}}}\right)$$

Where y is the label (± 1) and b is the bias parameter. R is set at 200.

For assignment, the point is assigned label $+1$ if $\hat{y} > 1 - \hat{y} - b$, and otherwise it is assigned -1.

4.1.2 Results

4.1.2.1 Havlíček Dataset

The dataset in [HCT+19] was generated using the Havlíček encoding discussed in this section in the following manner: Two-dimensional points were encoded into the circuit, and a random unitary operator, U was applied. They were then measured with the observable $Z_1 Z_2$. If the expected value of the measurement was greater than or equal to some threshold, Δ, then it was assigned a positive label. If it was less than or equal to $-\Delta$, it was assigned a negative label. In this case, [HCT+19] used $\Delta = 0.3$.

The paper does not say anything about measurements between $-\Delta$ and Δ, but presumably these are just not included for the dataset.

The task was then to learn the original random unitary operator used to produce the dataset, U.

Five trials were run for 100 iterations each, taking about six minutes each. Note one of the trials made basically no progress towards convergence, perhaps a set of poor initial conditions.

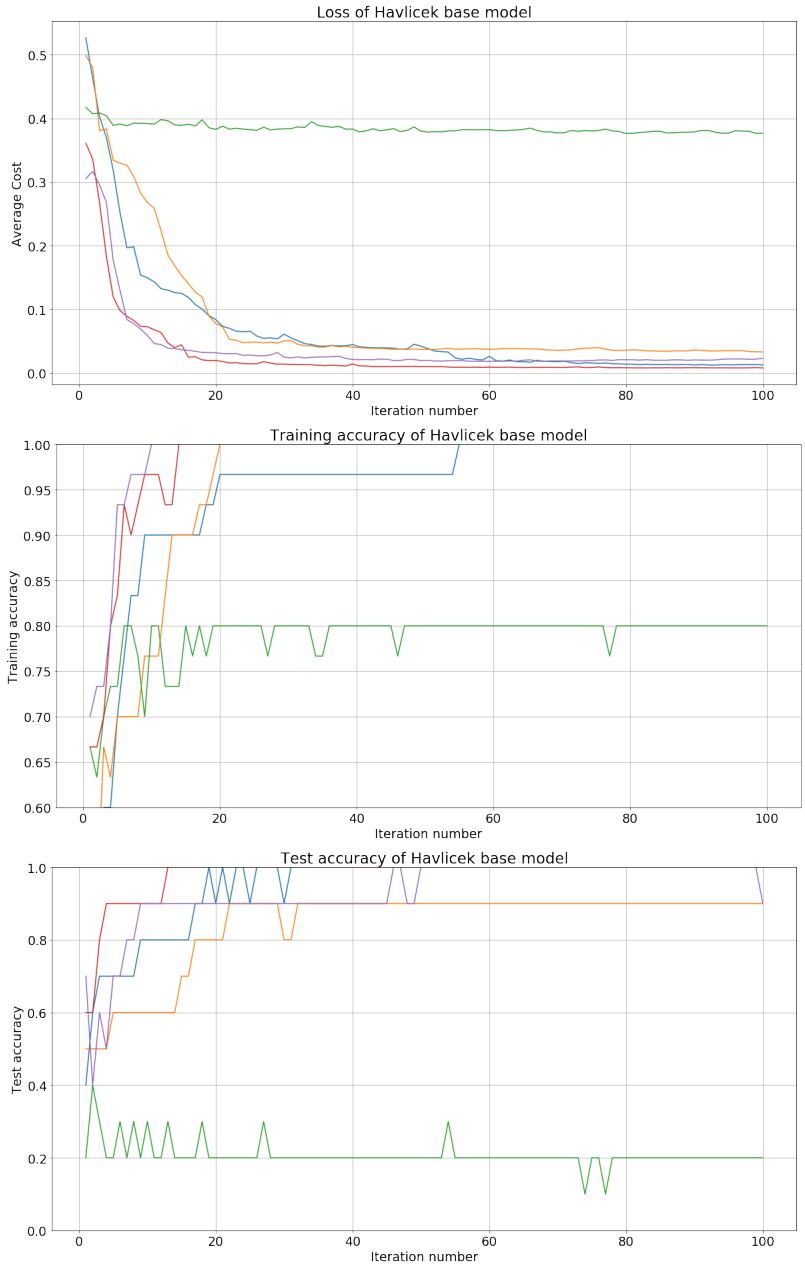

Figure 10. Metrics for Havlíček base model on Havlíček dataset

4.1.2.2 Iris Dataset

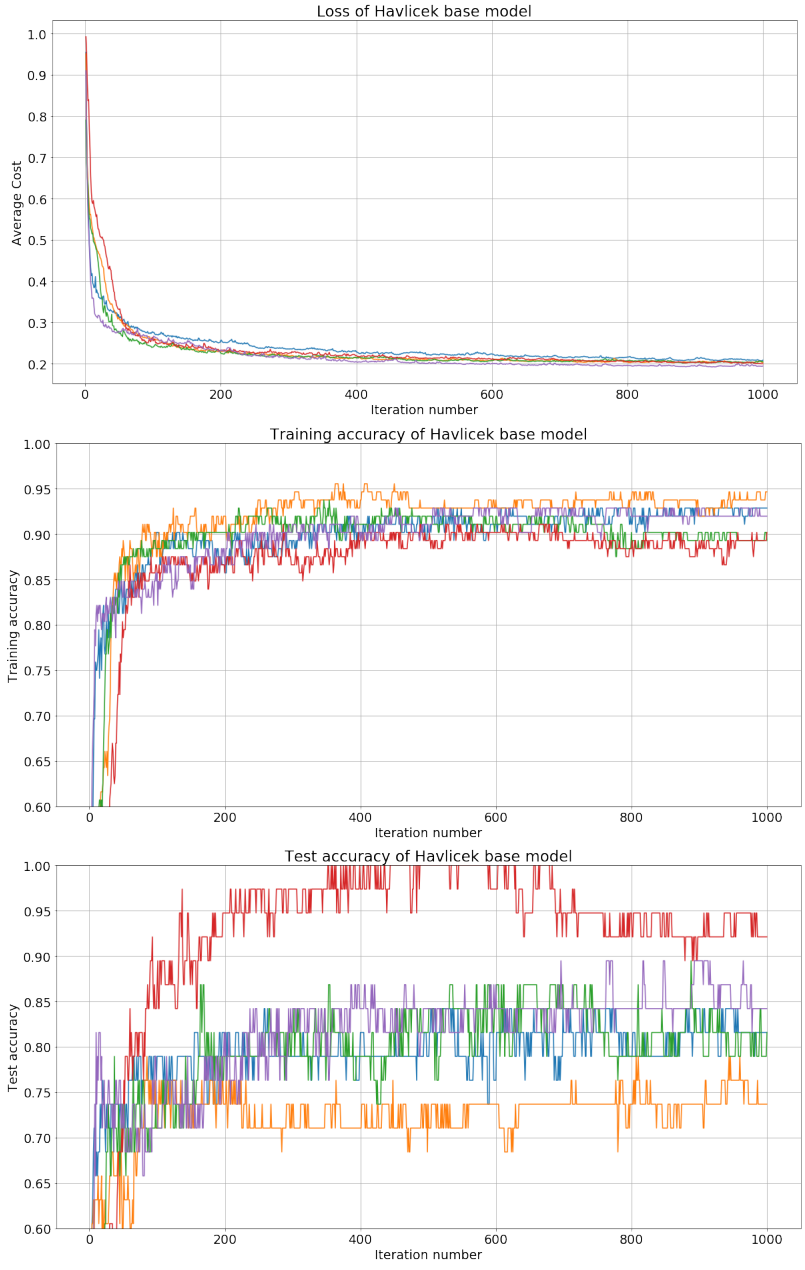

Figure 11. Metrics for Havlíček base model

