THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF MATHEMATICS

A MEAN-FIELD THEORY FOR QUANTUM NEURAL NETWORKS

RYAN COHEN
SPRING 2024

A thesis
submitted in partial fulfillment
of the requirements
for baccalaureate degrees
in Mathematics
with honors in Mathematics

Reviewed and approved* by the following:

Xiantao Li
Professor of Mathematics
Thesis Supervisor

Carina Curto
Professor of Mathematics
Honors Adviser

*Signatures are on file in the Schreyer Honors College.

Abstract

Quantum machine learning is an emerging technology with the potential to solve many problems in data science. It works with a large quantum circuit with many parameters, so the model and training process are quite complicated. We need to develop a mathematical description for the training problem in the context of a large parameter space. In a previous paper on classical neural networks, it was demonstrated that for neural networks with a growing number of parameters n, the loss function becomes convex, and the approximation error of the network scales as $O(n^{-1})$. In this thesis, we extend those results to a quantum framework. We prove that the loss landscape is asymptotically convex and run numerical experiments to attempt to show that the error scales as $O(n^{-1})$, but conclude that the latter requires further study.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Neural networks and mean-field theory

Neural networks have shown great promise in approximating high-dimensional functions, and have become a prevalent tool in a number of fields. However, despite their impressive results, neural networks are often considered a "black-box" in that their mathematical foundations are not completely understood. Recently, many researchers have adapted mean-field theory from physics in order to better understand the mechanisms of deep neural networks. In essence, mean-field theory is a way to study high-dimensional particle systems by averaging its behavior so that we do not need to consider every particle individually. In the context of neural networks, we can model the parameters as interacting particles that evolve over time, which is the equivalent of the training of the network.

In the context of the previous results from Rotskoff and Vanden-Eijnden [1]., we consider a real-valued function f with d-dimensional inputs approximated by our network

$$f_n(x) = \frac{1}{n}\sum_{i=1}^{n} c_i \varphi(x, y_i) \tag{1.1}$$

where $c_i \in \mathbb{R}$, $y_i \in \mathbb{R}^N$, and φ is some kernel. To study the training of the network, we view our parameters as interacting particles that evolve over time, and hence we can consider f_n as a function of both the number of parameters n and time t:

$$f_n(t, x) = \frac{1}{n}\sum_{i=1}^{n} C_i(t) \varphi(x, Y_i(t)) \tag{1.2}$$

The main results of the paper states that for a given $x \in \mathbb{R}^d$, as the number of parameters n increases, f_n will converge to a function f_0 with an error rate of n^{-1}, i.e.

$$f_n(t) = f_0(t) + O(n^{-1}) \tag{1.3}$$

Furthermore, f_0 evolves on a convex landscape. This is important as it guarantees that training the network will minimize the loss function. This is demonstrated in Figure 1.1: we can see that moving towards in the direction of the negative gradient will always find the global minimum in a convex function, but may cause us to become "stuck" at a local minimum in a non-convex function.

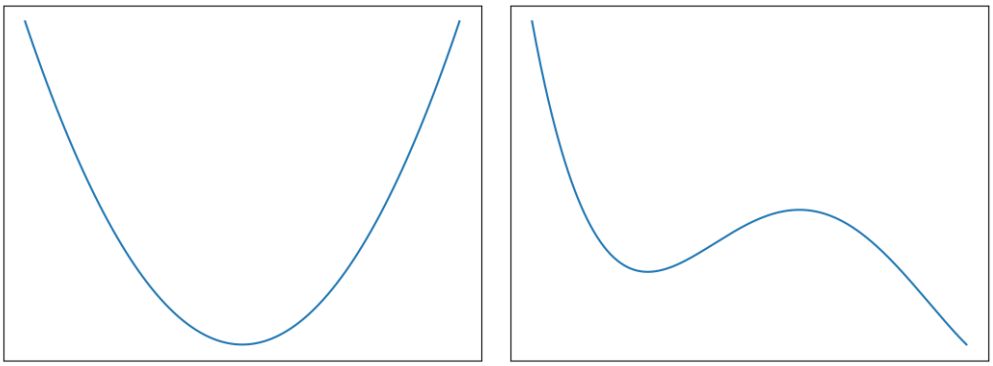

Figure 1.1: A convex and non-convex function

Hence, the convexity of f_0 implies that $f_0(t) \to f$ as $t \to \infty$. In other words, as the number of parameters tends to infinity *and* we continuously train the network, it will converge to the objective function. These results give us insight into the limiting behavior of neural networks. Now, we will attempt to extend these results to quantum neural networks.

