QUANTUM ACTIVATION FUNCTIONS FOR NEURAL NETWORK REGULARIZATION

by

Christopher Hickey

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science

Department of Computer Science

West Lafayette, Indiana

August 2023

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Wojciech Szpankowski, Chair

Department of Computer Science

Dr. Andreas Jung, Co-Chair

Department of Physics and Astronomy

Dr. Ananth Grama

Department of Computer Science

Approved by:

Dr. Kihong Park

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

The Bias-Variance Trade-off, where restricting the size of a hypothesis class can limit the generalization error of a model, is a canonical problem in Machine Learning, and a particular issue for high-variance models like Neural Networks that do not have enough parameters to enter the interpolating regime. Regularization techniques add bias to a model to lower testing error at the cost of increasing training error. This paper applies quantum circuits as activation functions in order to regularize a Feed-Forward Neural Network. The network using Quantum Activation Functions is compared against a network of the same dimensions except using Rectified Linear Unit (ReLU) activation functions, which can fit any arbitrary function. The Quantum Activation Function network is then shown to have comparable training performance to ReLU networks, both with and without regularization, for the tasks of binary classification, polynomial regression, and regression on a multicollinear dataset, which is a dataset whose design matrix is rank-deficient.

The Quantum Activation Function network is shown to achieve regularization comparable to networks with L2-Regularization, the most commonly used method for neural network regularization today, with regularization parameters in the range of $\lambda \in [.1, .5]$, while still allowing the model to maintain enough variance to achieve low training error. While there are limitations to the current physical implementation of quantum computers, there is potential for future architecture, or hardware-based, regularization methods that leverage the aspects of quantum circuits that provide lower generalization error.

1. INTRODUCTION

Deep Learning - Machine Learning with Deep Neural Networks - has demonstrated successful application in many fields where data is abundant such as Engineering, Finance, Medicine, and the Physical Sciences, and has become the central research focus in the field of Machine Learning over the past decade. As with other Machine Learning methods the goal of Deep Learning is to develop a model that expresses the distribution $p_{model}(\mathbf{x}; \theta)$ where

$$p_{model}(\mathbf{y} \mid \mathbf{x}; \theta) \approx \ p_{\mathbf{data}}(\mathbf{y} \mid \mathbf{x}) \tag{1.1}$$

Here, the data points used to train the model are drawn from the unknown distribution p_{data}. When expressed as individual data tuples, the vector $\mathbf{x}$ is knowna s the feature vector, while $\mathbf{y}$ is the target vector. Therefore, a learning algorithm seeks to learn the set of parameters θ in order to approximate this conditional distribution x and y are drawn from, assuming that such a distribution can be expressed by a parameterized distribution over the same set of data points. While there are numerous neural network architectures for specific tasks such as Convolutional Neural Networks for Computer Vision and Transformers for Natural Language Processing, this paper will be focused on the simple Multilayer Perceptron, which is a feedforward network that consists of a sequence of fully-connected linear layers with non-linear activation functions between them. The linear layers $\mathbf{W}$ of the network are represented by matrices, and then a vector of additive values known as a bias, whose elements are optimized to minimize a specified loss function, while the non-linear activation function allow the network. For the purposes of this paper, all neural networks used will be "single hidden layer" networks (although they are technically 2-layer networks consisting of an input layer in addition to a single hidden layer), which is any linear layer of a feedforward network after the first layer that operates on the vector of values, and has the general form of:

$$N(\mathbf{x}) = \mathbf{h^T}\sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{1.2}$$

Where σ is the non-linear activation function that is applied at the single hidden layer, and $\mathbf{h^T}$, $\mathbf{W}$, and $\mathbf{b}$ are learnable parameters, as in the values that the optimization algorithm will select over, that will perform 4-to-1 mappings for three different machine learning tasks. The following visualization may also be used to understand a single hidden layer network:

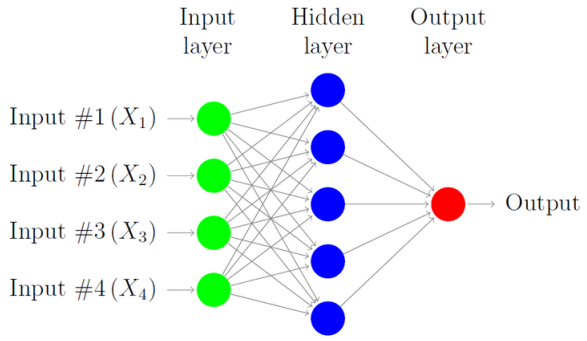

Figure 1.1. A Single Hidden Layer Neural Network

This formulation of a Neural Network results in a piece-wise linear mapping that can be fit to express an arbitrary function in order to map the set of input data points to their corresponding outputs. All tasks considered in this paper will also be Supervised Learning problems, where for each input of an independent variable there is a paired output value, and the mapping between them is assumed to be learned parameterically. The measure a networks prediction accuracy is given by it's Empirical Risk or it's "loss", which is represented with $\mathcal{L}(N(\mathbf{X}), \mathbf{Y})$. This means that the output of a network $N(\mathbf{X})$ when passed the elements of the design matrix $\mathbf{X}$) is compared against the matrix of labelled target data $\mathbf{Y}$ in order to judge how successful a network is. The loss is usually evaluated for each data point with an elementwise loss function l:

$$\mathcal{L}(N(\mathbf{X}), \mathbf{Y}) = \sum_{\mathbf{x}\in\mathbf{X}),\mathbf{y}\in\mathbf{Y}} l\left(\mathbf{x}, \mathbf{y}\right) \tag{1.3}$$

The two primary phases of a training algorithms process are the training stage, where network parameters are adjusted to minimize a specified loss function, which is any metric on the space of values the target vector is a member of, and then a testing stage where the testing, or "generalization", error is measured by the networks performance on datapoints not provided in the training set. This demonstrates two challenges to the machine learning

practitioner, first the error provided by the loss function during the training stage must decreasingly converge to demonstrate that the network can indeed "learn" the data set at all, but then the actual value of a neural network is demonstrated by its ability to then apply the predictions its learned to unseen data.

1.1 The Approximation of Arbitrary Functions with Neural Networks

The success of Deep Neural Networks versus other Machine Learning methods is based on the former's ability to learn a mapping that approximates the conditional relation of an arbitrary data distribution. For example, a neural network with at least one hidden-layer is able to approximate any function to an accuracy such that an error between any output of the network and the corresponding output of the target function will be bounded by some constant ϵ, that is:

$$| N(\mathbf{x}) - \mathbf{f}(\mathbf{x}) | < \epsilon \tag{1.4}$$

where $N(x)$ is the expression of the neural network in Eq. 1.2, while $f(\mathbf{x})$ is the target function to be approximated. [1] However, an issue often faced when applying Neural Networks is the "memorization" phenomena, where a model will overfit the exact mappings provided during the training phase without being able to generalize this success to unseen data. This means that while training error may be very small at the end of the training stage, the distribution being expressed by the network is not approximating the data distribution and so it is not able to successfully map the data later provided during the testing stage. [2]

This problem is particularly prevalent for tasks sampled with few datapoints or sparse design matrices, which are the datapoints for a problem organized in a matrix. Single hidden layer networks using the ReLU activation function on their hidden layer, which performs the operation:

$$\sigma_{ReLU}(\mathbf{x}) = \begin{bmatrix} max(0, x_0) \\ max(0, x_1) \\ \vdots \\ max(0, x_n) \end{bmatrix} \tag{1.5}$$

can memorize and overfit a number of datapoints on the order of the number of hidden layers in a network. [3] This count of hidden layer nodes in a single hidden layer network is known as the width of the network.

1.2 The Bias-Variance Trade-off

The problem of overfitting in Machine Learning is a consequence of optimizing over the parameter set for a model with high variance, meaning the network is learning to express a distribution whose values will tend to be sampled far from the expectation value. Conversely, the bias of a distribution that a model learns to express is a measure of how tightly predicted points group around the expectation value of the distribution. In machine learning research the inverse relation between the bias and variance of the network, and the associated consequences on the statistical learning a model is able to achieve, is known as the Bias-Variance Tradeoff. [4]

In the case of Deep Neural Networks, the problem for many tasks is not that the distribution learned by the model has too high of bias, but that it has far too high variance. This manifests in the model overfitting data points without learning the underlying distribution that the data is drawn from. This is observed in practice when a model has a very low training error, but then returns a high error during the testing stage. The generalization gap, which is the absolute value of the difference between the training error and testing error, is the primary measure for the balance of bias and variance in a neural network. It is more desirable to have a low generalization gap, which is usually a result of low variance function approximation, rather than a high generalization gap even if this means the training error is higher than it would otherwise be. The reason for this preferential behavior is because the actual practical use of machine learning models is to deploy the models in practical,

real-world applications where the output of the function to be learned is desired, rather than just in the training stage where the output is already known with the target data labels. However, the model must also not have too much bias as to result in excessively high training error where the learning algorithm never converges during the training stage. This would lead to a high training and testing error, and therefore a misleadingly low generalization gap, so it is also necessary to look at the networks performance both during the training and testing stages, and not just the generalization gap. This means a good measure of how much bias and variance a network is expressing is found by observing the training errors progress over the training stage steps and then periodically measuring how successful the model is at the predicting the testing data mapping, all without using those testing set trials to update the network parameters. [5]

The Hypothesis class, which is the set of all possible mappings the network can express, is based on both the architecture of the model as well as what parameters the optimization algorithm can converge to select. The Hypothesis class's structure limits what the end result of the training stage will be, as it must be a member of this class. The capacity of a hypothesis class is a measure of the size of the class, although in practice it is often not written explicitly. For example, consider the capacity of the class of functions that take the general form:

$$\mathcal{H}_1 = \{\hat{y_1}(x_1, x_2) = \alpha x_1 + \gamma \mid \alpha, \gamma \in \mathbb{R}\} \tag{1.6}$$

and then compare to:

$$\mathcal{H}_2 = \{\hat{y_2}(x_1, x_2) = \alpha x_1 + \beta x_2 + \gamma \mid \alpha, \beta, \gamma \in \mathbb{R}\} \tag{1.7}$$

While both of classes, convergence of the optimization algorithm permitted, can express an infinite amount of functions, the relative higher complexity of $\mathcal{H}_2$ compared to $\mathcal{H}_1$, thanks to the presence of an addition term, allows it to be said that:

$$\mid \mathcal{H}_2 \mid > \mid \mathcal{H}_1 \mid \tag{1.8}$$

where $\mid \mathcal{H}_1 \mid$ and $\mid \mathcal{H}_2 \mid$ are the capacities of $\mathcal{H}_1$ and $\mathcal{H}_2$, respectively. Since the members of $\mathcal{H}_2$ can express all the members of $\mathcal{H}_2$ by setting $\beta = 0$, and then additional functions by setting $\beta \neq 0$. Of course, one of these classes looks to just "throw away" one of the input values, but it can be seen how limiting the number of functions a hypothesis class member can express will lower the capacity of that class. This also demonstrates how, while difficult to find explicitly and almost never done in practice, the capacity of a Deep Neural Network model is a function of the size of the network, as well as architectural layout of that network.

The high capacity of the hypothesis class of models implemented with neural networks is evident in their ability to arbitrarily fit functions as universal approximators. Vapnik and Chervonenkis show that the generalization gap of a model is upper bounded by a function of the capacity of it's hypothesis class. For machine learning algorithms, there is a careful balancing methodology to have a high enough hypothesis class capacity to learn a data set while still ensuring the class doesn't have too high of a capacity so that it may still generalize to unseen data. This implies that there is an optimal capacity for a hypothesis that that isn't too small as to underfit the data and not too great as to overfit the data. Since, without appropriate accommodation, Deep Neural Networks can have some of the highest capacity hypothesis classes of all machine learning models, efforts must often be made to limit the capacity of the networks hypothesis class in order to achieve lower generalization error, and therefore better performance in deployment. Another result from is that the generalization gap can also be lowered by increasing the number of training samples processed during the training stage.[6] However, for many ad-hoc applications of deep learning, there may not be enough training samples available to sufficiently lower the generalization gap, and so efforts must be made to limit the capacity of the hypothesis class.