Five trials were run for 1000 iterations each. This took about six hours each, with little variation in time. The results can be seen above. Average cost reached an average of 0.202 across trials, with training accuracy reaching 0.918 and test accuracy reaching 0.826.

It is apparent there is some overfitting, since the test accuracy is somewhat worse than the training accuracy.

4.2 Variational Encoding Havlíček Model

4.2.1 Differences From Base Model

The only difference in the variational Havlíček model is the use of variational encoding, by substituting $a_i x_i + b_i$ for each x_i in the features. a_i and b_i are then learned.

`U_gate` is replaced with `U_gate_var`:

```
def U_gate_var(x, n, alphas, betas):
    for i in range(n):
        qml.RZ(2 * alphas[i] * x[i] + betas[i], wires=i)

    for i in range(n):
        for j in range(i):
            phi_gate2_var(x[j], x[i], j, i, alphas[j], betas[j]
                          ,alphas[i], betas[i])
```

4.2.2 Results

Again, five trials were run on the Iris dataset for 1000 iterations each, for about six hours in total (no significant difference in time). Average cost reached an average of 0.116 across trials, with training accuracy reaching 0.961 and test accuracy reaching 0.926.

It is worth noting that for some reason, the optimizer did not noticeably adjust the initial values of variational encoding parameters in any of the trials. The variational encoding parameters may be difficult to learn. Despite this, even the worst trial's average loss was still below the best trial's average loss for the base model. However, this was ineffective for the Havlíček dataset.

