Solving Binary Classification Problems with Quantum Neural Networks

James Saslow

***Abstract*—Quantum computing offers promising avenues in computational parallelism, and leveraging these quantum capabilities is an active area of research for machine learning. In this study, we explore the applications of Quantum Neural Networks (QNNs) in binary classification tasks, aiming to develop efficient models with minimal complexity. We present a simple QNN model for binary classification as a Parameterized Quantum Circuit (PQC) to encode input data and perform computations. Through experiments on 3 benchmark datasets - Iris, Breast Cancer Wisconsin, and MNIST, we demonstrate the effectiveness of our QNN approach. Despite the inherent challenges posed by non-linearly separable data, QNN models exhibit exceptional performance. Our findings highlight the promise of QNN in solving real-world classification tasks.**

Keywords: *Quantum Neural Networks, Quantum Machine Learning, Binary Classification, Iris, Breast Cancer Wisconsin, MNIST*

I. Introduction

A. Problem Statement

The primary objective of this research is to develop a simple Quantum Neural Network (QNN) with minimal model complexity capable of solving binary classification tasks. Although state-of-the-art classical neural networks have been shown to succeed at binary classification tasks, we apply *Quantum Machine Learning* paradigms to create a model with fewer trainable parameters and a possibility for exponential speed-ups [1]. Quantum computing leverages the quantum mechanical features of superposition and entanglement to grant faster computational speeds, which we take advantage of in our model. This research focuses on training a QNN classifier to perform binary classification on the following 3 datasets:

- *Iris Dataset* (Linearly Separable)
- *Breast Cancer Wisconsin (Original) Dataset* (Not Linearly Separable)
- *MNIST Dataset* (Not Linearly Separable)

Our simple QNN models linearly separable data exceptionally well up to 0% model error. With non-linearly separable data, such as the breast cancer and MNIST datasets, the QNN still prevails as a remarkable classifier, generating models with errors $< 13\%$. Classical perceptron models are also implemented as a 2-input 1-output neural network for comparison to their quantum counterparts (See Appendix C.)

II. Quantum Neural Network Architecture

This QNN has two input parameters $\mathbf{x} = (x_0, x_1)^T$, four trainable model parameters $\vec{\omega} = (\omega_0, \omega_1, \omega_2, \omega_3)^T$, and a single quantum measurement output that yields a binary value '0' or '1'. The '0' or '1' output is the QNN's prediction of the binary class assignment for the data point $\mathbf{x}_i$. The QNN is physically realized via construction of a Parameterized Quantum Circuit (PQC).

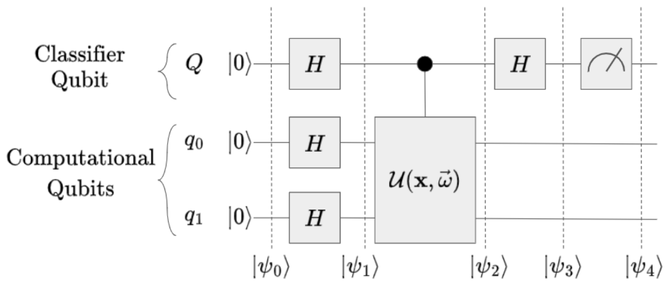


Figure: 1 Quantum Neural Network Parameterized Quantum Circuit

The PQC described in Figure 1 utilizes a *Classifier Qubit* register, which classifies the data point $\mathbf{x}_i$ upon measurement and a *Computational Qubit* register, which encodes information about $\mathbf{x}_i$ and does computations with model parameters $\vec{\omega}$. Each subprocess and construction of the quantum circuit will be described in the subsections below. For quantum gate definitions and Dirac notation formalism, see Appendix A.

A. Hadamard Transformation

As most quantum algorithms start, we begin the PQC with a uniform, coherent superposition state by enacting a Hadamard Transformation on the state of all zeros. Starting with the qubit register in the state of all zeros $|\psi_0\rangle = |0\rangle|00\rangle$, we get:

$$|\psi_1\rangle = \frac{1}{\sqrt{2}}\Big(|0\rangle + |1\rangle\Big)\frac{1}{\sqrt{N}}\sum_{j=0}^{N-1}|j\rangle \tag{1}$$

where $n \equiv$ number of computational qubits and $N \equiv 2^n$. In the case of our PQC in Figure 1, $n = 2$, $N = 4$.