The complexity for most machine learning models can be modified by adjusting how dramatically individual learnable parameters affect the mapping. For example, as in Eqn. 1.5, the length of the input vector in a regression problem can be intentionally limited to be smaller than what is supplied from the data distribution, and this can prevent a learning algorithm from over-representing certain feature vector values if. In practice, it is very difficult to know which of the features the network accepts are irrelevant, and often impossible since little is likely known about the parameterization of the data distribution.

The "throw away" example in Eqn. 1.7 cannot often be wantonly done in practice and so alternative methods of limiting the capacity of a hypothesis class must be sought. One of the most common ways this is done in practice is by "dampening" how large certain parameters become over the course of training, which can achieve the same intention as the "throw away" example without having to commit to always having some parameter equal 0. There is also the commonly used drop-out regularization, which shuts off certain network parameters with a selected probability each time the network is evaluated. This is also similar to the "throw away" example, although with greater flexibility since it is not necessarily the same parameter being set to 0 each time.

The generalization gap bound demonstrated by Vapnik and Chernovenkis shows it is conceptually important to note that if two models both are able to fit data equally well, the simpler of the two would be preferable, as a model with a lower complexity will have a smaller hypothesis class capacity. The generalization gap being shown as a function of the hypothesis class is usually only formally shown in the context of classification tasks where the Vapnik-Chernovenkis (VC) Dimension can be explicitly found in order to then give a bound. For other more complicated tasks the capacity of a hypothesis class limiting the generalization gap is used as an intuitive justification for increasing the bias of a model. This would also lead to a tighter upper bound on the generalization gap of that model in practice. This principle is known as Occam's Razor and provides further confidence for searching for methods for increasing the bias of a model at the cost of its variance. [5]

1.3 The Modern Interpolating Regime

One important note to make is an empirical result challenging the orthodoxy of the Bias-Variance Tradeoff, demonstrated by Belkin *et al.* They show that if the number of parameters, and therefore the hypothesis class capacity, far exceed the overfitting range there is actually a decrease in testing loss. The authors note an example that if the network has as many parameters as the product of the test examples and the number of classes for a classification task, then the network will be able to "interpolate" the mappings, meaning the testing loss

will converge to 0. Once the capacity of a hypothesis class exceeds this "interpolating" threshold, the network begins to exactly map the test data to the exact result.

This is part of a larger trend in Deep Learning research where "bigger is better", and publications seek to find methods in hardware and architecture to construct very large models that can still be realistically computed upon in efficient ways. This may imply that the Bias-Variance Tradeoff is an obsolete assumption, and methods for increasing the bias of a network are not worthwhile endeavours. However, since the capacity of a Neural Network is proportional to its size, there is a still the potential problem that machine learning practioners may not be able to implement large enough models. For example, the authors note that in order to interpolate the ImageNet classification set, a neural network would need to exceed 10^9 parameters. While many successful Deep Learning Models will be trained with access to significant computational resources, often in the form of large cloud-computing storage facilities, there are still instances where such access is infeasible. This is particularly true for embedded system development or smaller organizations and individuals with limited resources. Furthermore, the training period of a network is also elongated by the size of a model, particularly for many industry model that may have upwards of 10^6 parameters, and so if there is a time constraint on the training stage of a Neural Network model this may also be impossible to achieve. For this reason, regularization research is still an active and productive topic in Machine Learning research.

1.4 Regularization

Regularization is the practice of intentionally increasing a model's bias and therefore it's training error, in order to lower its variance, and therefore it's testing error. Of course, an excessive amount of bias will prevent a model from accurately predicting a data set, and so when first determining the validity of a model a low enough training loss must be shown. Since variance is so abundant in the raw application of Deep Neural Networks, particularly ReLU networks, regularization is more proactively sought to achieve a lower generalization gap.

One type of Regularization is to modify the loss function the learning algorithm optimizes with an additional term. For example, L2-Regularization looks to limit the effective capacity of a hypothesis class by adding a term of the 2-norm of the parameter weights arranged in a vector:

$$\mathcal{L}_{L2}(N(\mathbf{X}), \mathbf{Y}) = \sum_{\mathbf{x}\in\mathbf{X},\mathbf{y}\in\mathbf{Y}} l\left(\mathbf{N}(\mathbf{x}), \mathbf{y}\right) + \lambda\|\mathbf{w}\|_{\mathbf{2}} \tag{1.9}$$

where λ is known as the "Regularization Parameter", which is a constant that introduces a greater amount of bias into the model as it is increased. Since a learning algorithm seeks to minimize this loss function, this additional term punishes the increase of the values in the weight parameter vector, which prevents the weights from getting continuously increased at each training set. When the weights are prevented from getting too large, the contribution of the features to the output of the network becomes less dramatic, and this prevents the network from expressing the set of functions where those weights are much more significant than allowed with L2-Regularization. [4] This type of regularization limits the effective capacity of the model, meaning that while the actual size and shape of the network may architecturally admit a larger group of functions, the optimizer will not set the parameters to the weights that allow the network to express those functions. L2-Regularization is the most common type of regularization used in Deep Learning.

Alternatively, Regularization methods can limit the representational capacity, which this paper is concerned with, of a hypothesis class, which would mean limiting the amount of the expressible functions that can be fit by a model over it's parameters. This means that the network architecture itself is modified so as to admit a smaller class of functions to be expressed. The most common technique for this is "dropout" regularization, which is shuts off each parameter in a neural network with some set probability at each evaluation by setting it to 0. This is similar to the "throw away" example from the section on the Bias-Variance Tradeoff, although not as dependent on knowing which parameters are most helpful to shutoff beforehand in order to limit the capacity of a hypothesis class.

1.5 Nonlinear Activation Functions

The universal expressive power of a single hidden layer feedforward network holds as long as the applied non-linear activation function does not apply a polynomial function to the feature vector. However, the activation function must also be non-linear in order for the network to be able to fit arbitrary function piecewise. This paper is concerned with using quantum circuits as activation functions for Neural Networks, but all operations performed on the qubits, the computational medium of quantum computers, can be represented as matrices, which are linear transformations. Therefore, there must be some additional non-linearity operation performed alongside the quantum circuit operation, which will later be shown to be a series of inverse tangent operations to encode the feature vector into the qubits the quantum circuits will operate on.

The motivation for using a quantum circuit as a Regularizing activation function comes from Kobayashi *et al.* who note that neural networks restricted to perform unitary operations, which are represented by matrices whose adjoint is their own inverse, will be resistant to overfitting. Any Unitary operation preserves the 2-norm of a vector, since:

$$\|\mathbf{Ux}\|_{\mathbf{2}} = (\mathbf{Ux})^{\dagger}\mathbf{Ux} = \mathbf{x}^{\dagger}\mathbf{U}^{\dagger}\mathbf{Ux} = \mathbf{x}^{\dagger}\mathbf{Ix} = \mathbf{x}^{\dagger}\mathbf{x} = \|\mathbf{x}\|_{\mathbf{2}} \tag{1.10}$$

Since all operations performed by Quantum Computers are represented by unitary matrices, there is potential that they can be applied to mitigate overfitting in Neural Networks. Now, the linear layers will not be restricted to be unitary in this instance (although that is a potential source of future research), but the substitution of at least a partial amount of the operations of the network with unitary operations will still be sought out for the aim of reducing the variance of the conditional distribution that the model learns.

1.6 Quantum Circuits

The Quantum Circuit model is the most commonly used framework for Quantum Computing, the other popular framework being the Quantum Annealer model. The computational medium in Quantum Computing is the qubit, which is similar to the digital bit except

it exists in a superposition of the excited $|1\rangle$ and ground $|0\rangle$ state. While during computation the qubit can exist in a superposition of these states, during measurement the qubit must collapse to one of the two probabilistically. [9] The superposition of these states together are represented by the state vector $|\psi\rangle$ can be written as a linear combination of these two states, represented by the vectors:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \alpha \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \beta \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \tag{1.11}$$

$|1\rangle$ and $|0\rangle$ form a basis for the vector space that $|\psi\rangle$ is a member of. The values of α and β are known as the amplitudes of the two basis states $|1\rangle$ and $|0\rangle$, respectively, which can be both positive and negative values. The amplitudes are modified by the circuits operations to increase the probability that one of the states will be the one the state vector collapses to at the time of measurement. The Born Rule, here shown for just one qubit, then provides the probability p that the qubit q will be measured in each state:

$$p(q = 0) =\mid \alpha \mid^2, \quad p(q = 1) =\mid \beta \mid^2 \tag{1.12}$$

And since the qubit must collapse to one of the two states then it must hold that:

$$\mid \alpha \mid^2 + \mid \beta \mid^2 = 1 \tag{1.13}$$

The salient point to this result is that at all times the state vector must be of unit length since:

$$\||\psi\rangle\|_2 = \sqrt{\mid \alpha \mid^2 + \mid \beta \mid^2} = \sqrt{1} = 1 \tag{1.14}$$

Meaning that at any point the state vector must be normalized to have a length of 1. For more than one qubit, the length of the state vector is 2^n for n qubits, since that is the total number of digital values that could be collapsed to with n qubits. The Quantum Circuit model represents computation as a series of gate operations, analogous to digital logic gates, represented by unitary matrices performed on the state vector. [9] The goal of a quantum

circuit it to adjust the amplitudes enumerated in the state vector in order to increase the probability of having a desired state be measured at the end of the computation. The act of measuring in quantum mechanics causes the state vector to collapse to one of its basis states, and this process can be represented by a matrix M_{i} called the "Measurement operator" for an outcome i. The probability that a qubit in a state $|\psi\rangle$ will be measured in a state i is given by:

$$p(q = \mathrm{i}) = \langle\psi|\mathbf{M_i}^\dagger\mathbf{M_i}|\psi\rangle \tag{1.15}$$

where $\langle\psi|$ is the conjugate transpose of $|\psi\rangle$. Since the state vector has to collapse to one of the n of the outcomes in a quantum measurement process can result, this means:

$$\sum_{\mathrm{i}}^{n} p(q = \mathrm{i}) = \sum_{\mathrm{i}}^{n}\langle\psi|\mathbf{M_i}^\dagger\mathbf{M_i}|\psi\rangle = \langle\psi|(\sum_{\mathbf{i}}^{\mathbf{n}}\mathbf{M_i}^\dagger\mathbf{M_i})|\psi\rangle = \mathit{1} \tag{1.16}$$

And since the length of the state vector must be 1 this means:

$$\sum_{\mathrm{i}}^{n}\mathbf{M_i}^\dagger\mathbf{M_i} = \mathbf{I} \tag{1.17}$$

This is the formulation for measurement in a quantum mechanical system in general. However, measurement in quantum computing is a simplified version of the generalized measurement process in quantum mechanics that can be described as a projective measurement, which is usually made by using a Hermitian, an operator equal to it's own conjugate transpose, quantum logic gate as the measurement operator. [9] These measurements can be made, or done "elementwise" as this paper will do for each qubit in the circuit, each having its own state vector. The projective measurement is described by an "Observable", which is a Hermitian matrix $\mathbf{O}$. Hermitian matrices, which are just "complex symmetric" matrices, therefore have a spectral decomposition:

$$\mathbf{O} = \sum_{\lambda}^{\mathbf{n}}\lambda\mathbf{P}_\lambda \tag{1.18}$$

where $\mathbf{P}_\lambda$ is a projection matrix that projects onto the eigenvector of $\mathbf{O}$ with an eigenvalue of λ, which is one of the n outcomes that can be measured once the observable is applied.