1.2 Quantum neural networks

Quantum neural networks, as opposed to their classical counterparts, consist of a quantum circuit with parameterized gates. We encode our data into the circuit's qubits and measure its output compared to our objective function. Then, similar to classical neural networks, we compute the gradient and adjust the circuits parameters to get a more accurate output in later passes.

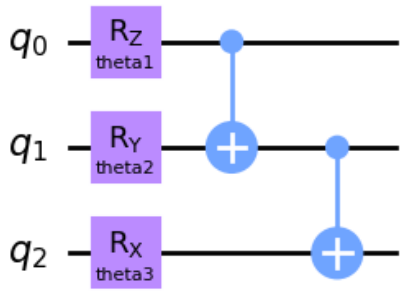

Figure 1.2: Example of a parameterized quantum circuit

In the figure above, we can see a circuit with 3 qubits, each with a parameterized quantum gate and CNOT gates entangling the qubits. More generally, we can represent a quantum machine learning problem as a sequence of parameterized unitary operators acting on some Hermitian operator H which we measure the expectation of:

$$E(\theta) = \langle \psi_0(x) | U(\theta)^\dagger H U(\theta) | \psi_0(x) \rangle \tag{1.4}$$

where $\psi_0(x)$ represents the initial state of our circuit given an input x, $A^\dagger$ is the complex conjugate of a matrix, and $U(\theta) = U_L(\theta_L) \dots U_1(W_1)$, with each $U_i(\theta_i)$ unitary [2]. As we can see, the structure of such a function differs significantly from the classical case. It will take some work to rederive the results on the evolution of such a network.

Chapter 2

Asymptotic Convexity

2.1 The loss function and gradient flow

We would like our approximating function to be in a form more similar to that in equation 1.1. Fortunately, since H is a Hermitian matrix, we can decompose it into the form

$$H = \sum_{j=1}^{n} c_j p_j p_j^\dagger \tag{2.1}$$

where $u_1, \dots u_n$ are orthonormal eigenvectors and $c_1, \dots c_n$ are the corresponding eigenvalues. Denote $P_j = p_j p_j^\dagger$. Then, we propose a class of approximating functions as

$$E_n(x) = \frac{1}{n} \sum_{j=1}^{n} c_j \langle \psi_0(x) | U(\vec{\theta_j})^\dagger P_j U(\vec{\theta_j}) | \psi_0(x) \rangle \tag{2.2}$$

Note that the n^{-1} term will not change the optimization of the parameters. Now, suppose that our domain of inputs is some compact $\Omega \subset \mathbb{R}^d$. Then, we will define our loss function as

$$\ell(E, E_n) = \frac{1}{2} \int_\Omega |E(x) - E_n(x)|^2 dx \tag{2.3}$$

Expanding this, we get

$$\frac{1}{2} \int_\Omega |E(x)|^2 dx - \frac{1}{n} \int_\Omega E(x) E_n(x) dx + \frac{1}{2n^2} \int_\Omega |E_n(x)|^2 dx \tag{2.4}$$