B. Controlled - Unitary Construction

We begin constructing the Unitary matrix $\mathcal{U}(\mathbf{x}, \vec{\omega})$. [1] defines $\mathcal{U}(\mathbf{x}, \vec{\omega})$ to be

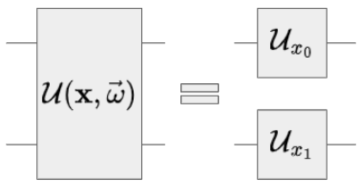


Figure: 2 Unitary $\mathcal{U}(\mathbf{x},\vec{\omega})$ Matrix Construction

where

$$\mathcal{U}_{x_j} \equiv \begin{pmatrix} e^{i\omega_{j0}x_j} & 0 \\ 0 & e^{i\omega_{j1}x_j} \end{pmatrix}$$

For simplicity, we relabel our model parameters such that $(\omega_{00},\omega_{01},\omega_{10},\omega_{11}) \to (\omega_0,\omega_1,\omega_2,\omega_3)$.

Taking the tensor product $\mathcal{U}(\mathbf{x},\vec{\omega}) = \mathcal{U}_{x_0} \otimes \mathcal{U}_{x_1}$, we get:

$$\mathcal{U} = \begin{pmatrix} e^{i\alpha_0} & 0 & 0 & 0 \\ 0 & e^{i\alpha_1} & 0 & 0 \\ 0 & 0 & e^{i\alpha_2} & 0 \\ 0 & 0 & 0 & e^{i\alpha_3} \end{pmatrix}$$

where $\vec{\alpha} = (\alpha_0,\alpha_1,\alpha_2,\alpha_3)^T$ are the phases of the eigenvalues of $\mathcal{U}(\mathbf{x},\vec{\omega})$. In particular,

$$\vec{\alpha} = \begin{pmatrix} \mathbf{x}\cdot(\omega_0,\omega_2)^T \\ \mathbf{x}\cdot(\omega_0,\omega_3)^T \\ \mathbf{x}\cdot(\omega_1,\omega_2)^T \\ \mathbf{x}\cdot(\omega_1,\omega_3)^T \end{pmatrix}$$

To learn how $\mathcal{U}(\mathbf{x},\vec{\omega})$ is actually implemented in quantum computing hardware, see Appendix B for the gate decomposition of $\mathcal{U}(\mathbf{x},\vec{\omega})$.

Using the unitary $\mathcal{U}(\mathbf{x},\vec{\omega})$, we can construct the controlled unitary using the following definition:

$$C\mathcal{U} \equiv |0\rangle\langle 0| \otimes \mathbb{I}^{\otimes 2} + |1\rangle\langle 1| \otimes \mathcal{U}$$

Recognizing that $\mathcal{U}\,|j\rangle = e^{i\alpha_j}\,|j\rangle$,

$$|\psi_2\rangle = \frac{1}{\sqrt{2N}}\left(|0\rangle \sum_{j=0}^{N-1} + |1\rangle \sum_{j=0}^{N-1} e^{i\alpha_j}\,|j\rangle \right) \tag{2}$$

C. The Periodic Activation Function

Although the phase information is properly encoded in the wavefunction in (eq 2), if we measure the classifier qubit at this stage, we will measure either the $|0\rangle$ or $|1\rangle$ state with 50% probability. The Hadamard gate following the Controlled-$\mathcal{U}$ acts as a periodic activation function, making the probability of measuring $|0\rangle$ or $|1\rangle$ dependent on the phases in $\vec{\alpha}$. The resultant wavefunction after applying a Hadamard to the Classifier Qubit Register is:

$$|\psi_3\rangle = \frac{1}{2\sqrt{N}}\left(|0\rangle \sum_{j=0}^{N-1} (1+e^{i\alpha_j})\,|j\rangle + |1\rangle \sum_{j=0}^{N-1} (1-e^{i\alpha_j})\,|j\rangle \right) \tag{3}$$

D. Measurement

Measuring the classifier qubit using the Born rule, we find

$$P_0 = \frac{1}{4N}\sum_{j=0}^{N-1} |1+e^{i\alpha_j}|^2 \tag{4}$$

$$= \frac{1}{2N}\sum_{j=0}^{N-1} \Big(1 + cos(\alpha_j)\Big) \tag{5}$$

$$P_1 = \frac{1}{4N}\sum_{j=0}^{N-1} |1-e^{i\alpha_j}|^2 \tag{6}$$

$$= \frac{1}{2N}\sum_{j=0}^{N-1} \Big(1 - cos(\alpha_j)\Big) \tag{7}$$

where p_0 is the probability of the classifier qubit being in the $|0\rangle$ state, and p_1 is the probability of the classifier qubit being in the $|1\rangle$ state.