Therefore the probability for the system $|\psi\rangle$ to be measured to have an outcome λ is given by:

$$p(q = \lambda) = \langle\psi|\mathbf{P}_\lambda|\psi\rangle \tag{1.19}$$

In this paper the observable used will be the Pauli Z-gate, allowing the outcome of a measurement to be the two eigenvalues of that gate, which happen to be the eigenvalues for eigenvectors of the standard computational basis vectors $|1\rangle$ and $|0\rangle$. Since each state being measured is a probabilistic process, the expectation value of the state vector at measurement time with observable $\mathbf{O}$ is given by:

$$\mathbb{E}(\mathbf{O}) = \sum_{\lambda}^{n} \lambda p(q = \lambda) \tag{1.20}$$

And so for the projective measurements used in quantum computing, such as those for the observable provided by the Pauli-Z gate $\mathbf{Z}$:

$$\mathbb{E}(\mathbf{Z}) = \sum_{\lambda}^{\mathbf{n}} \lambda p(\mathbf{q} = \lambda) = \sum_{\lambda}^{\mathbf{n}} \lambda\langle\psi|\mathbf{P}_\lambda|\psi\rangle = \langle\psi|(\sum_{\lambda}^{\mathbf{n}} \lambda\mathbf{P}_\lambda)|\psi\rangle = \langle\psi|\mathbf{Z}|\psi\rangle \tag{1.21}$$

This is the actual operation from quantum measurement theory we will be using in this paper (and the reason we have discussed this theory at this point). Another important concept in quantum computing theory is the application of entanglement, the quantum mechanical phenomenon where the state vector of one system is dependent on the state vector of another if they have interacted previously, even if they are not directly in the present. In quantum computing entanglement is intentionally introduced between qubits in order to allow dependent behavior to occur between them in a way digital bits do not allow for. In quantum circuits entanglement is done using the "Conditional NOT" or (CNOT) gates where the state vector of one. However, if the state vector of the control qubit is a superposition of both the excited and ground states, then the two qubits become entangled, where the inversion of the target qubit, and therefore all of its subsequent states, is dependent on what the control qubits state ends up being at the time of measurement.

An important consequence of each gate being a unitary operator is that, unlike digital logic gates, every operation is completely reversible, as for each unitary operation a gate may express $\mathbf{U}$, it's adjoint $\mathbf{U}^\dagger$ is also a unitary operation that can be expressed as a quantum gate. So to undo and fully recover the information present in the state vector before any operation $\mathbf{U}^\dagger$ one must simply apply an operation expressing its adjoint $\mathbf{U}$ assuming measurement hasn't been made at this point.

1.7 Quantum Machine Learning

There has been previous research interest in leveraging the unitary operation of all quantum circuits to make Quantum Neural Networks that are resistant to overfitting. However, previous publications would use tunable gate parameters analogous to the tunable linear layers in classical neural networks, whereas this paper will leverage quantum circuits as non-tunable activation functions and keep the linear layers between those activation functions classical. These tunable gate models Mitarai *et al.* and Chen *et al.* do however provide an effective way of encoding digital information into quantum circuits by way of parameterized gate parameters. For example, successive parameterized gates can encode digital information into qubits by adjusting the amplitudes of a qubit as a function of the parameterized gate arguments. Therefore, a similar method of encoding digital values into quantum circuits is used in this paper, but without the gates downstream in the circuit of these encoding that are optimized during the learning algorithm.

It is also notable that there has even been research on regularization of Quantum Machine Learning models. For example, Kobayashi *et al.* discuss using turning off CNOT gate connections with some probability, similar to classical neural network dropout layers, in order to combat overfitting in their Quantum Neural Network models. However, this paper is distinguished from that one as the aim of this paper is to use quantum circuits for regularization, rather than use regularization on quantum circuits.

2. QUANTUM ACTIVATION FUNCTION NETWORKS

There is a problem many quantum computing practitioners will face, which is that the actual execution of quantum computing software will likely have discrepancies from what they expect from their theoretical calculation. This is because of the imperfections of currently available quantum computing hardware that result in noisy qubit states as well as difficulties with reading out qubit states at measurement time. More noise is introduced at each gate operation on a qubit, and so the more computation that takes place in quantum computing, the more discrepancy from theoretical values is introduced. This is known as decoherence and is a significant source of error in presently available quantum circuits. There are efforts in quantum computing research to introduce error-correcting methods analogous to those in digital computing, but these require significantly more qubits than many computers are currently able to maintain in superpositional states. This has lead researchers to dub the current stage of quantum computing research as the Noisy-Intermediate Scale Quantum (NISQ) era. [12]

Furthermore, access to quantum computing resources is scarce and costly, limiting access to use for smaller organizations or independent researchers. This is particularly true for Quantum Machine Learning researchers, as the process of training a neural network, classical or quantum, can involve hundreds, if not thousands, of network evaluations. This means that for a quantum network, potentially thousands of circuit trials must be run to train a single model. For commercially available quantum computers, circuits are able to be run by submitting code over the internet to the quantum computing provider, who will then run the circuit locally on a quantum computer and return the results over the internet. This also introduce network latencies that can make training a neural network a long process. So there are both limitations in harware and logistics to engineering a purely quantum machine learning model.

One potential remedy is to only have a part of the neural network run on a quantum circuit, essentially creating a neural network that is a hybrid of both quantum and classical neural networks. This would mean that a sizeable portion of the network would still be executed by a digital computer while still seeking to leverage the overfitting resistance of

quantum neural networks. Therefore there are essentially three options to achieve a hybrid network. The first option would be to simply replace some of the layers of a classical network with quantum circuit layers in a similar configuration that Mitarai *et al.* had done. This is an active area of research with Arthur and Date showing some promising results in using a mix of quantum and classical layers to do binary classification. Another option is to replace the linear layers of the neural network with tunable quantum circuits and keep the activation functions as digital computer operations. However, this would not achieve a significant speed up since the activation function is only a $O(n)$ operation, since it applies a constant time operation on each of the n elements of a feature vector, whereas the linear layer operation is an $O(n^2)$ matrix vector product. So it would be more effective from a speed-up perspective to instead keep the classical linear layers executed on a digital computer and to instead use a quantum circuit as the neural networks activation function.

The focus of this paper is this third option, which is to keep the linear layers of the neural network as classical matrices and then to replace the non-linear activation function with a quantum circuit. This circuit will then be compared in its ability to regularize a neural network vs. L2-Regularized networks using the ReLU activation function. It is important to note that the size of the networks will be smaller than most neural networks to accommodate the limitations quantum circuit trials will bring over 1000 training steps, since we want both the ReLU network and the Quantum Activation Function (QAF) network to have the same dimensions and to only differ in their activation functions.

2.1 Quantum Activation Function Circuit

In order to do an "apples-to-apples" comparison between the ReLU network and QAF network, the same depth and width will be used for both, meaning that the activation functions must map out of and into the same output and input dimensions. As stated before this will be a single hidden layer network in both cases. So the layout is a 4-to-4 linear layer, followed by a 4-to-4 activation function of either ReLU or QAF, followed by another 4-to-1 linear layer to complete the network. There are three sections of the quantum circuit: the encoding section that takes in the output of the classical neural network layer, the entangling

section that ensures the qubits are put into a maximally entangled state, and the Output Measurement Section that measures the qubit values and passes them onto the next part of the larger Neural Network. While the classical layers of the networks are tuned by the optimization algorithm, and there are no tuned parameters for quantum circuit.

Encoding Section: The encoding section begins with four ground state qubits, meaning all four start the activation function process with a value of $|0\rangle$. The qubits first each pass through a *Hadamard gate* and then pass through two rotation gates in succession, the *RY gate* and the *RZ gate*. The Hadamard gate puts the ground state into a superposition with equal amplitudes for the two basis states. The RY gate performs a rotation around the Y-axis of the "Bloch Sphere", which is a visual representation of a single qubit's superpositional state. The RZ gate then performs a rotation around the Z-axis of the Bloch Sphere. [9] What this means is that the amplitudes of the basis states, which correspond to the probabilities of each state being measured, are altered as a function of the input values. This is where the nonlinearity of the QAF is introduced, which is a requirement for the Neural Network to fit piecewise an arbitrary function. The operations of the gates are matrix-vector multiplication performed on $|0\rangle$ with matrices of:

$$\mathbf{H} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \tag{2.1}$$

$$\mathbf{R}_y(\alpha) = \begin{bmatrix} cos(\frac{\alpha}{2}) & -sin(\frac{\alpha}{2}) \\ sin(\frac{\alpha}{2}) & cos(\frac{\alpha}{2}) \end{bmatrix} \tag{2.2}$$

$$\mathbf{R}_z(\beta) = \begin{bmatrix} \mathrm{e}^{-\mathrm{i}\frac{\beta}{2}} & 0 \\ 0 & \mathrm{e}^{\mathrm{i}\frac{\beta}{2}} \end{bmatrix} \tag{2.3}$$

The input parameters of the two rotation gates are functions of the output of the previous classical neural network layer. Since the previous classical layer supplies a length-four vector called $\mathbf{x}$, corresponding to the four qubits in the circuit, to the quantum activation function, the values are encoded as follows into the rotation gates:

$$\alpha = tan^{-1}(x_{\mathrm{i}}), \quad \beta = tan^{-1}(x_{\mathrm{i}}^2) \;\; \forall \; x_{\mathrm{i}} \in \mathbf{x} \tag{2.4}$$

The use of the inverse tangent function in the encoding introduces non-linearity into the circuit, while the squaring the argument of the RZ gate before applying the inverse tangent function also allows higher dimensional values to be expressed in the encoding. Effectively, these two successive gates allow potentially any vector on the Bloch Sphere to be encoded. This encoding requirement could be relaxed however, and this would likely introduce more bias into the network as the range of outputs the circuit could return as a function of the inputs would be restricted. This is perhaps one avenue for modifying the amount of regularization the QAF provides to the network. However, the focus of this paper is mainly to investigate the validity of using QAFs in practice, and so that study will be left to future research. The encoding section expresses the following:

Algorithm 1 Encoding Section Procedure

Input: x ▷ Takes in length 4 digital vector
Output: q ▷ Return array of 4 qubits
$q \leftarrow$ new $qubit[4]$ ▷ Declare four qubit array
for i $= 0 : 3$ **do**
  $q[\mathrm{i}] \leftarrow |0\rangle$ ▷ Initialize each qubit to ground state

  $\alpha \leftarrow tan^{-1}(x[\mathrm{i}])$
  $\beta \leftarrow tan^{-1}(x[\mathrm{i}]^2)$
  $H(q[\mathrm{i}])$ ▷ Apply Hadamard
  $Ry(q[\mathrm{i}], \alpha)$ ▷ Apply R_y with gate parameter α
  $Rz(q[\mathrm{i}], \beta)$ ▷ Apply R_z with gate parameter β

end for
return q

Entangling Section: The entangling section uses CNOT gates to entangle the states of the four qubits encoding our data. Earlier it was mentioned how CNOT gates allow two qubits to become dependent on one another, and many quantum computing algorithms use this interdependence to achieve additional computation without needing communication to occur constantly between two values like in digital computation. CNOT gates apply an

inverting operation where the amplitudes of the two basis states are swapped for the target qubit, as long as the control qubit is in the excited state, meaning $|1\rangle$. A two-qubit state vector has a length of four, and so the CNOT gate applies the following four-by-four matrix operation for :

$$\mathbf{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{2.5}$$

This entangling allows the qubits' states to become dependant on one another, which can allow more complicated functions to be expressed by the circuit than if there was no interaction between the qubits. It is also notable that this section seeks to maximally entangle the qubits, meaning each qubit is entangled with every other qubit. There is perhaps potential for others configurations of QAFs with less entanglement, and future research could be conducted on how this may affect the amount of regularization the QAF applies to the network. This gate in the following pseudocode then takes two arguments, the first being the control qubit and the second being the target qubit. In order to entangle each qubit with every other qubit, CNOT operations are performed on qubit pairs of increasing "strides" apart from one another, which generalize the procedure for any number of qubits. This resulting entanglement is achieved with:

Algorithm 2 Entanglement Procedure

```
Input: q                                   ▷ Takes in the current qubit states
Output: q
for stride = 1 : 3 do
    for i = 0 : 3 do
        CNOT(q[i],q[(i+stride)%4])         ▷ Entangle the two qubits
    end for
end for
return q
```

One thing to note is this procedure is generalized so that any number of qubits can be used, but for the trials conducted later only four qubit circuits will be used.