Since the first term is independent of n we can denote it C_E. For the second term, we can move the summation outside of the integral and introduce a shorthand for the integral since it only depends on $\vec{\theta_j}$:

$$\begin{aligned} \int_\Omega E(x) E_n(x) dx = \int_\Omega E(x) \sum_{j=1}^{n} c_j \langle \psi_0(x) | U(\vec{\theta_j})^\dagger P_j U(\vec{\theta_j}) | \psi_0(x) \rangle dx = \\ \sum_{j=1}^{n} c_j \int_\Omega E(x) \langle \psi_0(x) | U(\vec{\theta_j})^\dagger P_j U(\vec{\theta_j}) | \psi_0(x) \rangle dx = \sum_{j=1}^{n} c_j F(\vec{\theta_j}) \end{aligned} \tag{2.5}$$

where $F(\vec{\theta_j})$ represents the integral term. For the last term, we can do a similar process. Expanding the sum, we get

$$\begin{aligned} \int_\Omega |E_n(x)|^2 dx = \int \left| \sum_{j=1}^{n} c_j \langle \psi_0(x) | U(\vec{\theta_j})^\dagger P_j U(\vec{\theta_j}) | \psi_0(x) \rangle \right|^2 dx = \\ \int_\Omega \sum_{i,j=1}^{n} c_i c_j \langle \psi_0(x) | U(\vec{\theta_j})^\dagger P_j U(\vec{\theta_j}) | \psi_0(x) \rangle \langle \psi_0(x) | U(\vec{\theta_i})^\dagger P_i U(\vec{\theta_i}) | \psi_0(x) \rangle dx \end{aligned} \tag{2.6}$$

Then, we can move the summation outside the integral and condense:

$$\sum_{i,j=1}^{n} c_i c_j \int \langle \psi_0(x) | U(\vec{\theta_j})^\dagger P_j U(\vec{\theta_j}) | \psi_0(x) \rangle \langle \psi_0(x) | U(\vec{\theta_i})^\dagger P_i U(\vec{\theta_i}) | \psi_0(x) \rangle dx = \sum_{i,j=1}^{n} c_j c_j K(\vec{\theta}_i, \vec{\theta}_j) \tag{2.7}$$

where $K(\vec{\theta}_i, \vec{\theta}_j)$ represents the integral as a function of two parameters. Hence, the loss function becomes

$$l(E, E_n) = C_f - \frac{1}{n}\sum_{j=1} c_j F(\vec{\theta}_j) + \frac{1}{2n^2}\sum_{i,j=1}^{n} c_j c_j K(\vec{\theta}_i, \vec{\theta}_j) \tag{2.8}$$

and we can rescale it by a factor of n since minimizing ℓ is the same as minimizing $n\ell$, so we will be trying to minimize the function

$$nC_f - \sum_{j=1}^{n} c_j F(\vec{\theta}_j) + \frac{1}{2n}\sum_{i,j=1}^{n} c_i c_j K(\vec{\theta}_i, \vec{\theta}_j) \tag{2.9}$$

Now, we wish to calculate the formulas for gradient flow, a continuous analogue for gradient descent, for optimizing both $\vec{\theta}_j$ and c_j. Notice that this will require finding the gradient of $U(\vec{\theta}_j)$. We know that if U is a unitary matrix, we can represent it in the form e^{-iH}, where H is Hermitian. Hence, we can write

$$U(\vec{\theta}_j) = \left(e^{-i\theta_j^{(1)}V_1}, \ldots, -e^{-i\theta_j^{(L)}V_L}\right) \tag{2.10}$$

with V_k Hermitian and $\theta_j^{(k)} \in \mathbb{R}$, so we can easily calculate the gradient as

$$\nabla U(\vec{\theta}_j) = \left(iV_1 e^{-i\theta_j^{(1)}V_1}, \ldots, iV_L e^{-i\theta_j^{(L)}V_L}\right) \tag{2.11}$$

We must also consider $\nabla U(\vec{\theta}_j)^\dagger$. Since it is the complex conjugate of $\nabla U(\vec{\theta}_j)$, using the product rule leaves us with twice the real part of $\nabla U(\vec{\theta}_j)$. Hence, we will have that

$$\nabla F(\vec{\theta}_j) = 2\int_\Omega E(x) c_j \langle\psi_0|U(\vec{\theta}_j)^\dagger P_j \nabla U(\vec{\theta}_j)|\psi_0\rangle \tag{2.12}$$

and similarly

$$\nabla K(\vec{\theta}_i, \vec{\theta}_j) = 2\int_\Omega \langle\psi_0(x)|U(\vec{\theta}_j)^\dagger P_j \nabla U(\vec{\theta}_j)|\psi_0(x)\rangle\langle\psi_0(x)|U(\vec{\theta}_i)^\dagger P_i U(\vec{\theta}_i)|\psi_0(x)\rangle dx \tag{2.13}$$