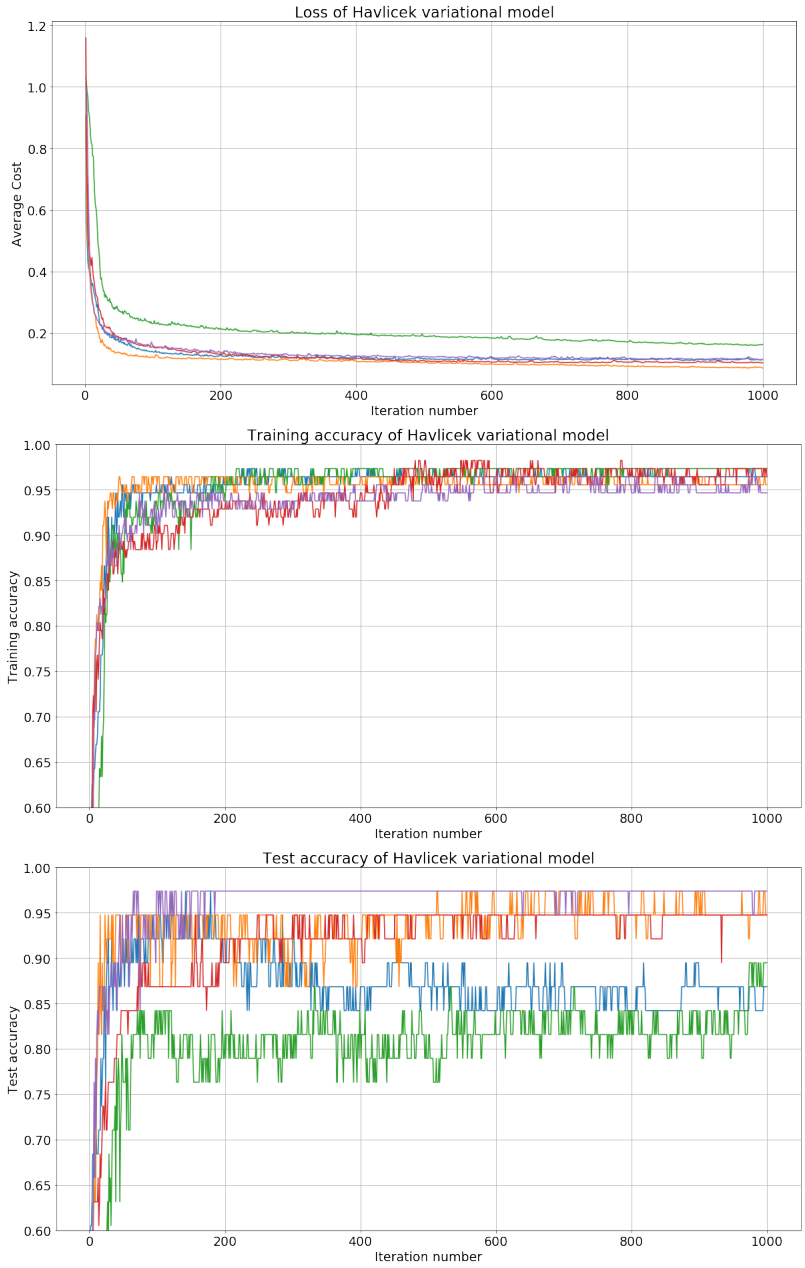

Figure 12. Metrics for Havlíček variational model

As expected, this model performed better than the base model. The lack of time difference from the base model likely indicates most of the time is spent in the simulator, and the extra variables for backpropagation do not add much

overhead (at least, at this stage). It is still perhaps slightly overfit, but the test accuracy is improved. We can compare the two models below:

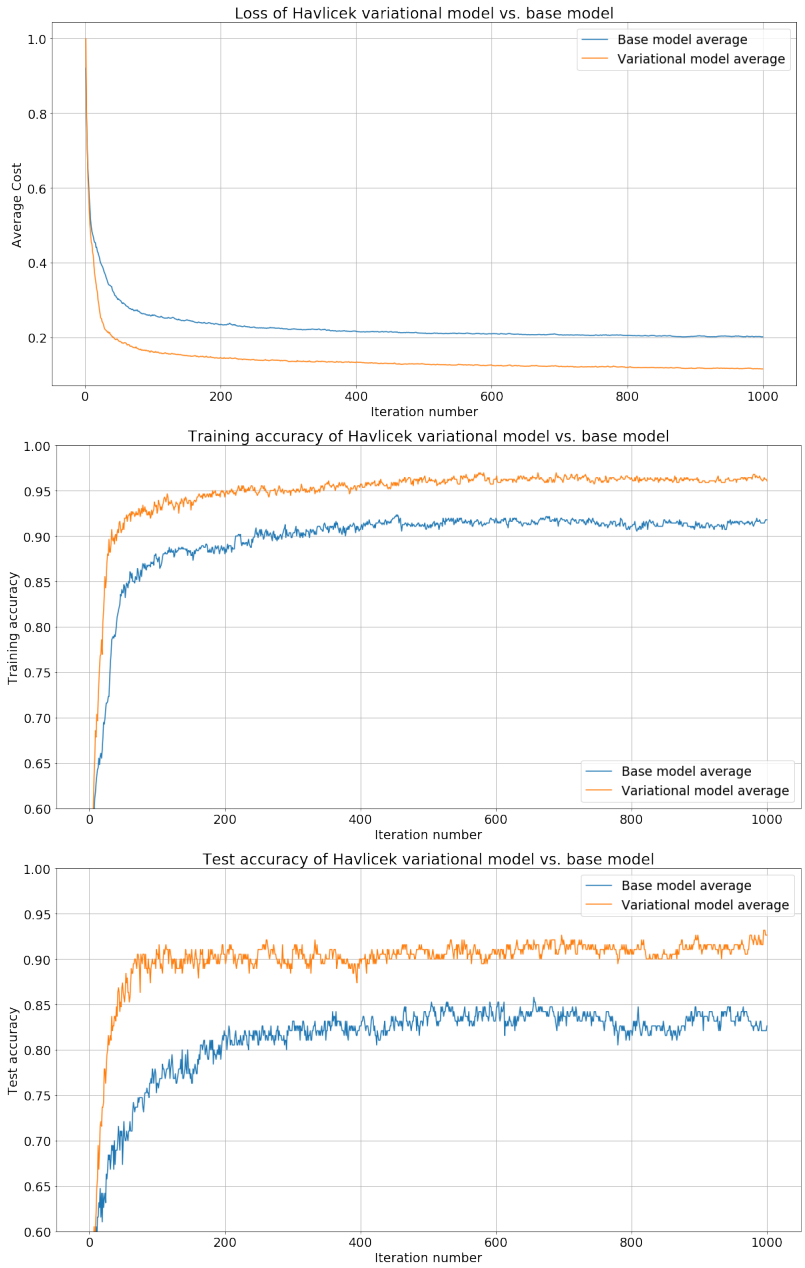

Figure 13. Comparison between Havlíček variational and base models

4.3 Amplitude Encoding Model

4.3.1 Model

Amplitude encoding encodes data into the probability amplitudes of the quantum state of the circuit. For a data point $(x_0, x_1, ..., x_{n-1})$, the data is normalized so that their sum of squares add up to one (since that is required for the probability amplitudes of a quantum state), $(x'_0, x'_1, ..., x'_{n-1})$. Then, this is encoded into the quantum state, which becomes $x_0 |0\rangle + x_1 |1\rangle + ... + x_{n-1} |n-1\rangle$.

The way in which this is done is described in [SP19, Chapter 5.2]:

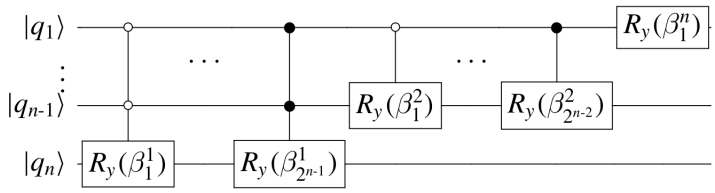

Figure 14. Quantum circuit implementing amplitude encoding described at [SP19, pg.152]

With the values of β_j^s as:

$$\beta_j^s = 2 \arcsin\left(\sqrt{\frac{\sum_{l=1}^{2^{s-1}} \left|x'_{(2j-1)2^{s-1}+l}\right|^2}{\sum_{l=1}^{2^s} \left|x'_{(j-1)2^s+l}\right|^2}}\right) \tag{4.3.1.1}$$

Where x'_i is the normalized ith component of the data point.

```
def get_beta(feats, j, s):
    sum1 = sum([feats[(2*j+1) * 2**s + l]**2 for l in range(2**s)]))
    sum2 = sum([feats[j*2**(s+1) + l]**2 for l in range(2**(s+1))])
    return 2 * np.arcsin(np.sqrt(sum1/sum2))

def get_betas(feats):
    n = max(1,int(np.ceil(np.log(len(feats))/np.log(2))))
    return [[get_beta(feats, j, s)
             for j in range(2**(n-s-1))]
            for s in range(n)]
```

The result is a list of lists for the multi-controlled rotation angles. There is one angle for the last list, as it is a single qubit rotation, two for the penultimate (two qubits, one of which is the control), four for the antepenultimate (three qubits, two of which are controls), and so on.

The black circles correspond to being controlled on 0 (gate only applied if 0) while the white circles correspond to being controlled on 1.

However, note that [SP19] probably intended for the above circuit to be implemented in reverse, as otherwise a circuit initialised from $|00...0\rangle$ would

only be affected by the 0-controlled gates. Indeed, it appears that applying the above circuit in the reverse order gives the desired result.

Note also that the multi-controlled gates are generally not something which is typically possible on real hardware, nor implemented in most simulators, at least not without the use of auxiliary qubits. Fortunately, however, [MVBS05] describes a method for transforming a series of multi-controlled rotations (namely, 2^k rotations controlled on k qubits, which is called a uniformly controlled rotation) into a series of one-qubit rotations and CNOT gates.

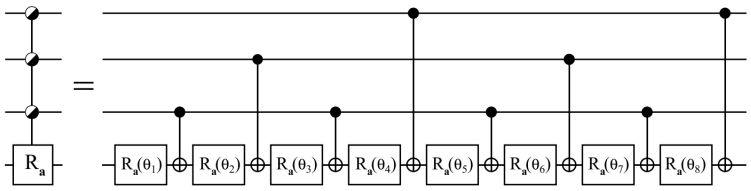

Figure 15. Uniformly controlled rotation decomposed into single-qubit rotations and CNOT gates from [MVBS05]

The angles here can be obtained from β_j^s from:

$$\vec{\theta} = M\vec{\beta^s} \tag{4.3.1.2}$$

Where

$$M_{ij} = 2^{-k}(-1)^{b_{j-1}\cdot g_{i-1}} \tag{4.3.1.3}$$

b_m is the binary representation of m and g_m is the Gray code representation of m (the easiest way to obtain this is m XORed with m bitshifted right by one bit).