E. Threshold Function

To extract the prediction of the QNN classifier, we round p_1 to the nearest whole number. This is known as the *threshold function*.

$$round(P_1|\mathbf{x}_k;\vec{\omega}) = \begin{cases} 0 & \implies \mathbf{x}_k \in \text{Class '0'} \\ 1 & \implies \mathbf{x}_k \in \text{Class '1'} \end{cases}$$

F. Cost Function

The cost function for this QNN is defined as

$$C = \frac{1}{2s}\sum_{j=0}^{s} (d_j - p_{1_j})^2 \tag{8}$$

where s is the number of data in the training set, d_j is the correct classification for data point $\mathbf{x}_j$, and p_{1_j} is the output of the QNN for $\mathbf{x}_j;\vec{\omega}$.

G. Gradient Descent

We update the model weights with the usual gradient descent learning scheme

$$\Delta\vec{\omega}_j = -\eta\nabla_{\vec{\omega}}C_j \tag{9}$$

where $\Delta\omega_j$ is the update to $\vec{\omega}$ from analyzing the gradient on the training data point $\mathbf{x}_j$, and η is the learning rate of the network.

The gradient can be written as $\nabla_{\vec{\omega}}C_j = (\frac{\partial C_j}{\partial\omega_0}, \frac{\partial C_j}{\partial\omega_1}, \frac{\partial C_j}{\partial\omega_2}, \frac{\partial C_j}{\partial\omega_3})$.

Therefore, we're interested in computing $\partial C_j/\partial\omega_k$ for $k \in \{0,1,2,3\}$ to determine a closed-form expression for our model parameter update rule. Applying the chain rule,

$$\frac{\partial C_j}{\partial\omega_k} = \frac{\partial C_j}{\partial P_{1_j}}\frac{\partial P_{1_j}}{\partial\alpha_j}\frac{\partial\alpha_j}{\partial\omega_k} \tag{10}$$

or

$$\frac{\partial C_j}{\partial \omega_k} = \frac{\partial C_j}{\partial P_{1_j}} \frac{\partial P_{1_j}}{\partial \omega_k} \tag{11}$$

we see a clear series of steps for calculating the gradient.

We find an analytical, closed-form expression for (eq 11) below:

$$\begin{cases} \frac{\partial C_j}{\partial P_{1_j}} = -\frac{1}{s}(d_j - P_{1_j}) \\ \frac{\partial P_{1_j}}{\partial \omega_k} \implies \begin{cases} \partial P_{1_j}/\partial \omega_0 = \frac{1}{8}x_0(sin\ \alpha_0 + sin\ \alpha_1) \\ \partial P_{1_j}/\partial \omega_1 = \frac{1}{8}x_0(sin\ \alpha_2 + sin\ \alpha_3) \\ \partial P_{1_j}/\partial \omega_2 = \frac{1}{8}x_1(sin\ \alpha_0 + sin\ \alpha_2) \\ \partial P_{1_j}/\partial \omega_3 = \frac{1}{8}x_1(sin\ \alpha_1 + sin\ \alpha_3) \end{cases} \end{cases}$$

III. Datasets

This research employs three datasets to train and test the performance of the QNN and CNN. Each data set undergoes pre-processing tasks to prepare the data for use as neural network inputs. These pre-processing strategies include normalization, feature encoding, and dimensionality reduction.

A. Iris Dataset

—-Different species of flowers and their characteristics—-

The Iris dataset from the UCI Machine Learning repository is a four-feature dataset with 150 instances intended for classification tasks. This dataset contains three classes of data. In particular, one class is linearly separable from the other two, but the other two classes are not linearly separable from each other. The features of this dataset are 'sepal length,' 'sepal width,' 'petal length,' and 'petal width.' The class instances are 'iris-virginica,' 'iris-versicolor,' and 'iris-setosa.'