The case for c_j is much simpler, the gradient being

$$-F(\vec{\theta}_j) + \frac{1}{n}\sum_{i=1}^{N} C_i K(\vec{\theta}_i, \vec{\theta}_j) \tag{2.14}$$

Then, to model the evolution of $\{c_j, \theta_j\}_{j=1}^n$, we parameterize by time to get $\{C_j(t), \Theta_j(t)\}_{j=1}^n$, viewing our set as a system of interacting particles. Our gradient flows are then

$$\begin{aligned} \dot{C}_j &= F(\vec{\Theta}_j) - \frac{1}{n}\sum_{i=1}^{n} C_i K(\vec{\Theta}_i, \vec{\Theta}_j) \\ \dot{\Theta}_j &= C_j \nabla F(\vec{\Theta}_j) - \frac{1}{n}\sum_{i=1} C_i C_j \nabla K(\vec{\Theta}_i, \vec{\Theta}_j) \end{aligned} \tag{2.15}$$

2.2 The empirical distribution

Now, rather than studying the evolution of the parameters individually, we will look at their empirical distribution. Let

$$\rho_n(t,c,\theta) = \frac{1}{n}\sum_{j=1}^{n}\delta(c - C_j(t))\delta(\theta - \Theta_j(t)) \tag{2.16}$$

where δ is the Dirac delta defined as

$$\delta(x) = \begin{cases}\infty, & x = 0\\ 0, & x \neq 0\end{cases} \tag{2.17}$$

so $\rho_n = 0$ unless (c,θ) is in the set $\{C_j(t), \Theta_j(t)\}_{j=1}^n$. One useful property of the delta function is that for some function f we have

$$\int_\Omega f(x)\delta(x-c)dx = f(c) \tag{2.18}$$

Taking the derivative of ρ_n with respect to t gives

$$\partial_t\rho_n(t,c,\theta) = -\frac{1}{n}\sum_{j=1}^{n}\delta(c - C_j(t)\nabla\delta(\theta - \Theta_j(t))\cdot\dot{\Theta}_j - \frac{1}{n}\sum_{j=1}^{n}\partial_c\delta(c - C_j(t)\nabla\delta(\theta - \Theta_j(t))\cdot\dot{C}_j \tag{2.19}$$

and then we can substitute in our gradient flow equations from 2.15 to get

$$\begin{aligned}-\nabla\cdot\left(\frac{1}{n}\sum_{j=1}^{n}\delta(c - C_j(t))\delta(\theta - \Theta_j(t))\left(\frac{1}{n}c\nabla F(\theta) - \frac{1}{n}\sum_{i=1}c C_i\nabla K(\Theta_i,\theta)\right)\right) -\\ \partial_c\left(\frac{1}{n}\sum_{j=1}^{n}\delta(c - C_j(t))\delta(\theta - \Theta_j(t))\left(\frac{1}{n}F(\theta) - \frac{1}{n}\sum_{i=1}^{n}C_iK(\Theta_i,\theta))\right)\right)\end{aligned} \tag{2.20}$$

and notice that the equation for ρ_n appears twice in this, so we can simplify to

$$-\nabla\cdot\left(\rho_n\left(c\nabla F(\theta) - \frac{1}{n}\sum_{i=1}cC_i\nabla K(\Theta_i,\theta)\right)\right) - \partial_c\left(\rho_n\left(F(\theta) - \frac{1}{n}\sum_{i=1}^{n}C_iK(\Theta_i,\theta))\right)\right) \tag{2.21}$$