Output Measurement Section: The final section of the circuit returns the expectation value of each qubit in the computational basis individually. The expectation value of a qubit q_{i} of state ψ_{i} measured in the computational basis is given by equation Eqn. 1.20, and since this is computed elementwise for the output vector $\mathbf{r}$, the elements of this vector are:

$$r_{\mathrm{i}} = \langle \psi_{\mathrm{i}} | \mathbf{Z} | \psi_{\mathrm{i}} \rangle \tag{2.6}$$

where:

$$\mathbf{Z} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{2.7}$$

These four expectation values are then returned from the function. This is different than most expectation value measurements in quantum computing, and is done in an effort to still keep the different features of the feature vector distinct from one another, rather than collectively taking an expectation value that will result in a single number. The measurement output process is then:

Algorithm 3 Output Measurement Procedure

Input: q ▷ Takes in the current qubit states
Output: r ▷ Returns a length-4 digital vector
$r \leftarrow$ new $float[4]$ ▷ Declare array of two floats
for i $= 0 : 3$ **do**
    r[i] $\leftarrow exp_Z(q[\mathrm{i}])$ ▷ Perform $\langle \psi_{\mathrm{i}} | \mathbf{Z} | \psi_{\mathrm{i}} \rangle$ and store in r[i]

end for
return r

These three procedural stages together can also be visualized with a pictorial depiction, which is why quantum circuit layouts are often called "quantum algorithms", as they describe a process. This circuit is what is applied at the hidden layer of a single hidden layer network such as the one depicted in Figure 1.1.

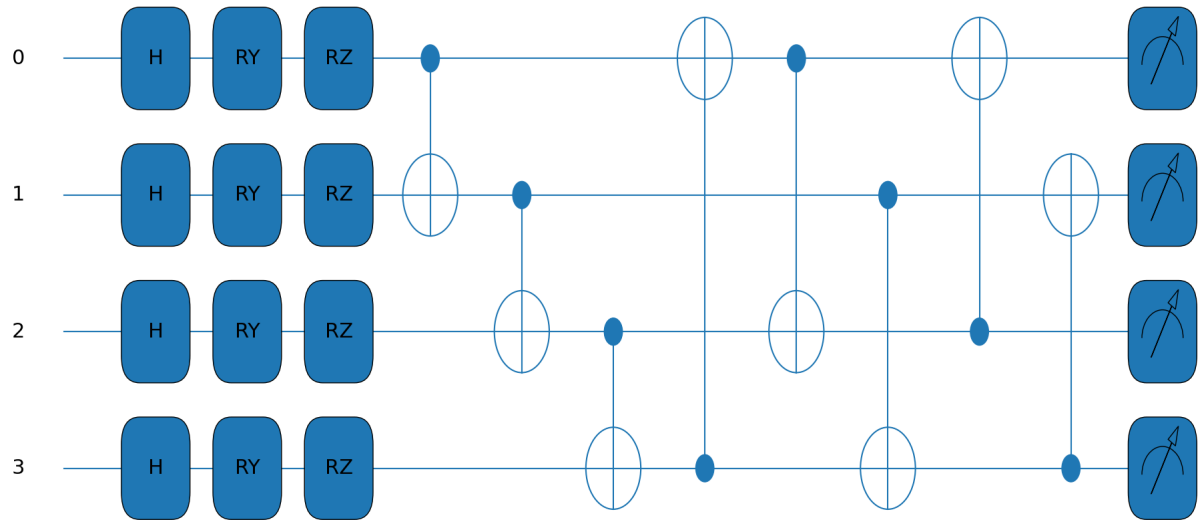

Figure 2.1. Visualization of Quantum Activation Function Circuit

In order to optimize the linear layers of the network, the backpropagation algorithm is performed to obtain the gradient of the network. [4] This requires knowing the gradient of both the layer weight matrices, as well as the effect that the quantum circuit activation function has on the gradient of the network, with respect to their inputs. The quantum circuit's gradient is then determined with the parameter shift rule, a finite difference method that finds the exact gradient by applying a change of $\frac{\pi}{2}$, both positive and negative, to each of the input gates, in this case the RY and RZ gates for each of the four qubits, and evaluating the output at these shifted inputs. While this parameter shift rule is often applied to update the gate parameters in the quantum circuit of a fully quantum neural network, here it is just used to find the gradient of the activation function, which is then supplied to the backpropagation algorithm to find the gradient of the entire neural network. Before the expectation value was set to be:

$$\langle \psi_{\mathrm{i}} | \mathbf{Z} | \psi_{\mathrm{i}} \rangle \tag{2.8}$$

Allowing the state at the time of measurement to be the result of performing a unitary operation parameterized by the components of the input vector x, and calling such an operator $U(x_{\mathrm{i}})$, allows the state to be written as:

$$|\psi_{\mathrm{i}}\rangle = \mathbf{U}(x_{\mathrm{i}})|0\rangle \tag{2.9}$$

where x_{i} is the i^{th} component of the input vector x. The expectation value can then be written as:

$$\langle 0|\mathbf{U}(x_{\mathrm{i}})^{\dagger}\ \mathbf{Z}\ U(x_{\mathrm{i}})|0\rangle \tag{2.10}$$

Using the parameter shift rule, the gradient expression for the function that takes the expectation value of the circuit output is then found by evaluating this expectation value with input x_{i} shifted up by $\frac{\pi}{2}$ and subtracting from it x_{i} shifted down by $\frac{\pi}{2}$ and diving both terms in half:

$$\nabla\langle\psi_{\mathrm{i}}|\mathbf{Z}|\psi_{\mathrm{i}}\rangle = \frac{1}{2}(\langle 0|\mathbf{U}(x_{\mathrm{i}} + \frac{\boldsymbol{\pi}}{2})^{\dagger}\ \mathbf{Z}\ \mathbf{U}(x_{\mathrm{i}} + \frac{\boldsymbol{\pi}}{2})|0\rangle - \langle 0|\mathbf{U}(x_{\mathrm{i}} - \frac{\boldsymbol{\pi}}{2})^{\dagger}\ \mathbf{Z}\ \mathbf{U}(x_{\mathrm{i}} - \frac{\boldsymbol{\pi}}{2})|0\rangle) \tag{2.11}$$

$$\Rightarrow \nabla\langle\psi_{\mathrm{i}}|\mathbf{Z}|\psi_{\mathrm{i}}\rangle = \frac{1}{2}\langle 0|(\mathbf{U}(x_{\mathrm{i}} + \frac{\boldsymbol{\pi}}{2})^{\dagger}\ \mathbf{Z}\ \mathbf{U}(x_{\mathrm{i}} + \frac{\boldsymbol{\pi}}{2}) - \mathbf{U}(x_{\mathrm{i}} - \frac{\boldsymbol{\pi}}{2})^{\dagger}\ \mathbf{Z}\ \mathbf{U}(x_{\mathrm{i}} - \frac{\boldsymbol{\pi}}{2}))|0\rangle \tag{2.12}$$

This gradient value is then passed to the backpropagation algorithm that uses reverse-mode automatic differentiation to find the gradient of the entire neural network. The optimizer then uses the first and second moment of the network's gradient to update the weights of the classical layers. One important thing to note is that the term is the middle in practice is usually just computed on digital computers by performing the matrix operations to provide backpropagation with the gradient of the QAF.

Quantum Activation Functions seek to limit the representational capacity of a hypothesis class, as opposed to the effective capacity like L2-Regularization does. The reason that the QAF regularizes the single hidden layer network model is because it limits the range of functions that can be included in the set of the hypothesis class, forcing the network to express a simpler class of functions. This may sound counterintuitive, as the above sections on

quantum computing circuits are certainly more complicated than commonly used activation functions such as ReLU or Sigmoid. However, while the math internal to the circuit is more complicated and requires some understanding of quantum mechanics to implement, when the network is viewed as an abstracted black box, the actual class of functions that the model is able to express is smaller than networks using ReLU or Sigmoid as it's activation function, since the output range of the QAF is given by:

$$QAF(\mathbf{x}) = \begin{bmatrix} [-1, 1] \\ [-1, 1] \\ [-1, 1] \\ [-1, 1] \end{bmatrix} \tag{2.13}$$

This is because the spectral decomposition of the Z-gate is:

$$\mathbf{Z} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = 1 \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} - 1 \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} = 1|0\rangle\langle 0| - 1|1\rangle\langle 1| \tag{2.14}$$

This shows that the eigenvalues of the Z-gate are 1 and -1, meaning that the expectation value will always land somewhere between these two values, since they are the two outcomes that can possibly be measured. This also shows why the Z-gate projective measurement is known as the computational basis measurement, as the eigenvectors its projects along are the standard quantum computing vectors of $|0\rangle$ and $|1\rangle$. It's important to note that this range can be stated since the expectation value is applied elementwise, rather than the usual collective operation that takes the expectation of the entire state vector at the end of the circuit operation.

The QAF then is limiting the hypothesis class of the network up until this point by restricting the elementwise operations to be $QAF(x) = [-1, 1]$, because even though there are an infinite number of values in $[-1, 1]$, the actual number of functions expressible is smaller compared to a ReLU's element wise operation of $ReLU(x) = max(0, x)$. This is distinct to the way architecture based methods, such as dropout layers, limit the representational capacity of a hypothesis class since the restriction is applied on the output range rather than on feature vector coefficients. So conceptually, the QAF network will apply less

regularization than dropout layer regularization for some turn-off probability values. This will later be shown to also be the case with L2-Regularization, where the QAF network has more regularization for some regularization parameter values, and less for others.

This is similar to methods in vector normalization that prevent certain features from being "overrepresented" in the vector as the network computation progresses. [14] For example, in classification there are often "more important" features than others, such a the size of a possibly cancerous growth vs. it's color, as a deciding feature to classify its malignancy. This example is used as, with most statistical learning theory, the task of classification gives a good theoretical framework to conceptually state what is happening internally with the model. However, what can happen is that the network weights these features significantly more that the "less important" features get overwhelmed, limiting the nuance that the classifier network can express. The QAF therefore allows relative feature importance to still be made, while not drowning out the less important features. This is similar to normalization layers that scale the magnitude of a layers vector to be of unit length. Although it is important to note that the QAF only somewhat "normalizes" the vector as the output is not bounded to have a 2-norm of 1. This is because the QAF can output a vector as large as:

$$\|QAF_{max}(\mathbf{x})\|_2 = \sqrt{(\pm 1)^2 + (\pm 1)^2 + (\pm 1)^2 + (\pm 1)^2} = 2 \tag{2.15}$$

This means there is still a larger hypothesis class expressed by a QAF, meaning more variance, than if the feature vector was just to be normalized at this point, while still being more regularized than a ReLU network of equal dimensions. The entangling procedure also provides additional expressibility and interaction between the different elements of the feature vector, which is the same motivation Kobayashi *et al.* have for including such a sequence. [8] This conceptually can be thought of as the different features interacting after their relative positions on the Bloch Spehere have been encoded. Once these encoded features become entangled they are able to become interdependent on one another which allows them to communicate their importance to the mapping relative to one another, the measure of which is expressed when each qubit is measured at the end of the circuit.

The QAF, as with all quantum circuits, also measures the output of each vector element as a stochastic process, which is the expectation value with probabilities of each outcome being determined by the Born Rule. This may serve as an additional regularization source akin to dropout layers since they both have the chance to "shut off" a feature with some probability. The dropout layer, with some passed value for the probability of shutting a feature off, sets some weight parameters to 0, while the QAF has the chance to read out any expectation value in $[-1, 1]$, and so can also set features to 0, or just close enough to 0. The difference with the QAF is that the probabilities of features being "shut off" are still reflective of the encoded and not just a constant parameter passed as in dropout layers. This may provide a sort of "informed dropout" that may be used to more effectively shut off the features that would cause the model to overfit rather than the trial-and-error approach of dropout layers that can converge to a regularized fit over many training steps. The other distinction is that the while the dropout layer will turn off elements of the linear layer matrix, possibly limiting the operators that can be expressed by that layer, the QAF's regularization only takes place in the activation function between the linear layers and so does not limit the range of mappings that the linear layers can take on.

The QAF's placement within the larger neural network then allows over multiple layers then filtering of features that are most important for the mapping without allowing these more important features from "blowing up" since the range of each element of the QAF is in $[-1, 1]$. The linear layers upstream of the QAF are optimized to give the more important features more weight, in the same way as in classic dimensionality reduction problems in machine learning, but with the addition that the QAF prevents this optimization from being too extreme in one direction in order to allow the downstream layers from still being able to recover information about the features that the upstream layers mapped to be "unimportant". This then mitigates an issue where the upstream layers tune a weight close to 0, and then downstream layers are unable to undo the effects of that tuning. For the trial tasks in the next section, the output of the QAF is then applied to a final linear layer that is just a row vector, to allow any final scaling and linear combinations to be performed in order to express values on the same order the target data would have. The operation of the QAF can then be summarized as performing nonlinear, semi-normalizing, filtering.