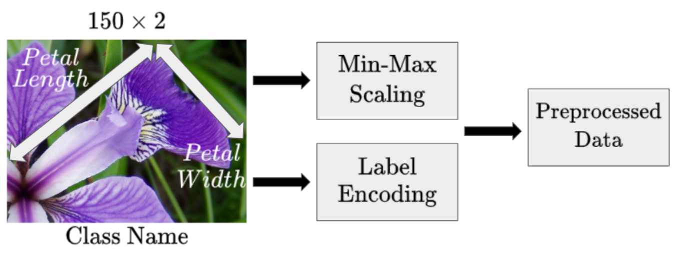


Figure: 3 Preprocessing Scheme for the Iris Dataset

Since our QML model can only take in two inputs, we utilize the 'petal length' and 'petal width' features and use Min-Max scaling to map them to an interval $\in [0, 1]$. Furthermore, we employ label encoding on the class instances, mapping them to binary variables. My choice of variable mapping is as follows:

- 'iris-virginica' $\rightarrow$ 1
- 'iris-versicolor' $\rightarrow$ 1
- 'iris-setosa' $\rightarrow$ 0

which results in a linearly separable dataset. This linear separability between the two classes becomes apparent in the following scatter plot of the data:

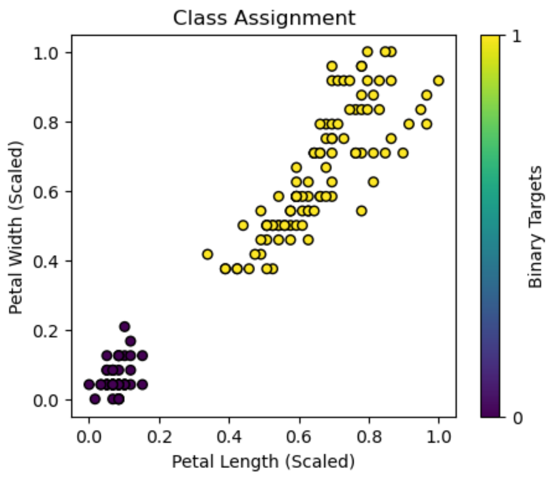


Figure: 4 Iris Dataset Class Assignment

Once all pre-processing steps are complete, each data point is randomly re-ordered in the data frame and exported as a *.csv* file to be used in our machine learning models.

B. Breast Cancer Wisconsin (Original) Dataset

—-Detecting Malignant or Benign Breast Cancer —-

The Breast Cancer Wisconsin (Original) Dataset from the UCI Machine Learning Repository is a 9-feature dataset with 699 instances and 2 classes of data. The features of this dataset are 'Clump thickness,' 'Uniformity of cell size,' 'Uniformity of cell shape,' 'Marginal Adhesion,' 'Single Epithelial Cell Size,' 'Bare Nuclei,' 'Bland Chromatin,' 'Normal Nucleoli,' and 'Mitosis' – all on an integer scale from 1 - 10. The class instances are '2', which indicates *benign*, and '4', which indicates *malignant*. To pre-process the data, we first Min-Max scale the 'Clump thickness' and 'Uniformity of cell size' features to fit on the interval $[0, 1]$. Then, we do label encoding to map the '2' and '4' targets to binary variables. In particular, I chose this specific mapping:

- '2' (Benign) $\rightarrow$ 0
- '4' (Malignant) $\rightarrow$ 1

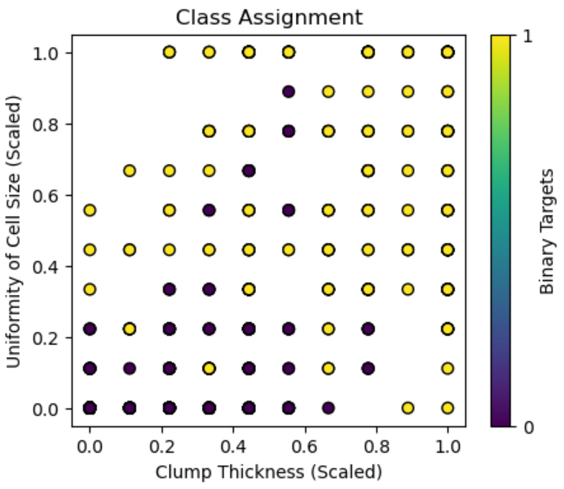


Figure: 5 Breast Cancer Wisconsin (Original) Class Assignment

Unlike the iris dataset, the breast cancer dataset is not linearly separable in clump thickness and uniformity of cell size.

Because our raw features are only on a 1-10 integer scale, our discrete domain has 100 allowable positions. Since there are 699 data samples, there are more data points than allowable positions, which means there will be overlap and non-linear separability between our two classes, making this binary classification problem fundamentally non-trivial.