Each set of 2^{n-s-1} multi-controlled rotations can be converted to a uniformly controlled rotation. Hence, in implementing Equation 4.3.1.2 to derive the values of $\vec{\theta}$ from the components of $\vec{\beta^s}$:

```
def get_thetas_helper(beta_s, k):
    return np.dot(mottonen_matrix(k), beta_s)

def get_thetas(betas):
    l = len(betas)
    return [get_thetas_helper(beta_s, l-i-1)
            for (i,beta_s) in enumerate(betas)]
```

And for the matrix used in 4.3.1.3:

```
def binary(n): return np.unpackbits(np.array([n], dtype=np.uint8))

def gray(n): return binary(n ^ n >> 1)

def mottonen_matrix(k):
```

```
    return 2**(-k) * np.array([[(-1)**(np.dot(binary(j), gray(i))%2)
        for j in range(2**k)] for i in range(2**k)])
```

The controls in the CNOTs are these number of gates above the target gate in general: 1,2,1,3,1,2,1,4, etc. This corresponds to OEIS sequence A001511, which, among other things, is one more than the number of trailing zeros in the binary expansion of the number in the sequence (for which there are several methods of obtaining). E.g., 1 has 0 trailing zeros, 2 has 1 trailing zero, 3 has 0, 4 has 2, etc. This is used in the state preparation function:

```
def state_prep(thetas, n):
    qml.RY(thetas[-1][0], wires=0)

    for s in range(n-2,-1,-1):
        for j in range(2**(n-s-1)):
            qml.RY(thetas[s][j], wires=n-s-1)
            qml.CNOT(wires=[max(0,n-s-1-trailing_zeros(j+1)+1),n-s-1])
```

The inference portion remains the same as before.

The variational models I tried took two forms:

The first was to perform the variational adjustments before the encoding angles were derived.

```
def state_prep_var1(x, alphas, betas, n):
    x_transform = alphas * x + betas
    norm = np.sqrt(np.sum(x_transform ** 2, -1))
    x_norm = (x_transform.T / norm).T
    thetas = get_thetas(get_betas(x_norm))
    qml.RY(thetas[-1][0], wires=0)

    for s in range(n-2,-1,-1):
        for j in range(2**(n-s-1)):
            qml.RY(thetas[s][j], wires=n-s-1)
            qml.CNOT(wires=[max(0,n-s-1-trailing_zeros(j+1)+1),n-s-1])
```

The second was to perform the adjustments on the values of θ.

```
def state_prep_var2(thetas, alphas, betas, n):
    qml.RY(alphas[-1][0] * thetas[-1][0] + betas[-1][0], wires=0)

    for s in range(n-2,-1,-1):
        for j in range(2**(n-s-1)):
            qml.RY(alphas[s][j] * thetas[s][j] + betas[s][j]
                  ,wires=n-s-1)
            qml.CNOT(wires=[max(0,n-s-1-trailing_zeros(j+1)+1),n-s-1])
```