3. PERFORMANCE AND COMPARISON TO RELU NETWORKS

Now the performance of the QAF will be demonstrated in three learning tasks, classification, regression of a polynomial target function, and regression of a multicollinear data set, against a ReLU network using L2-Regularization. The three networks are implemented in the PyTorch library for machine learning, while the Quantum Circuit is interfaced using the Pennylane library for quantum computing. Then a measure of how much regularization the QAF is providing will be determined by comparing performance against multiple results of L2-Regularized ReLU networks with different regularization parameter values, which is a method Kobayashi *et al.* mention as being a good benchmark for how "regularizing" a regularization method is. This is done by showing the generalization gap for the QAF network compared to various L2-regularized ReLU networks. As stated in the section on Regularization, the first step when designing a new machine learning algorithm is to show that it does not have too high of bias as to not be able to learn a data set, and so the first step is to demonstrate that a model's loss is decreasing during the training stage. Each task is executed over 1000 training steps where the entire training data set is processed, the corresponding loss is computed, and then the neural network weights are updated to minimize this loss. At each training step, the test set is also processed to demonstrate how the model performs against unseen data. The reason this data is unseen is because it is not used to update the network, only the loss for the training data set is. This allows the generalization error of the network to be demonstrated as more time is allowed for the networks to train, and also gives an indication how much overfitting and underfitting of the data the network will do. Since a network that heavily overfits will do so more and more as training progresses, it is then a good measure of regularization to demonstrate how well the network is generalizing as training progresses as well. [4] Finally, all the networks are trained with the Stochastic Gradient Descent optimizer in order to have as comparable optimization process outside of the difference in activation functions. Therefore, the only difference between the ReLu network and QAF network is the activation function, which will allow a more "apples-to-apples" comparison to take place.

3.1 Classification

The classification task seeks to classify possibly cancerous growths as malignant or benign as a function of four features: the mean radius, texture, perimeter, and area. Each datapoint consists of these four features and then a binary classification of 1 for malginant and 0 for benign. The data set is a canonical set of 569 real world diagnoses on breast cancer tumors regularly used to train simple machine learning models. The first goal is to compare the performance of the QAF to the ReLU without any form of regularization, which progresses in the following way over the 1000 training steps:

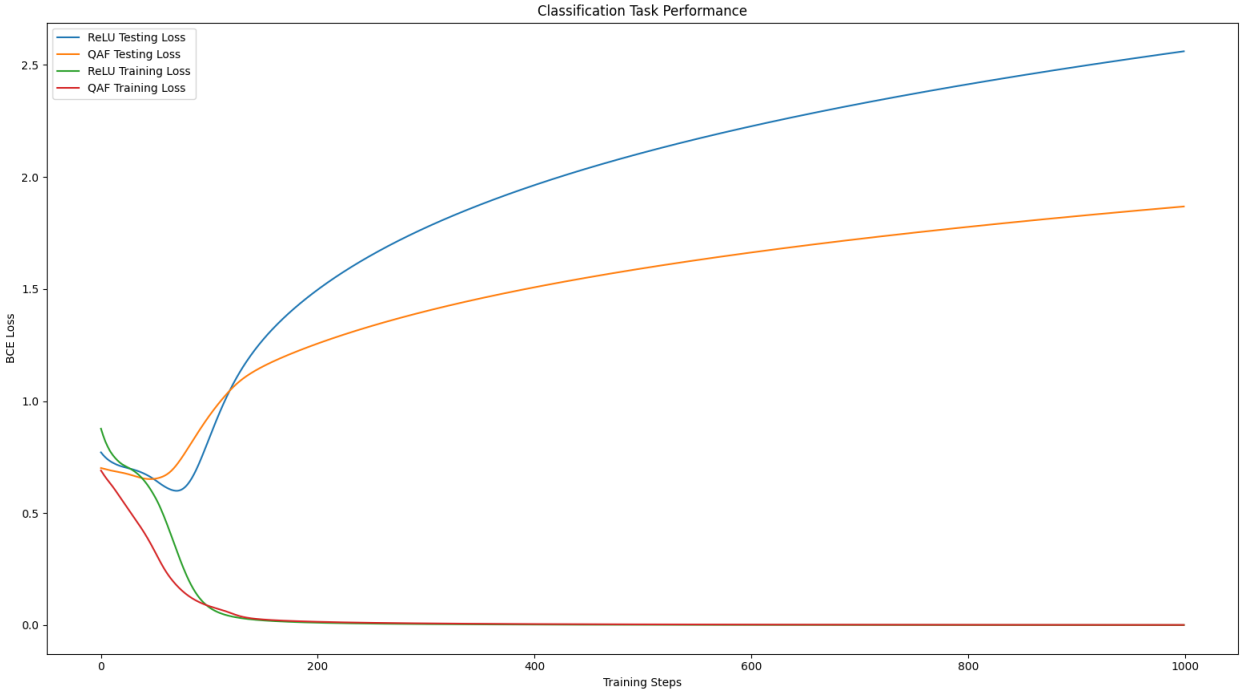

Figure 3.1. Classification Performance of QAF Network vs. Unregularized ReLU Network

The primary result from this trial is that the QAF network is capable of learning a data set as evidenced by the fact the training loss is decreasing to a comparable minima as the ReLU network. The other key result is that since the testing error is also lower for the QAF network than then ReLU network, it is providing some measure of regularization to the single hidden layer network. Now a trial is also run for an L2-Regularized network with $\lambda = .1$

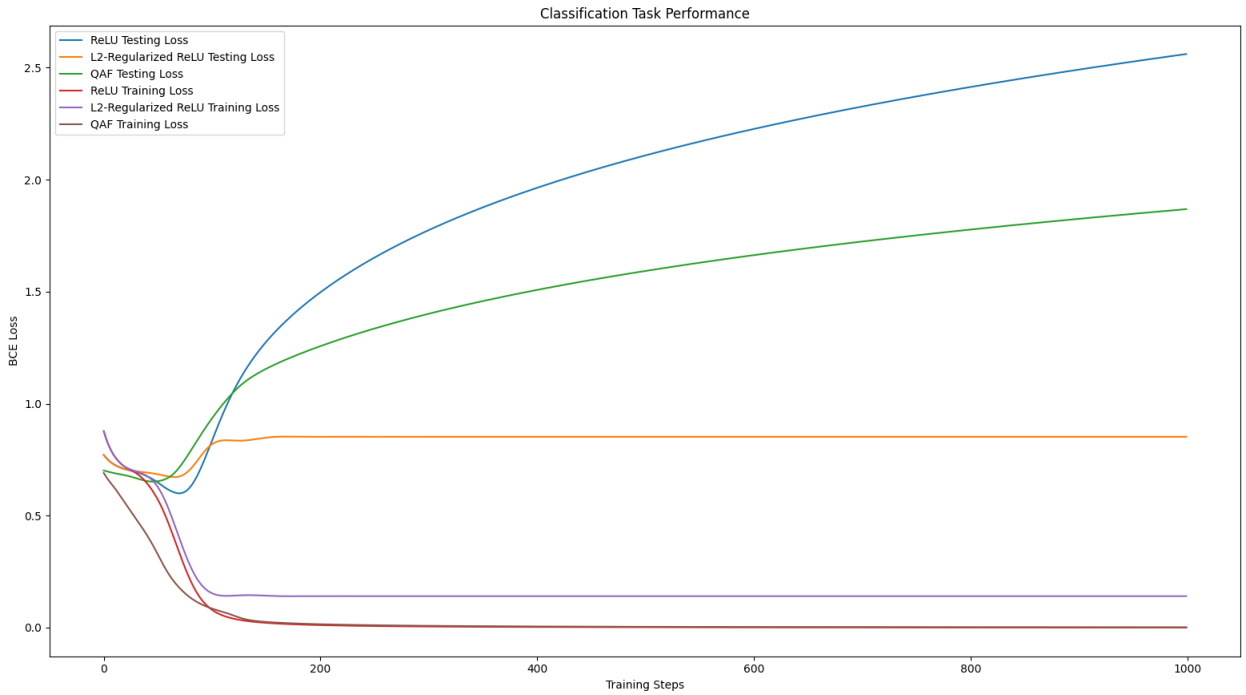

Figure 3.2. Classification performance of the three networks

One thing to note is that there is a slightly different starting point in both curves for the QAF network versus the two ReLU, despite all network parameters being initialized to the same values. This is because the ReLU and QAF networks are performing different operations since their activation functions differ. Since we now see that QAF is able to minimize it's training loss during the learning process, we can use the generalization gap plots to give the clearest measure of classification performance for the three networks.

The QAF network is shown to then generalize better than the unregularized ReLU network, but not as well as the L2-Regularized ReLU network in this classification task. Therefore, there is still some overfitting the QAF network is doing, albeit less than the unregularized ReLU network, as it tunes its parameters to minimize the training loss and to better fit the training sample data. This tells us that while there is still potentially better regularization methods for this task, the QAF is introducing more bias into the network than the ReLU network.

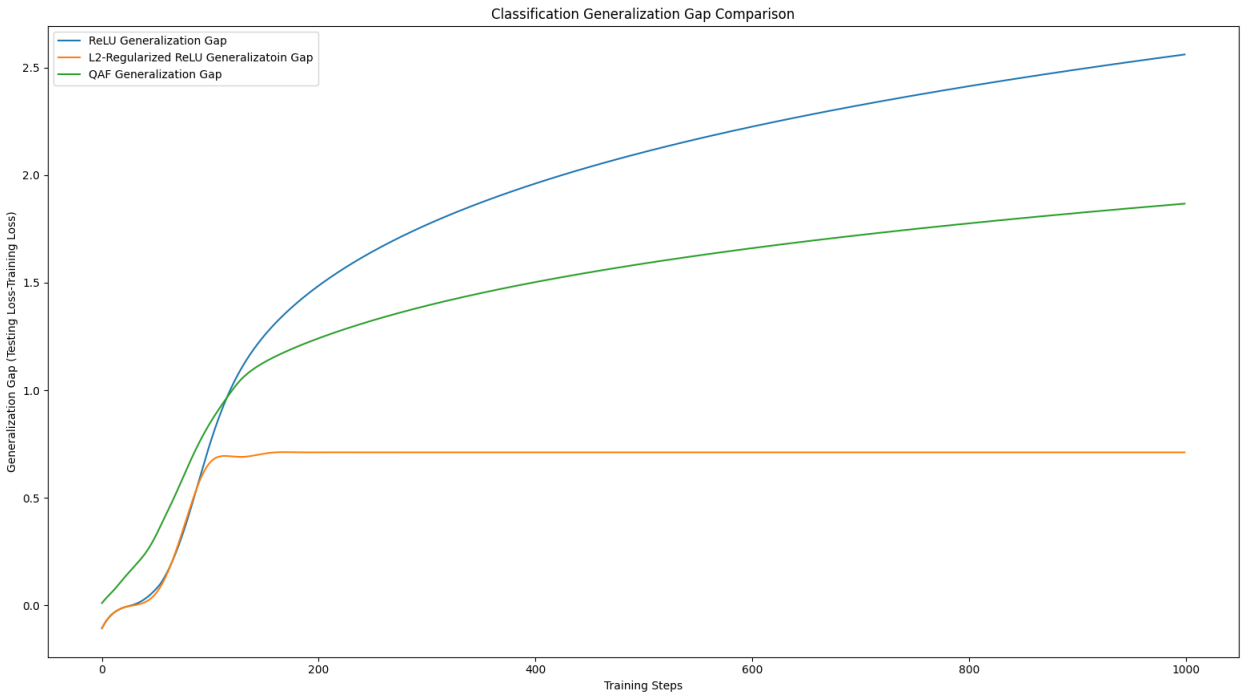

Figure 3.3. Classification generalization gaps of the three networks

3.2 Polynomial Regression

Now we turn towards two regression tasks, where the network looks to not classify a binary target value, but a continuous floating point value. For the polynomial regression task, a synthetic data set of independent variables $\mathbf{X}$ is generated from the standard normal distribution, $\mathcal{N}(0,1)$, which is a normal distribution with a mean of 0 and a variance of 1. The design $\mathbf{X}$ matrix is then a full rank matrix where each row is has four feature vectors mapped to be the features in a quartic function that is then evaluated and stored as the target data. As is standard practice for statistical regression trials, another value ϵ is added to the target vector to be noise, which in this case is a value sampled from $\mathcal{N}(0,1)$ again. This makes each i-th element of the target vector:

$$y_{\mathrm{i}} = x_{\mathrm{i},0}^4 + x_{\mathrm{i},1}^3 + x_{\mathrm{i},2}^2 + x_{\mathrm{i},3} + \epsilon, \quad \epsilon \sim \mathcal{N}(0,1) \tag{3.1}$$

where the values of $x_{i,j}$ make up the i-th row of the design matrix $\mathbf{X}$. This target data generation is motivated by testing the three networks ability to fit data that likely does not have an exact solution in the context of a single hidden layer network. Therefore, the models are likely to heavily "memorize" the training data while being far from fitting the underlying function, if regularization is not introduced. This issue is particularly troublesome for scenarios with little available sample data, and demonstrates a case where regularization methods are integral for successful network performance. As with the classification task, first it must be verified that the QAF network can learn the data set for a regression task:

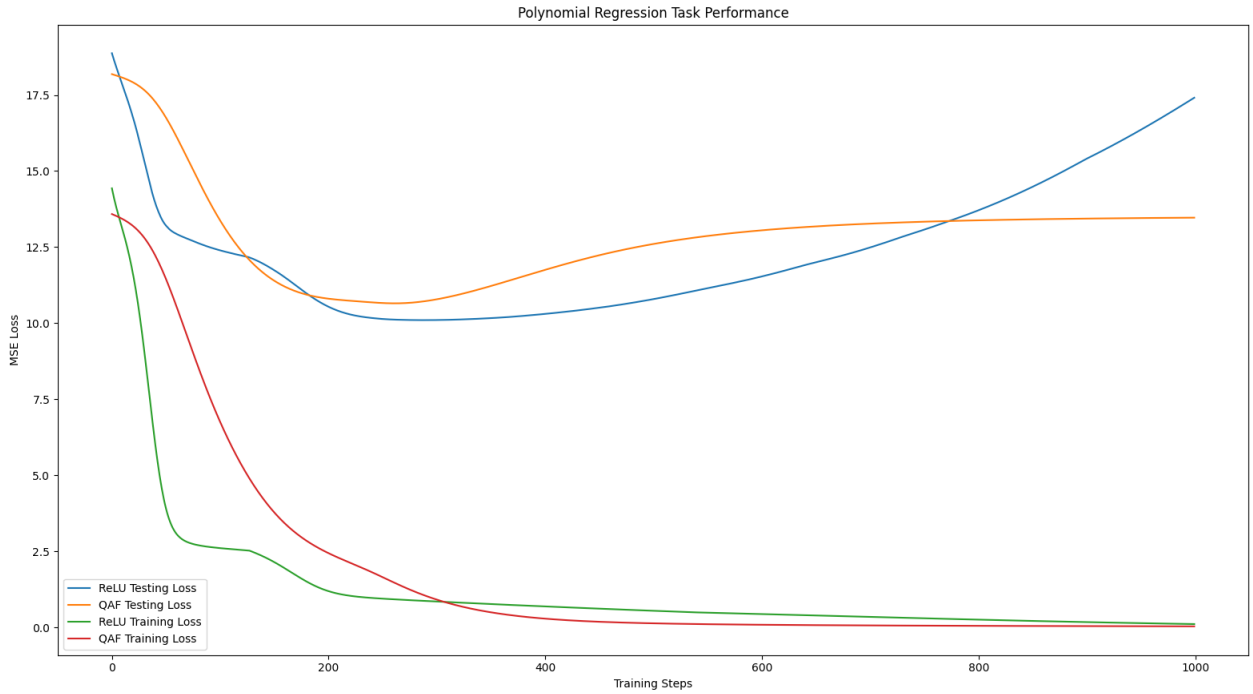

Figure 3.4. Polynomial Regression Performance of QAF Network vs. Unregularized ReLU Network

In this task both the ReLU and the QAF converge in their training loss, just as in the classification. This shows that the QAF is also able to map basic regression tasks over the course of its training steps. One thing to note is that it does take relatively more training steps for QAF to converge than the unregularized ReLU network. In many situations there may not be enough time to have a neural network go around 300 training steps, which is how long it roughly takes for the QAF networks training error to converge. The reason this is

notable is how long quantum circuit operations can take, whether they are simulated or send over the internet to access a remote quantum system. For example, for this task a training step for the ReLU network required an elapsed time on the order of $10^{-}4$ seconds where as the QAF network took on the order of 1 second. While this process for training the QAF may not be prohibitively long, that results in a roughly 15 minute training time for 1000 steps for a QAF network, and this may be prohibitively long for certain applications.

The regularization the QAF network applies in this task is useful because it has caused the testing loss to flatten out and not to increase as a function of time as the ReLU network does. This means that the ReLU network is overfitting more and more over time as the training algorithm is causing the network to memorize the training set more and more. The QAF is overfitting slightly as well however, since it begins to dip in its training curve before going back up. However, it is leveling off as the training steps increase as opposed to the ReLU curve continues growing. Therefore, the QAF also provides regularization when compared to the ReLU network for this task.

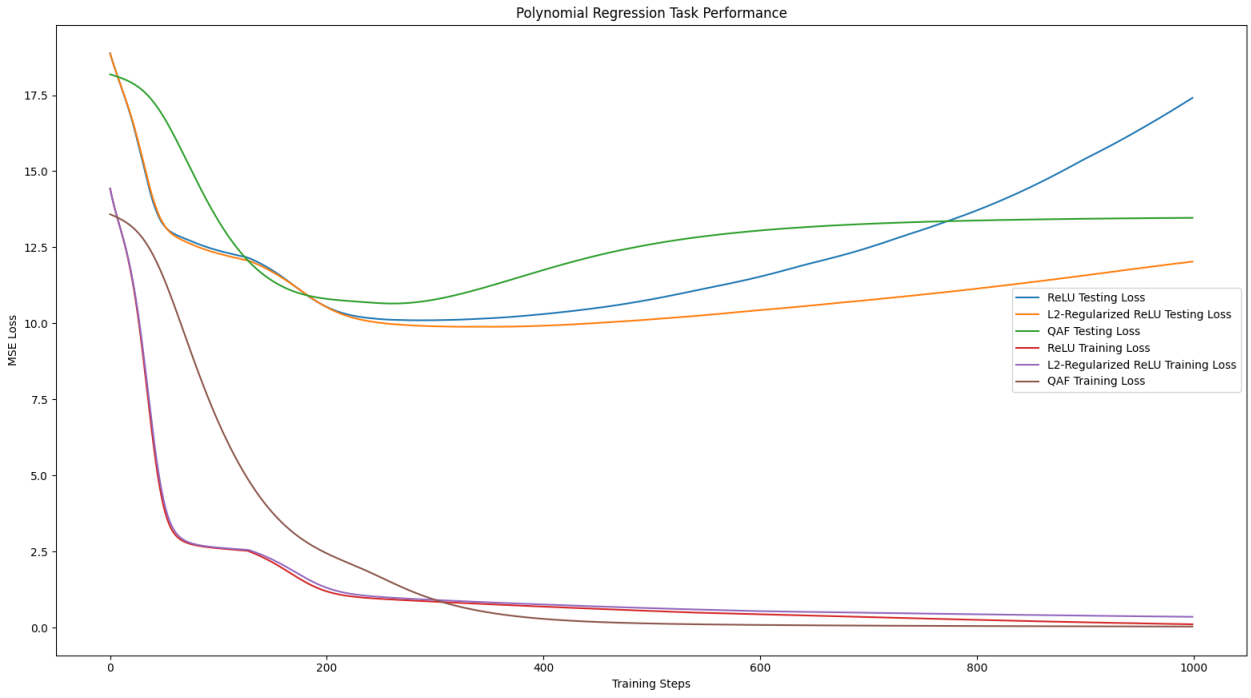

Figure 3.5. Polynomial regression performance of the three networks

Next the regularization power of the QAF is compared to the same dimension ReLU network, except with L2-Regularization applied with a $\lambda = .1$ regularization parameter, and just as before, the QAF takes relatively more epochs to converge than the regularized ReLU network as well. Since the testing loss of the L2-regularized network is lower than the QAF, we would say that again there is more regularization applied by it than the QAF. Since we now see that the QAF is able to minimize it's training loss during the learning process, we can use the generalization gap plots to give the clearest measure of performance for the three networks for mapping the data set. One important takeaway from Figure 3.5 and Figure 3.6 is that while the QAF has a higher generalization gap, the L2-regularized network is also trending up near the end of training, just as the unregularized ReLU network is. This is a sign that they may be memorizing their training set values still, it's just that the L2-Regularization makes them more resilient to overfitting.

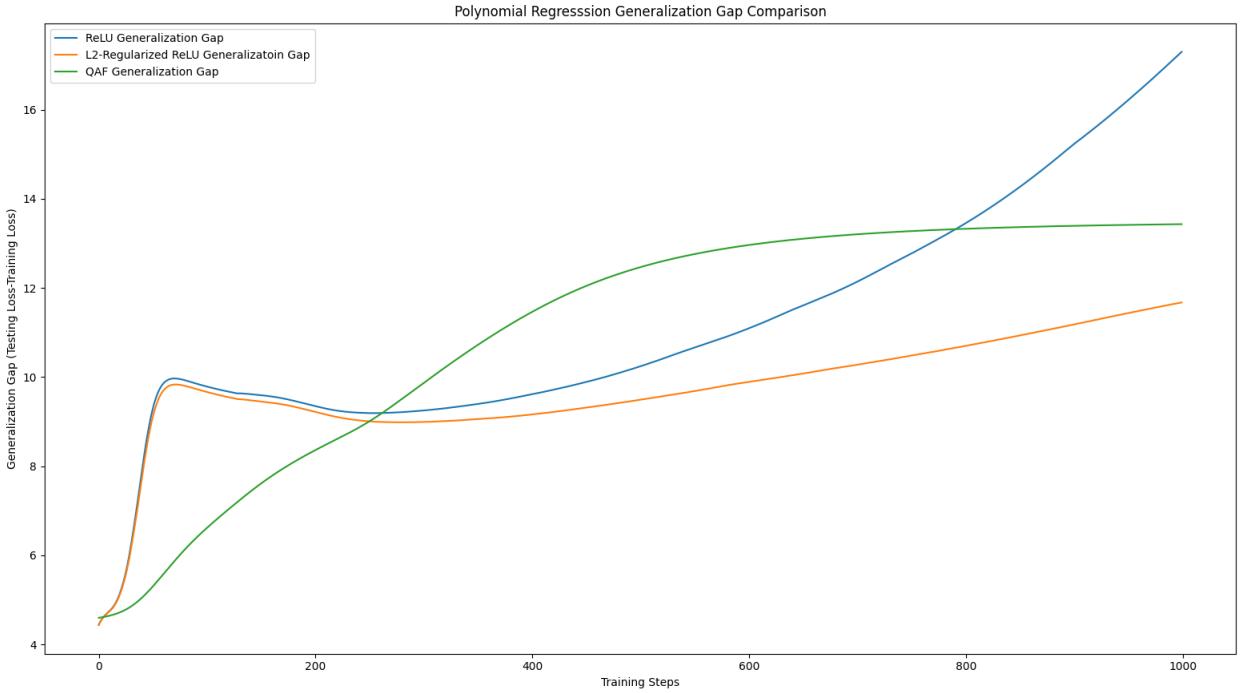

Figure 3.6. Polynomial regression generalization gaps of the three networks

On the other hand, the QAF generalization gap and testing error have leveled off, which once again shows there is a regularizing aspect to the application of QAFs that results in better performance than unregularized ReLU networks. Looking collectively at the classi-

fication and regression tasks demonstrated so far, there is evidence that the QAF provides regularization to the neural network by restricting the hypothesis class of the model in the ways previously described without needing to change the architecture of the optimized weight matrices like in dropout layers or needing to modify the objective function like with L2-Regularization. If then a scenario arises where it is impractical to use these two established regularization methods to restrict a hypothesis class, it might then be useful to apply QAFs. Although one important caveat to note is that the computational resources needed to simulate quantum circuits are significantly more expensive than those needed to implement dropout or L2-regularization for networks of equal shapes, and of course the resources needed to actually run the quantum computation are costly to access as well. However, so far it has been shown that QAFs will regularize a network, and so potential use cases will also be insightful to investigate.