Then, using property 2.18, we can insert an integral with two delta functions inside to rewrite the first summation as

$$\frac{1}{n}\sum_{i=1}^{n}cC_i\nabla K(\Theta_i,\theta) = \frac{1}{n}\sum_{i=1}^{n}\iint cc'\nabla K(\theta,\theta')\delta(c'-C_i)\delta(\theta'-\Theta_i)d\theta'dc' = \iint cc'\nabla K(\theta,\theta')\rho_n'd\theta'dc' \tag{2.22}$$

and similarly,

$$\frac{1}{n}\sum_{i=1}^{n}C_iK(\Theta_i,\theta)) = \iint c'K(\theta,\theta')\rho_n'd\theta'dc' \tag{2.23}$$

so the equation becomes

$$-\nabla \cdot \left(\rho_n c \nabla F(\theta) - \iint cc' \nabla K(\theta, \theta') \rho'_n \rho_n d\theta' dc' \right) - \partial_c \left(\rho_n F(\theta) - \iint c' K(\theta, \theta') \rho'_n \rho_n d\theta' dc' \right) \tag{2.24}$$

and then, taking the limit as $n \to \infty$ with ρ_0 as the limit of our empirical distribution, we get

$$\partial_t \rho_0 = \nabla \cdot \left(-\rho_0 c \nabla F(\theta) + \iint cc' \nabla K(\theta, \theta') \rho'_0 \rho_0 d\theta' dc' \right) + \partial_c \left(-\rho_0 F(\theta) + \iint c' K(\theta, \theta') \rho'_0 \rho_0 d\theta' dc' \right) d\theta' dc' \tag{2.25}$$

as the time derivative of the limit of the empirical distribution.

2.3 The limiting approximating function

With the time derivative of the limit of the empirical distribution, we can begin to study the evolution of a network considered to have "infinite" width mentioned in equation 1.3. In the case of our quantum neural network, we can define this network as

$$E_0(t, x) = \iint c \langle \psi_0(x) | U(\vec{\theta})^\dagger P U(\vec{\theta}) | \psi_0(x) \rangle \rho_0(t, \theta, c) \rangle d\theta dc \tag{2.26}$$

since integrating against the limit of the empirical distribution will evaluate the network at its parameters. Now, we wish to rewrite our equation for $\partial_t \rho_0$ in terms of E_0 in in order to study the time derivative of E_0. Firstly, we will expand the terms F and K to get the rather messy expression

$$\begin{aligned} &\nabla \cdot \Bigg(- c \nabla \int E(x) \langle \psi_0(x) | U(\theta^\dagger P U(\theta) | \psi_0(x) \rangle dx \rho_0 + \\ &\int cc' \nabla \int \langle \psi_0(x) | U(\theta)^\dagger P U(\theta) | \psi_0(x) \rangle \langle \psi_0(x) | U(\theta')^\dagger P_i U(\theta') | \psi_0(x) \rangle \rho_0 \rho'_0 dx d\theta' dc' \Bigg) + \\ &\partial_c \Bigg(- \int E(x) \langle \psi_0(x) | U(\vec{\theta})^\dagger P U(\vec{\theta}) | \psi_0(x) \rangle dx \rho_0 + \\ &\int c' \int \psi_0(x) | U(\theta)^\dagger P U(\theta) | \psi_0(x) \rangle \langle \psi_0(x) | U(\theta')^\dagger P_i U(\theta') | \psi_0(x) \rangle \rho_0 \rho'_0 dx d\theta' dc' \Bigg) \end{aligned} \tag{2.27}$$

Then, we can move the terms depending on θ and c outside the innermost integral since they do not depend on θ' or c' and rewrite in terms of E_0 to get

$$\begin{aligned} &\nabla \cdot \left(- c \nabla \int E(x) \langle \psi_0(x) | U(\theta)^\dagger P U(\theta) | \psi_0(x) \rangle dx \rho_0 + \int c \nabla_\theta \langle \psi_0(x) | U(\theta)^\dagger P U(\theta) | \psi_0(x) \rangle E_0(x) dx \rho_0 \right) + \\ &\partial_c \left(- \int E(x) \langle \psi_0(x) | U(\vec{\theta})^\dagger P U(\vec{\theta}) | \psi_0(x) \rangle dx \rho_0 + \int \langle \psi_0(x) | U(\theta)^\dagger P U(\theta) | \psi_0(x) \rangle E_0(x) dx \rho_0 \right) \end{aligned} \tag{2.28}$$