Finally, each data point is randomly re-ordered in the data frame and exported as a *.csv* file for use in our machine learning models.

C. MNIST Dataset

—-Distinguishing Handwritten 0's & 1's—-

The MNIST digits classification dataset from the Keras library is a 28×28 feature dataset with 60,000 pre-allocated training samples and 10,000 testing samples. The MNIST dataset has 9 classes of data, one for each handwritten digit 0 through 9. Each feature is a grayscale pixel value of the 28×28 pixel image. We combine the pre-allocated training and testing datasets into one dataset and filter out the dataset for handwritten 0's and 1's to do binary classification. This results in a filtered data set of 14780 instances.

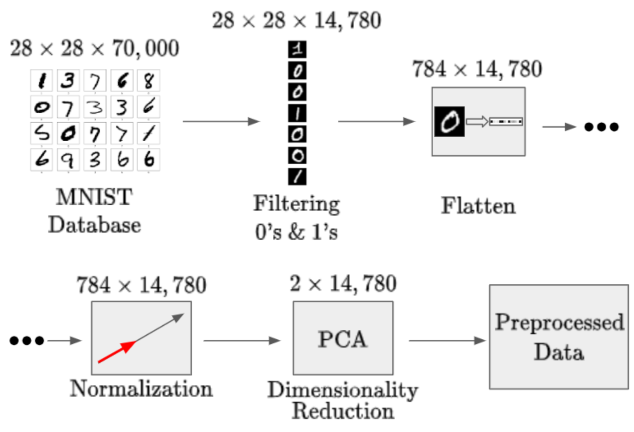


Figure: 6 Pre-processing Scheme for MNIST Dataset

We also have an apparent problem with our feature dimensionality. Each data point has $28 \times 28 = 784$ features, whereas our model can only take in 2 input features. We can apply PCA to reduce the dimensionality of each data point from $784 \rightarrow 2$. However, before we begin PCA, We should flatten each 28×28 matrix into a vector, then normalize that vector to prevent the PCA data from taking on large values.

We can also analyze plot our MNIST data in the PCA space.

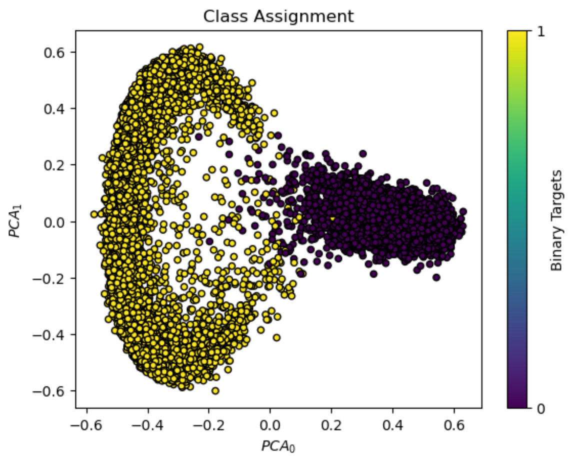


Figure: 7 MNIST Binary Digits Class Assignment

We notice an oval cluster form for the zeros and a c-shaped cluster form for the ones.

IV. Training and Performance

A. Iris Dataset Training

We allocate 80% of the data to training data and the remaining 20% to testing data, resulting in 120 training samples and 30 testing samples.

We run the QNN for 30 epochs at a learning rate of 1 for randomly generated model parameters $\vec{\omega}$ where $0 \leqslant \vec{\omega_j} < 2\pi$. The evaluation summary of the QNN at the 30th epoch can be described by Table I. We assess the performance of the QNN classifier by recording its predictions on test data and forming a *confusion matrix* as described by Figure 8.

As we can see, the QNN classifier successfully distinguishes between the two linearly separable classes without any misclassification in the test data.

We may further assess the performance of this binary classifier by calculating the *Accuracy*, *Precision*, *Recall (Sensitivity)*, *Specificity*, and the *False Positive Rate*.

By definition,

$$\begin{aligned}
\textit{Accuracy} &\equiv \frac{\text{TP} + \text{TN}}{m} \\
\textit{Precision} &\equiv \frac{\text{TP}}{\text{TP} + \text{FP}} \\
\textit{True Positive Rate} &\equiv \frac{\text{TP}}{\text{TP} + \text{FN}} \\
\textit{Specificity} &= \frac{\text{TN}}{\text{TN} + \text{FP}} \\
\textit{False Positive Rate} &\equiv 1 - \text{Specificity}
\end{aligned}$$

where m is the number of testing samples.