3.3 Multicollinear Regression

The final learning task is one such potential real-world use case for QAF networks. This is mainly because it is an instance where QAF networks not only generalize better than unregularized ReLU networks, but also the L2-Regularized network with $\lambda = .1$. Multicollinear data sets are those that have a high degree of correlation between more than two of the independent variables, which means the design matrix columns can be written as a linear combination of one another. The issue in traditional regression methods, such as least squares, with these sorts of data sets is that their design matrices are rank deficient, and therefore the matrix in the normal equation is as well:

$$\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b} \tag{3.2}$$

Since $\mathbf{A}^T\mathbf{A}$ is rank-deficient, the system that provides the least-squares solution cannot be inverted as the solution for $\mathbf{x}$ in the normal equation does not exist. This means it is a prime candidate for alternative methods in machine learning, such as neural networks. Another thing to note is the high condition number of multicollinear data matrices, which makes solving systems with them difficult with traditional systems without using preconditioners.

The way the synthetic data set is generated then makes the design matrix to be rank deficient, which can be done by first taking a randomly generated single columns, again with each element from the standard normal distribution, and then generating the other columns as successive linear combinations. [15] Real world data sets that exhibit multicollinearity often arise in econometrics, and this is often because most economic conditions are often functions of other economic conditions with a high and fairly simple correlation.

Multicollinear data sets admit a great deal of variance to models that attempt to fit them. If the vectors that represent each of the features of a data set can be written as linear combinations of one another, that means there is redundancy in the features of the data. For example, consider the case of a datapoint with 2 features x_1 and x_2, mapping from a plane of values to a single output y where there exists the relation:

$$y(x_1, x_2) = \alpha x_1 + \beta x_2, \quad x_2 = \gamma x_1 \tag{3.3}$$

Where an optimizer or other regression method selects α and β to best fit a target value.

$$\Rightarrow y(x_1, x_2) = (\alpha + \beta\gamma)x_1 = (\frac{\alpha}{\gamma} + \beta)x_2 \tag{3.4}$$

This function is just a line, rather than the plane that selects the weights that allow both features to be represented in the mapping. However, depending on each instance of the coefficients in the data an optimizer could select α and β to fit to accommodate x_1 or x_2 for the middle and right hand sides in the Eqn. 3.4, respectively. Since either way the loss returned would be very small, this would result in a model that has high variances for its estimators of α and β, as some datapoints would cause wildly different fits depending on which way the optimizer is tuned. This is the exact sort of problem described in the section on regularization can occur when one weight in the network can be "blown up" and optimized to dominate the mapping without learning the underlying distribution that accounts for both features. Therefore Tsao notes that the high variances that result from multicollinear data result in very poor estimators for network weights. This is relevant for unregularized ReLU networks, since issues where weights will are updated dramatically between training sets may result in a ReLU network that maps to accomodate one of the two equivalent functions for y

in Eqn. 3.4. In practice this would manifest with longer convergence times during training, as well as more difficulties generalizing.

This would also illustrate what Gujarati notes is that in multicollinear data sets, one or more of the coefficients of a fit will be statistically insignificant when determining the dependent variables value. The example in Eqn. 3.4 would actually be called just "collinear" since there's not more than two independent variables to be correlated, but it does illustrate the issues a neural network may have fitting the underlying distribution of a multicollinear data set. The standard prescription in regression analysis for multicollinear data is to apply some regularization techniques to lessen the impact of the data sets high variance.

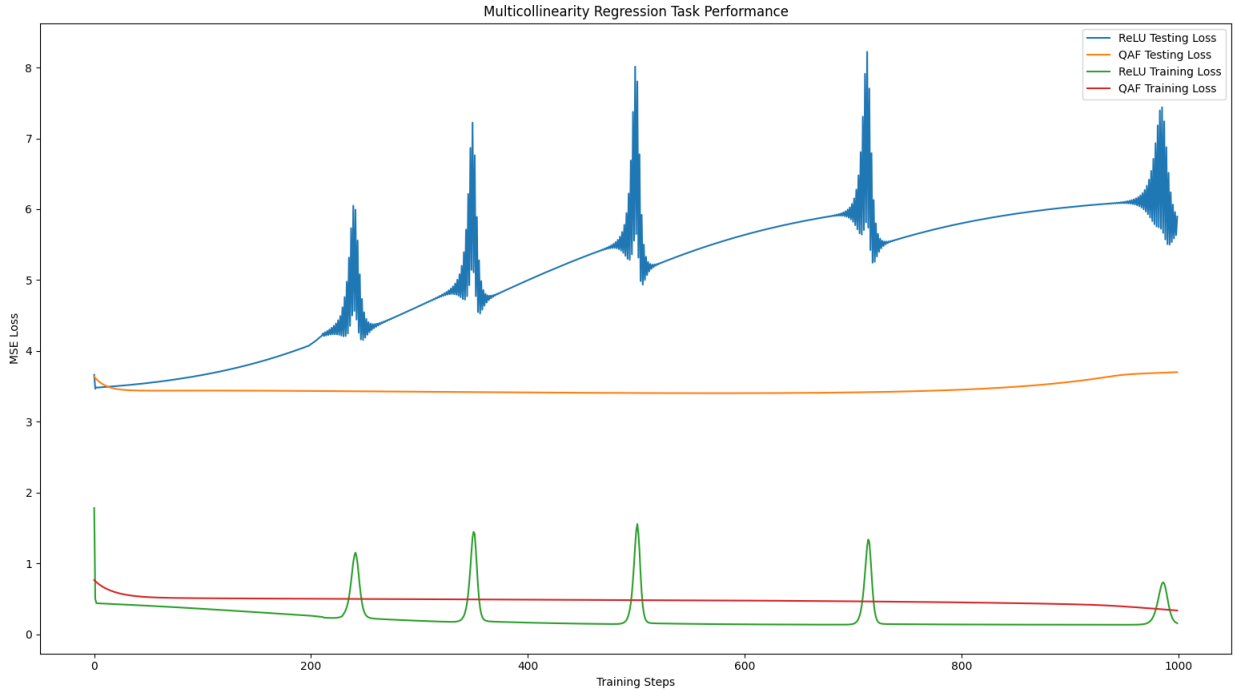

Figure 3.7. Multicollinear Regression Performance of QAF Network vs. Unregularized ReLU Network

The QAF network does not get as low of a training loss as the ReLU, but it does have significantly lower testing loss. This is expected for methods that introduce bias into a model. There is also the presence of jagged peaks in the training and testing loss o that was not seen in the previous tasks. One possible interpretation of this behavior is that as the optimizer approaches optimizing the network with values that are near singular, the loss requires a

rapid correction. This is why there are multiple peaks during the training. The bias the QAF adds to the network prevents the optimizer from tuning too closely to these singular values. When the L2-Regularized ReLU is run, it also does not show these jagged peaks, implying that it is the act of regularization that is stopping the optimizer from tuning the parameters close to those of a singular matrix. This task also shows that the QAF actually performs better and achieves a greater bias than the L2-Regularized ReLU network with $\lambda = .1$. This gives a sense that, while being somewhat dependent on the task, QAFs can apply some level of regularization. This also encourages testing the performance of this task with multiple L2-Regularized networks each with different regularization parameters to see if there is just some set amount of regularization that the QAF introduces into a model, or if it is just particularly adept at solving systems with rank deficient matrices.

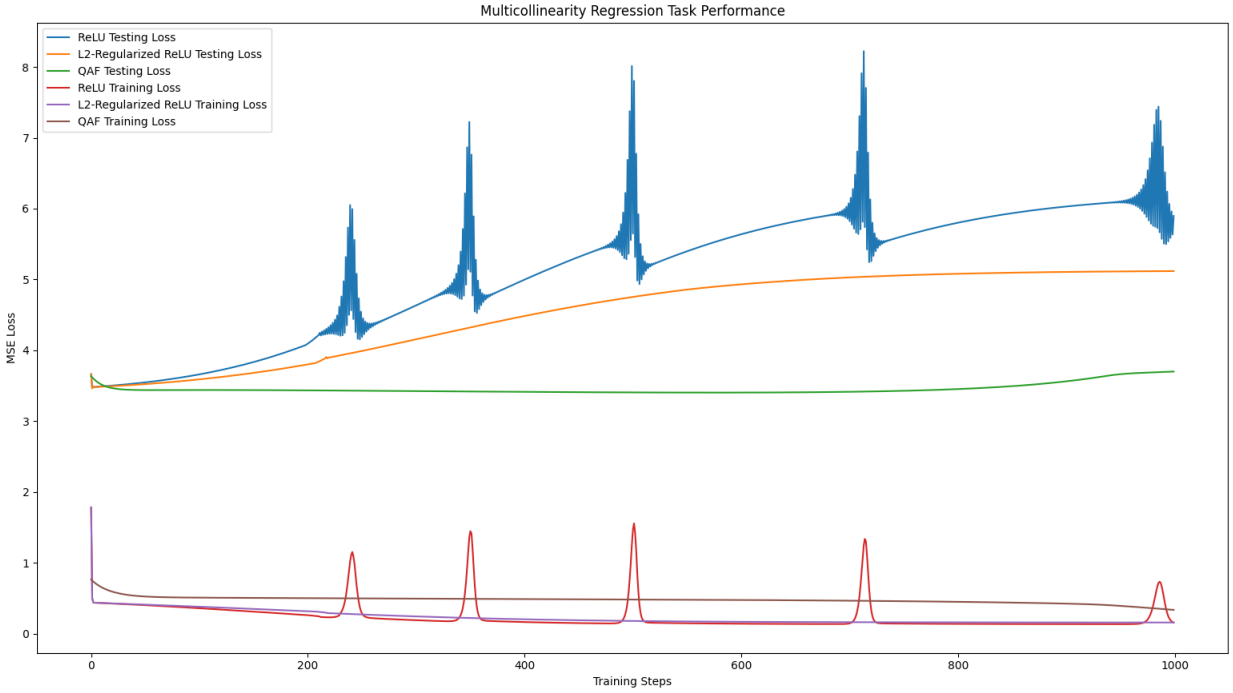

Figure 3.8. Multicollinear regression performance of the three networks

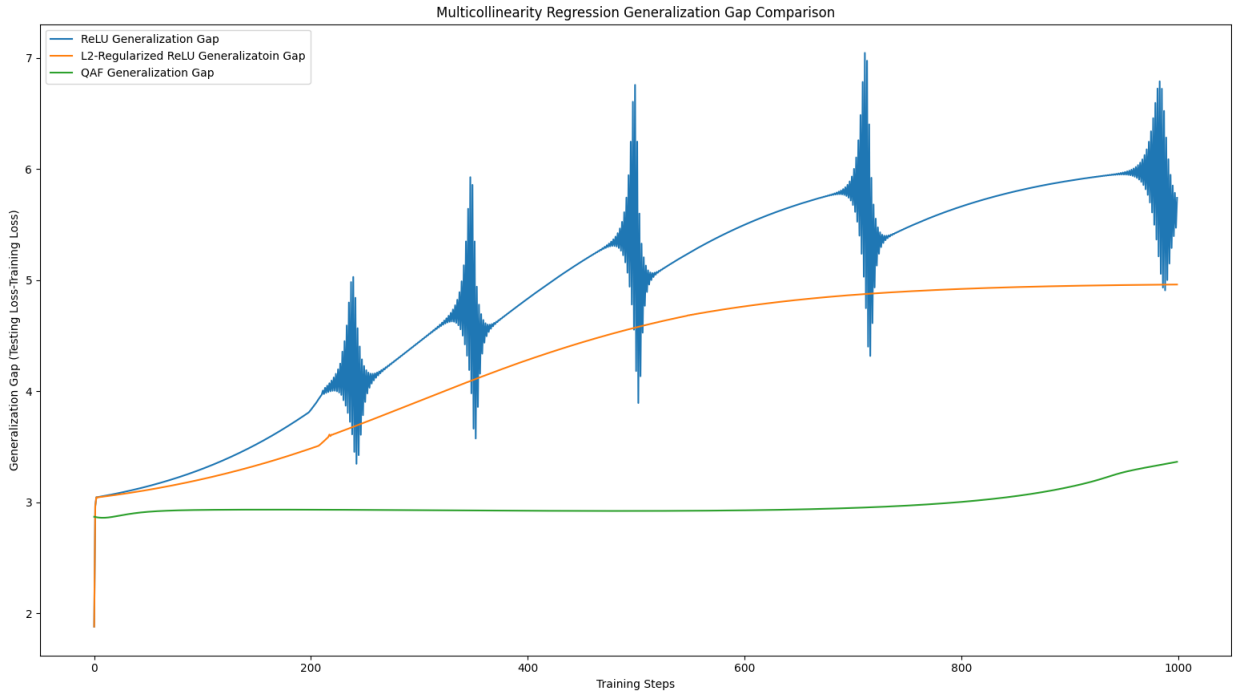

Figure 3.9. Multicollinearity regression generalization gaps of the three networks

3.4 Comparison to Different L2-Regularization Parameters

Now that it has been demonstrated that the QAF network, as well as the L2-Regularized ReLU network, is able to learn the training data as demonstrated by the training loss converging to a minimized value, now we can use the generalization gap performance to determine how much regularization the QAF is given compared to the traditional. One thing to note in this task is that it may be useful to also compare the regularization effects of the QAF network to a network with dropout layers, but due to the small size of the network thanks to size limitations of quantum circuit computations, dropout layers would likely introduce too much bias into the network, resulting in both training and testing error remaining high. This would give a misleading value of the generalization gap since the values would be low even though the training and testing error are high.