Then, applying the gradient to the term $\langle \psi_0(x) | U(\theta)^\dagger P U(\theta) | \psi_0(x) \rangle$ we can rearrange and factor to get

$$\partial_t \rho_0 = \nabla \cdot \left(c \int \langle \psi_0 | U(\theta)^\dagger P \nabla U(\theta) | \psi_0 \rangle (E_0(t, x) - E(x)) dx \rho_0 \right) + \partial_c \left(\int \langle \psi_0 | U(\theta)^\dagger P U(\theta) | \psi_0 \rangle dx \rho_0 \right) \tag{2.29}$$

With $\partial_t \rho_0$ in terms of E_0, we begin to study $\partial_t E_0$. Using 2.26, we have that

$$\partial_t E_0 = \iint c\langle\psi_0(x)|U(\vec{\theta})^\dagger P U(\vec{\theta})|\psi_0(x)\rangle \partial_t \rho_0(t,\theta,c) d\theta dc \tag{2.30}$$

and substituting in 2.29 we have

$$\iint c\langle\psi_0(x)|U(\vec{\theta})^\dagger P U(\vec{\theta})|\psi_0(x)\rangle \nabla \cdot \left(c \int \langle\psi_0|U(\theta)^\dagger P \nabla U(\theta)|\psi_0\rangle (E_0(t,x) - E(x)) dx \rho_0 \right) d\theta dc + \iint c\langle\psi_0(x)|U(\vec{\theta})^\dagger P U(\vec{\theta})|\psi_0(x)\rangle \partial_c \left(\int \langle\psi_0|U(\theta)^\dagger P U(\theta)|\psi_0\rangle dx \rho_0 \right) d\theta dc \tag{2.31}$$

To simplify, we can integrate both terms by parts. In each term, change the order of integration so that x is being integrated after θ and c. However, we have to add in a new variable x' since the first x in each term is not being integrated. Combining these, the equation becomes

$$\iint \Big(- c^2 \langle\psi_0(x)|U(\theta)^\dagger P \nabla U(\theta)|\psi_0(x)\rangle \langle\psi_0(x')|U(\theta)^\dagger P \nabla U(\theta)|\psi_0(x')\rangle - \langle\psi_0(x)|U(\theta)^\dagger P \nabla U(\theta)|\psi_0(x)\rangle \langle\psi_0(x')|U(\theta)^\dagger P \nabla U(\theta)|\psi_0(x')\rangle \Big) \rho_0 (E_0(t,x') - E(x)') d\theta dc dx' \tag{2.32}$$

Then, let

$$M(\rho, x, x') = \iint \Big(c^2 \langle\psi_0(x)|U(\theta)^\dagger P \nabla U(\theta)|\psi_0(x)\rangle \langle\psi_0(x')|U(\theta)^\dagger P \nabla U(\theta)|\psi_0(x')\rangle + \langle\psi_0(x)|U(\theta)^\dagger P \nabla U(\theta)|\psi_0(x)\rangle \langle\psi_0(x')|U(\theta)^\dagger P \nabla U(\theta)|\psi_0(x')\rangle \Big) \rho_0 d\theta dc \tag{2.33}$$

so we end up with

$$\partial_t E_0(t,x) = - \int M(\rho, x, x') (E_0(t,x') - E(x')) dx' \tag{2.34}$$

as the equation for the time derivative of E_0. Note that for vectors, the general quadratic form is $x^T A x$ and its derivative is $2Ax$. If we consider E_0 as an infinite dimensional vector, then 2.34 shows that it must have a quadratic form, where M is the linear operator A and it is integrated over x'. Hence, the limiting approximating function is quadratic and thus evolves on a convex landscape.