4.3.2 Results

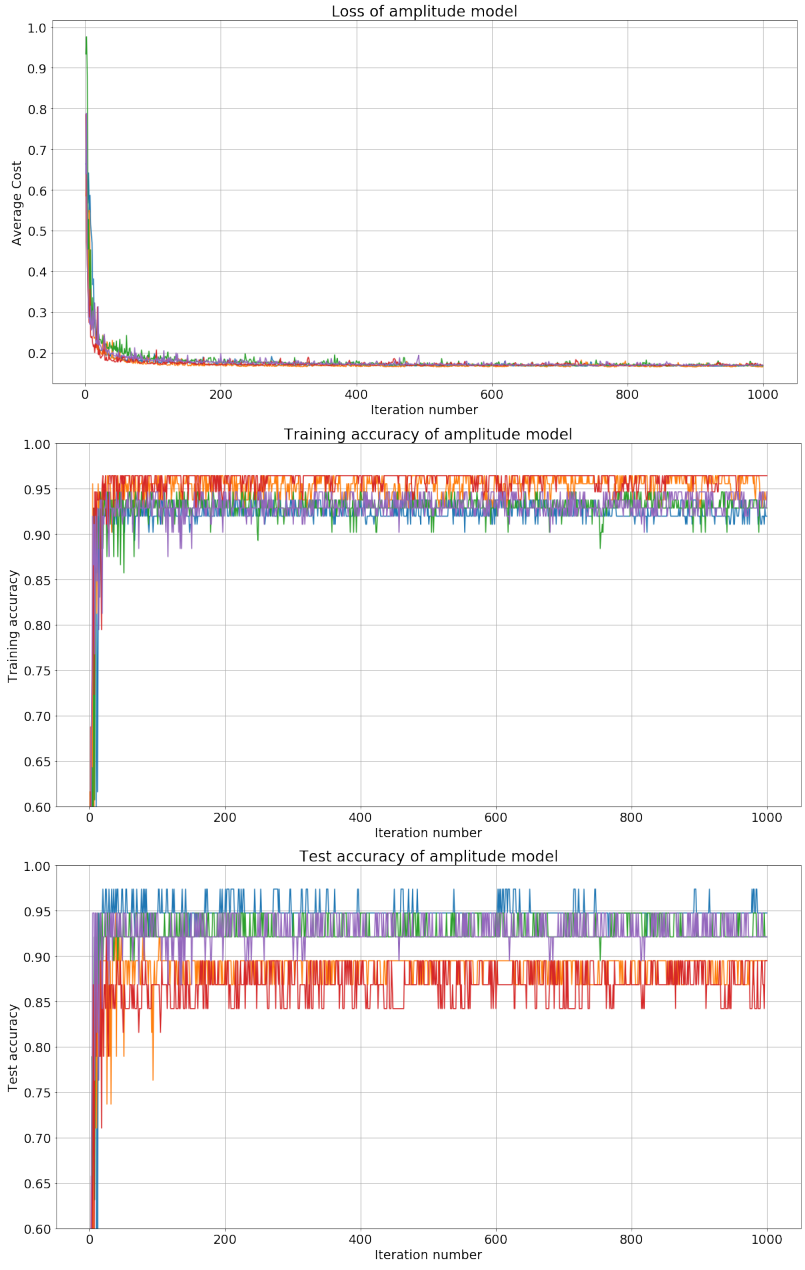

Figure 16. Metrics for amplitude model

Interestingly, these models ran at about five and a quarter hours on average, using the same number of parameters as the Havlíček base model. Average cost reached an average of 0.168 across trials, with training accuracy reaching 0.941 and test accuracy reaching 0.916.

This model is slightly less overfit compared to the Havlíček variational model. Note that the model initially performs better than the variational model, but the variational model catches up. Training accuracy for the variational model improves further, but it is overfit; the test accuracies for the two are about the same.

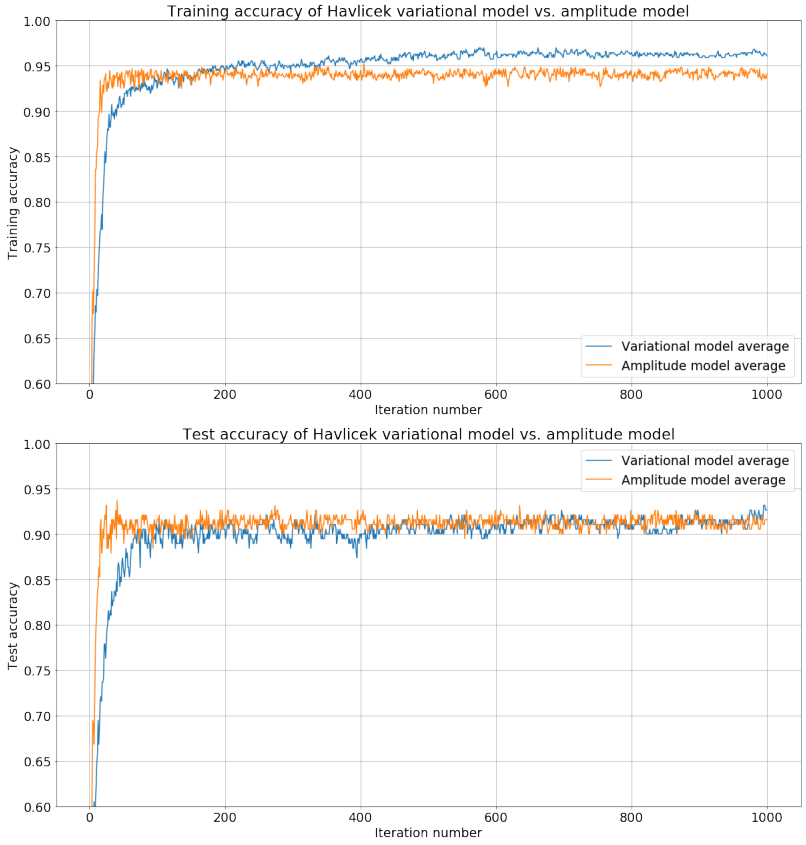

Figure 17. Comparison of Havlíček variational model and amplitude model

4.4 Amplitude Encoding with Measurement Assignment

4.4.1 Differences From Amplitude Encoding Model

The circuit portion of the model remains identical to the previous. Once again, the probabilities are retrieved from the circuit measurements. From these probabilities, a label is assigned based on which measurement state had the highest

probability.

For these experiments, I assigned state $|0000\rangle$ to label 1, state $|0001\rangle$ to label 2, state $|0002\rangle$ to label 3. The other states were ignored, the probabilities for these three states were compared, and the one with the highest probability received the assignment.

The loss used here was categorical cross entropy, which is the sum of the negative log probabilities of the incorrect states.

$$CCE\left(x,y\right)=-\sum_{i\neq y}\log\left(p_{i}\left(x\right)\right)$$

Where y is the label, x are the features, and $p_i\left(x\right)$ is the measurement probability for the state corresponding to label i.

4.4.2 Results

These models ran for about as long as the regular amplitude model, with an average cost of 0.734 (keep in mind this is CCE loss, which is not comparable to the loss we have been using for the other experiments), training accuracy of 0.898, and test accuracy of 0.821.

The performance was significantly worse than the regular amplitude encoding model, as could be expected, since there were fewer parameters available. The accuracy also varies quite a bit, as can be seen in the following figure. I believe it is likely that the worse performance is especially because the bias parameters come after measurement, which is a source of non-linearity.