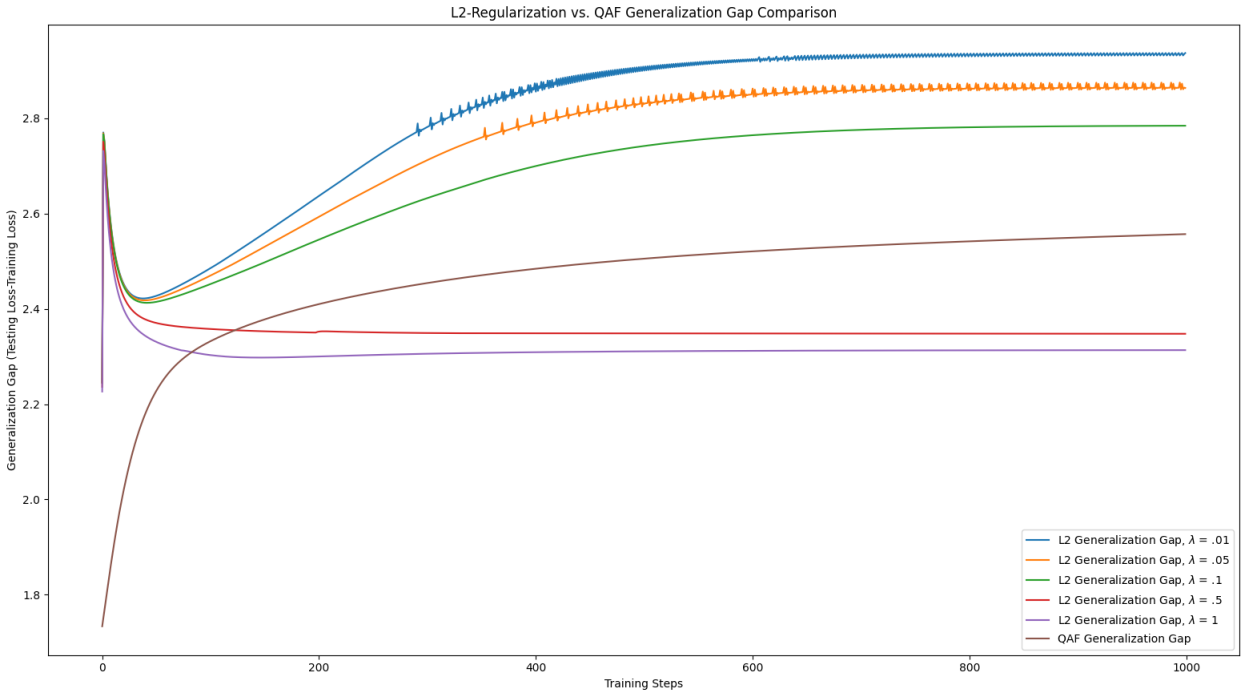

Figure 3.10. Comparison of QAF Network to multiple L2-Regularized Networks

Figure 3.10 shows that the amount of regularization applied by the QAF for the multi-collinearity regression task is comparable to L2-regularization with $\lambda \in [.1, .5]$. Note that, as in the earlier tasks, the losses for the L2-regularized networks and the QAF network do start off at different spots since initializing the linear layer weights to be equal for all the trial networks won't result in the same mapping for the first steps of training. Now while it appeared that in the first two learning tasks, the amount of regularization applied by the QAF was likely less than $\lambda = .1$ what this plot of generalization gap results demonstrates is that the QAF applies more regularization than some amounts of L2-regularization and less than others.

Since the operations the quantum circuit performs change the probability that a certain outcome will be measured, and those probabilities are reflected in the expectation we compute, if the gates of the circuit were modified to have the encoding push the qubit closer or farther away from the representation of the input vector on the Bloch sphere then the QAF would express more or less variance, respectively. For example, if the QAF overall was doing very little to transform the digital input vector and the output vector that is read out

was just a scaled version of the input, then there would be significantly less regularization introduced into the network. This implies that there may be something that can be modified about the structure of the QAF to change the amount of regularization applied, and this would likely be a result of exploring future research in how the architecture of QAF affect the performance of neural networks.

4. SUMMARY

In this paper, the validity of applying quantum circuits of the specified architecture was investigated, and the performance of neural networks with Quantum Activation Functions was compared to ReLU networks. First, the theory of regularization and quantum circuits was reviewed in order to justify design decisions when creating the layout of the QAF circuit. The design and theoretical motivations of the Quantum Activation Functions was then discussed with regard to their potential for mitigating overfitting in single-hidden-layer neural networks. An empirical study was then conducted comparing the performance of QAF networks against ReLU networks of the same dimensions, both with and without L2-Regularization, on three learning tasks: classfication, polynomial regression, and Regression of a multicollinear data set. These experimental results first demonstrated that single-hidden-layer neural networks with Quantum Activation Functions were able to learn a training data set to a comparable level as ReLU networks without regularization. The QAF networks were then shown to also be able to achieve a lower generalization gap than the ReLU networks without regularization, although not as low as ReLU networks regularized with L2-Regularization with a regularization parameter of $\lambda = .1$, except on the multicollinear data set. The amount of regularization applied to the model by the QAF network was attempted to be found experimentally by comparing it's generalization gap performance against several ReLU with L2-Regularization of varying parameters. This trial found that the regularization applied by the QAF network during the multicollinear data set regression task was comparable to regularization with L2-Regularization with a parameter of $\lambda \in [.1, .5]$. Recommendations for future research were then discussed, primarily potential work on how the layout of the QAF circuit affects the amount of regularization applied, as well as how this regularization might be dependent on the type of learning task.

5. RECOMMENDATIONS

The experiment comparing the QAF network's regularization ability to that of the regularized ReLU networks showed that on the multicollinearity regression task the QAF was applying a regularization comparable to L2-Regularization with $\lambda \in [.1, .5]$. This implies that they may be something that can be done to modify the gate layout of the QAF circuit to apply more or less regularization. This might be useful to the same degree that parameterizing L2-Regularization is, as machine learning practitioners sometimes find that they want to add or remove bias from their model depending on the situation. Future research could then be fruitful in finding what quantum gate operations could be changed in order to change the amount of regularization the QAF is applying.

This work would involve first finding a circuit that does as little regularization as possible. This may be done by just having an encoding layer that encodes each element of the input vector in such a way that the expectation of its qubit is roughly the value of the digital element divided by the magnitude of the input vector, since it will be restricted to be in the $[-1, 1]$ range at the time of measurement. Once this baseline QAF circuit is developed the effects of more complicated encoding will be measured, followed by the effects of varying amounts of entanglement. The goal of this research would then be to develop methods to vary the amount of regularization a QAF adds to a network with high predictability. This is most likely the next topic of research that would succeed this paper. The entanglement sections effect on the regularizing ability of the QAF would be also require extensive experimental development, since it is one of the more complicated aspects of quantum computing.

Another topic could stem from the fact that the QAF generalized better than the $\lambda = .1$ L2-Regularized ReLU network in the regression of multicollinear data task, but not in the polynomial regression or classification tasks. This means that there may be some tasks that QAF-based regularization might be more appropriate than L2-Regularization and vice versa. Determining stronger methods for predicting which is more appropriate would also likely require more experimental results.

Much of Neural Network research is still empirical results driven, and efforts in statistical learning theory are somewhat limited with regards to deep neural networks. However, the

rigorous results that do exist for neural networks are often achieved by narrowing the cone of focus to specific network architectures and scenarios, and so it is possible that by focusing more specifically on the QAF itself that there can be developments to find more precise ways of modifying the layout of the QAF circuit to achieve some desired outcome for a model. So there is potential that this future research may be able to be more theoretically-based than standard neural network research.

REFERENCES

[1] K. Hornik, "Approximation capabilities of mutlilayer feedforward networks," *Neural Networks*, vol. 4, pp. 251–257, 1990. DOI: https://doi.org/10.1037/0000126.

[2] C. Yun, S. Sra, and A. Jadbabaie, "Small relu networks are powerful memorizers: A tight analysis of memorization capacity," *33rd Conference on Neural Information Processing Systems*, 2019. DOI: https://doi.org/10.48550/arXiv.1810.07770.

[3] X. Zhang, Y. Yu, L. Wang, and Q. Gu, "Learning one-hidden-layer relu networks via gradient descent," *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics*, vol. 84, 2018. DOI: https://doi.org/10.48550/arXiv.1806.07808.

[4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[5] M. A. Little, *Machine Learning for Signal Processing: Data Science, Algorithms, and Computational Statistics*. Oxford University Press, 2019, ISBN: 0198714939.

[6] V. Vapnik and A. Chervonenkis, "On the uniform convergence of relative frequencies of events to their probabilities," *Theory of Probability and its Applications*, vol. 16, pp. 264–280, 2 1970. DOI: doi:10.1137/111602.

[7] M. Belkin, D. Hsu, S. Ma, and S. Mandal, "Reconciling modern machine-learning practice and the classical biasvariance trade-off," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 116, 32 2018. DOI: https://doi.org/10.48550/arXiv.1812.11118.

[8] M. Kobayashi, K. Nakaji, and N. Yamamoto, "Overfitting in quantum machine learning and entangling dropout," *Quantum Machine Intelligence*, vol. 30, 4 2023. DOI: https://doi.org/10.48550/arXiv.2205.11446.

[9] M. A. Nielsen and I. L. Chuang, *Machine Learning for Signal Processing: Data Science, Algorithms, and Computational Statistics*. Cambridge University Press, 2011, ISBN: 9781107002173.

[10] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii, "Quantum circuit learning," *PHYSICAL REVIEW A*, vol. 98, 2019. DOI: https://doi.org/10.48550/arXiv.1803.00745.

[11] S. Y.-C. Chen, S. Yoo, and Y.-L. L. Fang, "Quantum long short-term memory," *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020. DOI: https://doi.org/10.48550/arXiv.2009.01783.

[12] J. Preskill, "Approximation capabilities of mutlilayer feedforward networks," *Quantum*, vol. 2, pp. 79–99, 2018. DOI: https://doi.org/10.22331/q-2018-08-06-79.

[13] D. Arthur and P. Date, "A hybrid quantum-classical neural network architecture for binary classification," *A Hybrid Quantum-Classical Neural Network Architecture for Binary Classification*, 2022. DOI: https://doi.org/10.48550/arXiv.2201.01820.

[14] W. Jin, X. Liu, Y. Ma, C. Aggarwal, and J. Tang, "Feature overcorrelation in deep graph neural networks: A new perspective," *KDD '22: Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 709–719, 2022. DOI: https://doi.org/10.48550/arXiv.2206.07743.

[15] D. N. Gujarati, *Basic Econometrics*. McGraw-Hill, 1978, ISBN: 9780070251823.

[16] M. Tsao, "Group least squares regression for linear models with strongly correlated predictor variables," *Annals of the Institute of Statistical Mathematics, 2022*, 2022. DOI: https://doi.org/10.48550/arXiv.1804.02499.