Chapter 3

Numerical Results on Scaling of Network

3.1 Quantum neural network setup

In the original paper by Rotskoff and Vanden-Eijnden, they derived equation 1.3 showing that the empirical loss of the network scales at $O(n^{-1})$ and verified the results computationally [1]. While the formal derivation of this is rather technical, we can still attempt to reproduce their results numerically on a quantum on a quantum simulator.

We will be approximating the spherical 3-spin model on the 4 dimensional hypersphere in $\mathbb{R}^5$ defined as

$$f(x) = \frac{1}{d} \sum_{p,q,r=1}^{5} a_{p,q,r} x_p x_q x_r \tag{3.1}$$

where $\{a_{p,q,r}\}_{p,q,r=1}^{5}$ are randomly generated from a standard Gaussian distribution and each x lies on the surface of the 4-sphere. Specifically, our points will be of the form

$$\sqrt{5}(\sin(\theta)\cos(\varphi), \sin(\theta)\sin(\varphi), \cos(\theta), 0, 0) \tag{3.2}$$

where $\theta \in [0, \pi]$ is drawn uniformly and $\varphi \in [0, 2\pi]$ is fixed. This function has a large number of local minima so it will be difficult to optimize our loss function.

To create the quantum neural network, we must first designate a feature map and an ansatz. A feature map is a function used to encode data into a qubit. One commonly used method is to scale each vector element into $[-\pi, \pi]$ and encode them as the amplitude of angles [3] as

$$\psi_n^d = \cos(x_n^d)|0\rangle + \sin(x_n^d)|1\rangle \tag{3.3}$$

Then, we have to choose an ansatz, which contains the parameters that we optimize over. Choosing an ansatz is a difficult problem that is often highly dependent on the specifics of our target function, but for our purposes we only wish to see how the loss function scales with the width of the network. We will borrow an ansatz conventionally used for estimating the ground state energy of hydrogen atoms, which consists of an alternating series of R_Z and R_Y gates followed by several CNOT gates [4]. An example circuit with 5 qubits is depicted below:

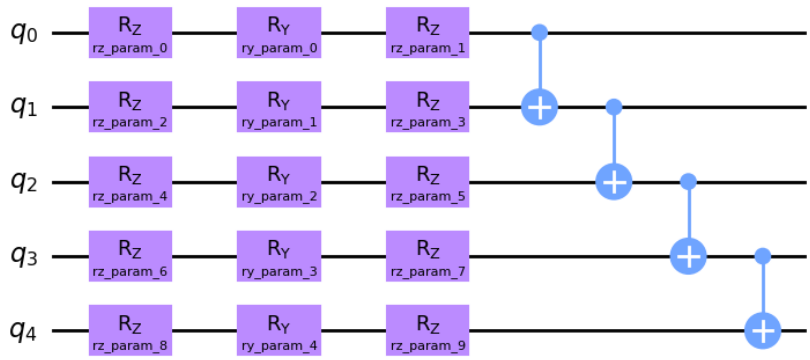

Figure 3.1: An ansatz with 5 qubits

Before our data passes through the quantum network, we will first pass it through one linear layer of a classical neural network to adjust the width. Then, the output of the classical layer will be encoded by the feature map. We will use the Adam optimization algorithm and the MSE loss function.

3.2 Results of simulation

We trained $n = 1, 2, \ldots 10$ layers for one realization of the target function, each for 200 iterations. The loss at the end of training is shown below:

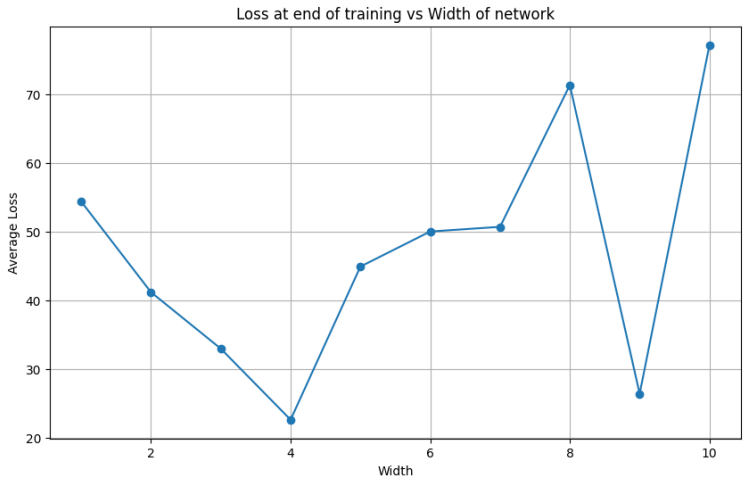

Figure 3.2: Loss after training compared to width of network

As we can see, there is no apparent pattern between layers and average loss at the end of training. There are a few reasons this could be the case. The most obvious reason is due to not dedicating enough computational resources. In classical neural networks, a network of width n involves calculations with $n \times n$ matrices. In the quantum case, since a quantum state is calculated through the tensor product of qubit states, an n qubit network will involve calculations with $2^n \times 2^n$ matrices. Thus, the training of the network takes significantly more time and resources. In the original paper, the network was trained with up to 256 layers for over one million iterations. In contrast, we only trained for up to 10 layers for 200 iterations. It is possible that the $O(n^{-1})$ pattern only becomes apparent over for sufficiently trained networks. Below, we have the training for $n = 4$:

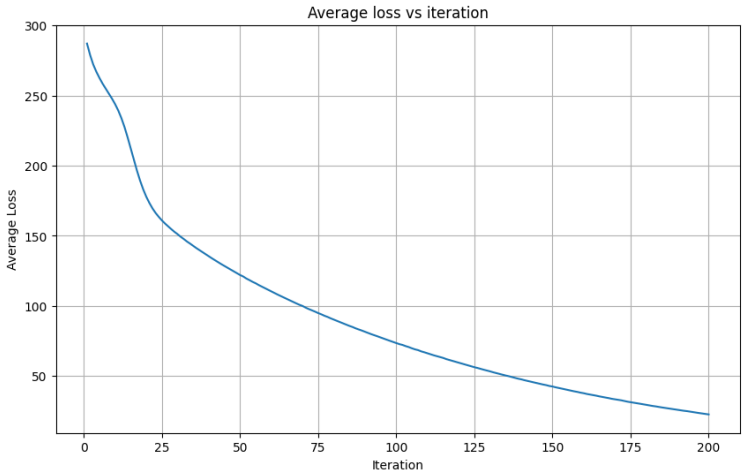

Figure 3.3: Average loss over time for network of width 4

We can see that the loss does not plateau at any point; it is likely that we still could have reduced the loss further with more iterations. The final loss may have become too dependent on the initial weights, rather than the width of the network. Further studies should be made into this, with more computing time and power.

The other possibility is that the $O(n^{-1})$ loss scaling simply does not hold in the case of quantum neural networks. Due to their fundamental differences in structure, it is possible that a similar result is impossible to derive, and the relationship between network width and loss is more complicated. For instance, it is known that randomly initialized parameters run into the issue of the gradient becoming exponentially small in the number of qubits [2]. However, the original derivation of this was beyond the scope of this paper, and thus requires further study.

Bibliography

[1] Grant Rotskoff and Eric Vanden-Eijnden. Trainability and accuracy of artificial neural networks: An interacting particle system approach. *Communications on Pure and Applied Mathematics*, 75(9):1889–1935, September 2022.

[2] Edward Grant, Leonard Wossnig, Mateusz Ostaszewski, and Marcello Benedetti. An initialization strategy for addressing barren plateaus in parametrized quantum circuits. *Quantum*, 3:214, December 2019.

[3] Edward Grant, Marcello Benedetti, Cao Shuxiang, Andrew Hallam, Joshua Lockhart, Vid Stojevic, Andrew G. Green, and Simone Severini. Hierarchical quantum classifiers. *npj Quantum Information*, 4:65, December 2018.

[4] Yuxuan Du, Tao Huang, Shan You, Min-Hsiu Hsieh, and Dacheng Tao. Quantum circuit architecture search for variational quantum algorithms. *npj Quantum Information*, 8:62, May 2022.