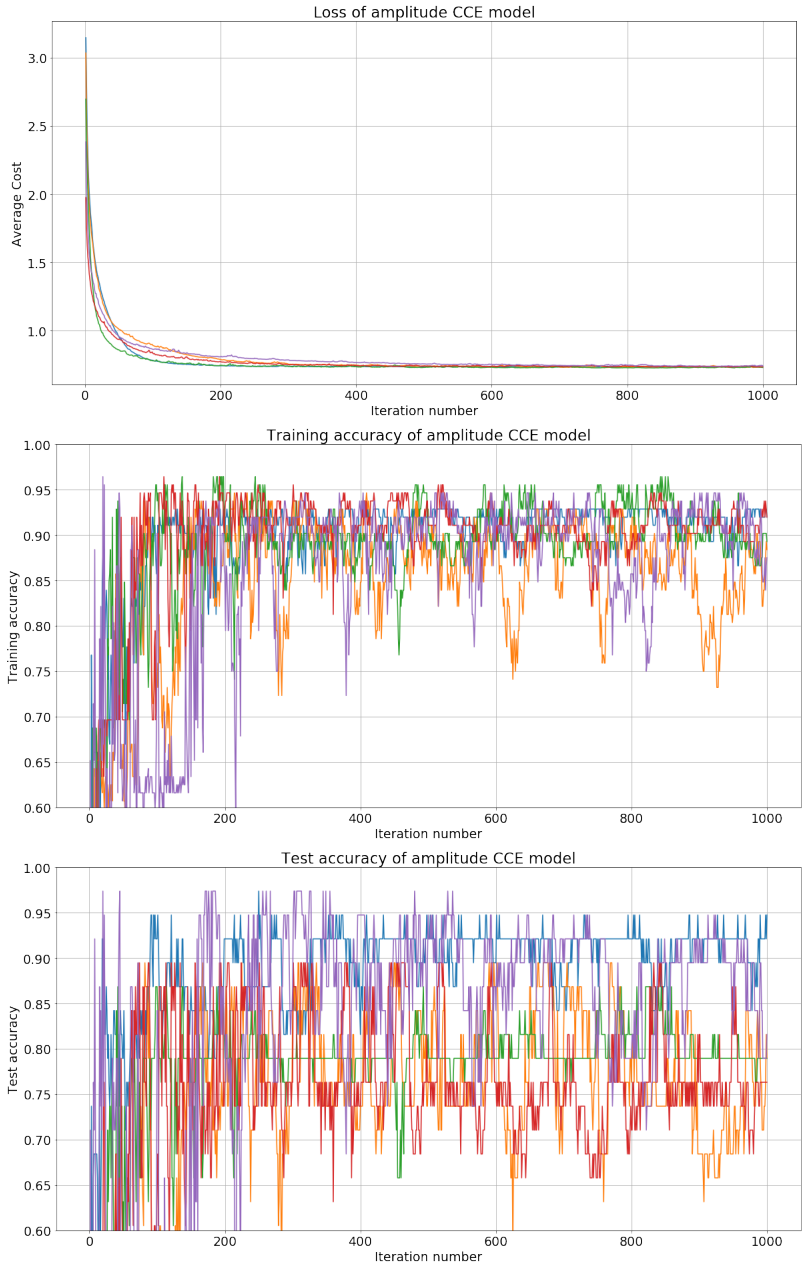

Figure 18. Metrics for amplitude CCE model

4.5 Multi-Circuit Model

4.5.1 Model

The multi-circuit model used the components from the amplitude encoding from before, but the circuit is now a layer:

```
@qml.qnode(dev, interface='torch')
def circuit_layer(layers, angles=None):
    state_prep(angles, num_qubits)
    W_gate(layers)

    return qml.probs(wires=range(num_qubits))
```

Between each circuit layer, I squared the measured results, renormalized, and obtained new angle encodings:

```
def multi_circuit(layers, angles=None):
    for layer in layers[:-1]:
        res = circuit_layer(layer, angles=angles)
        res = res**2
        res /= torch.sqrt(torch.sum(res**2))
        angles = get_thetas(get_betas(res.detach().numpy()))

    return circuit_layer(layers[-1], angles=angles)
```

From there, I proceeded with bias assignment. Note the call to `.detach`. PennyLane does not seem to allow tensors with gradients attached to be used in quantum gates, which may result in some problems with learning.

4.5.2 Results

These models ran for about ten hours each, with an average loss of 0.236, training accuracy of 0.838, and test accuracy of 0.853. Note that the performance was somewhat variable, as two of the trials were somewhat poor, but the other three were competitive with the previous models.

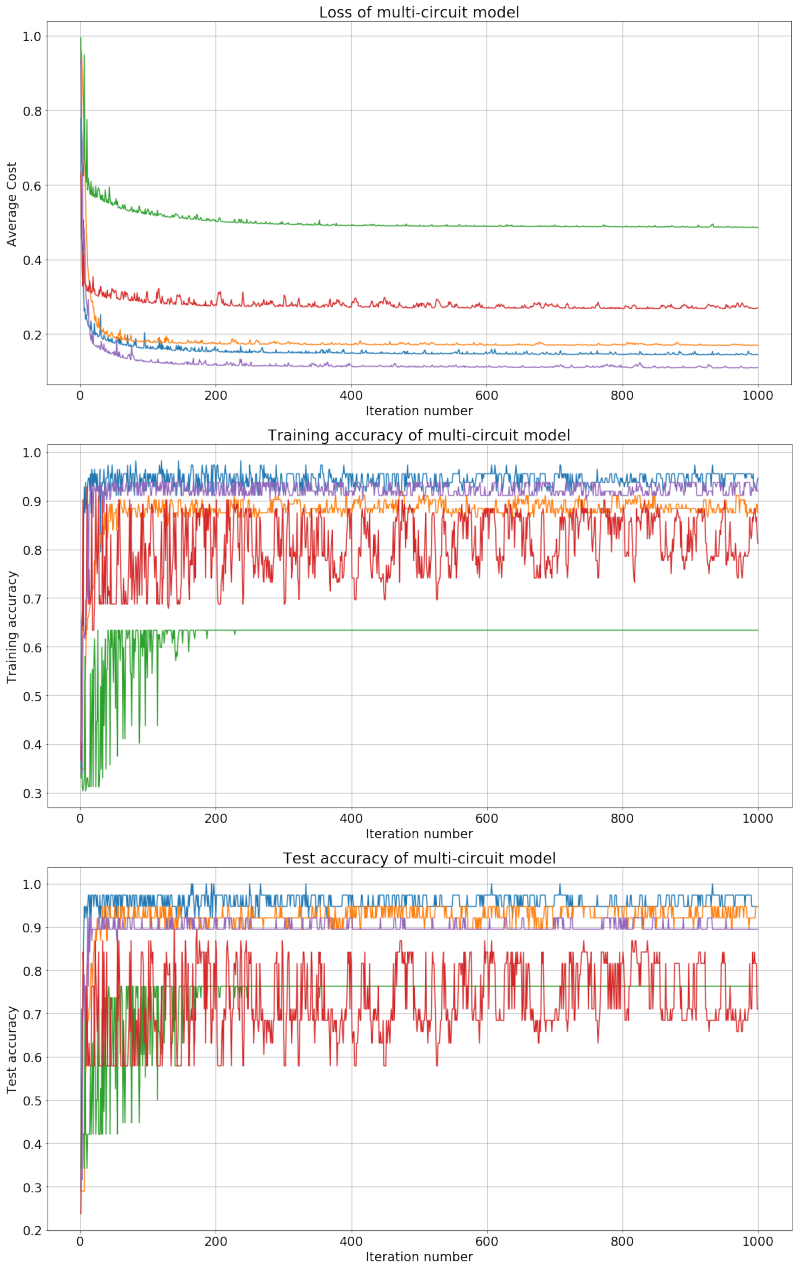

Figure 19. Metrics for multi-circuit model

5 Conclusion

I implemented several models for quantum circuits, comparing encoding methods. Variational encoding as proposed by Theis and Vidal appears to be effective with the Iris dataset. There were some limitations I encountered with PennyLane that I would like to see if they can be overcome, especially in regards to multi-circuit models.

There were some difficulties with the optimizer in the experiments with variational encoding that in future work would need to be overcome.

References

[AW16] Scott Aaronson and Zach Weinersmith. The talk. *Saturday Morning Breakfast Cereal*, Dec 2016.

[BIS+18] Ville Bergholm, Josh A. Izaac, Maria Schuld, Christian Gogolin, and Nathan Killoran. Pennylane: Automatic differentiation of hybrid quantum-classical computations. *ArXiv*, abs/1811.04968, 2018.

[Hal51] Paul R. Halmos. *Introduction to Hilbert space and the theory of spectral multiplicity*. Chelsea Publishing, 1951.

[HCT+19] Vojtěch Havlíček, Antonio D. Córcoles, Kristan Temme, Aram W. Harrow, Abhinav Kandala, Jerry M. Chow, and Jay M. Gambetta. Supervised learning with quantum-enhanced feature spaces. *Nature*, 567(7747):209212, Mar 2019.

[KLM10] Phillip Kaye, Raymond Laflamme, and Michele Mosca. *An introduction to quantum computing*. Oxford University Press, 2010.

[MKT17] Salvatore Mandrà, Helmut G Katzgraber, and Creighton Thomas. The pitfalls of planar spin-glass benchmarks: raising the bar for quantum annealers (again). *Quantum Science and Technology*, 2(3):038501, jul 2017.

[MVBS05] Mikko Möttönen, Juha Vartiainen, Ville Bergholm, and Martti Salomaa. Transformation of quantum states using uniformly controlled rotations. *Quantum Information & Computation*, 5:467–473, 09 2005.

[SP19] Maria Schuld and Francesco Petruccione. *Supervised Learning with Quantum Computers*. Springer, 2019.

[VT19] Javier Gil Vidal and Dirk Oliver Theis. Input redundancy for parameterized quantum circuits. *ArXiv*, abs/1901.11434, 2019.

[Xan] Xanadu. Variational classifier.

Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Andrew Lei**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

Quantum Computing Techniques for Machine Learning,

supervised by Dirk Oliver Theis.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Andrew Lei
10/08/2020