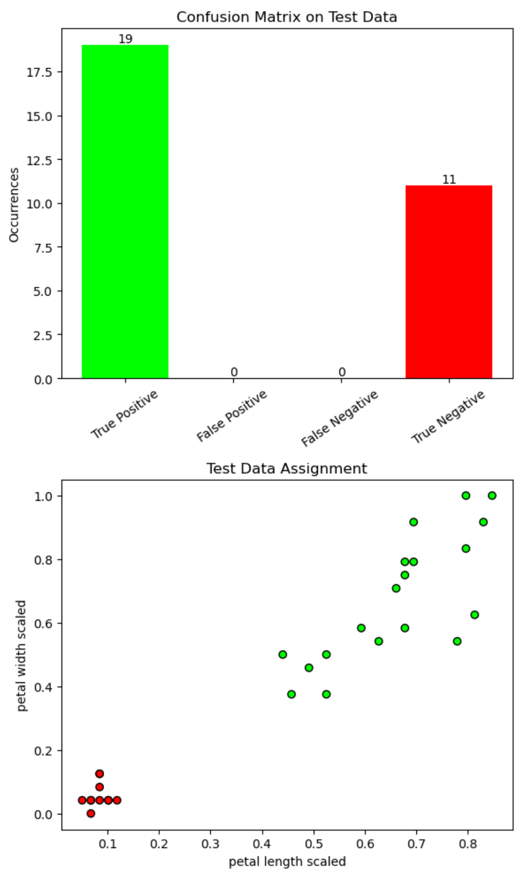


Figure: 8 QNN Iris Classifier Confusion Matrix

We can summarize the performance of the binary classifier by the following table

TABLE I
QNN Performance on Iris Data

Cost Function	0.02803
Training Model Error	0.0 %
Model Parameters $\vec{\omega}$	(0.82830, 2.97350, 3.09055, 1.40259)
Accuracy	1
Precision	1
True Positive Rate	**1**
Specificity	1
False Positive Rate	**0**

The Quantum Neural Network on the Iris test data yields perfect classification. However, this is not too surprising since the iris data is linearly separable, unlike the breast cancer and MNIST classification, which are non-trivial classification tasks.

B. *Breast Cancer Dataset Training*

Here we allocate 17% of the breast cancer dataset for training and the remaining 83% for testing data, resulting in 120 training samples and 579 testing samples.

After 30 epochs of training at a learning rate of 1, we get a confusion matrix given by the model's predictions on the test data:

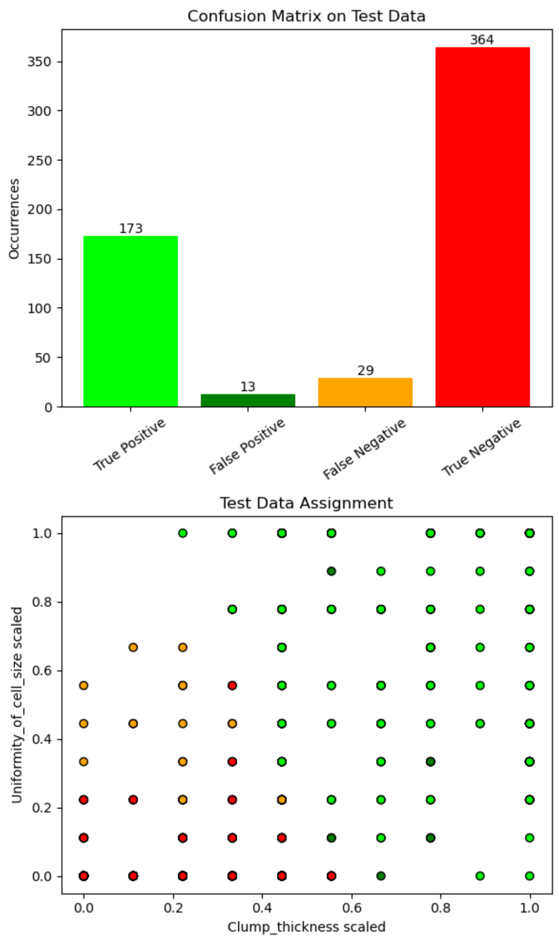


Figure: 9 QNN Breast Cancer Classifier Confusion Matrix

and a binary classifier performance given by the table below:

TABLE II
QNN Performance on Breast Cancer Data

Cost Function	0.03158
Training Model Error	5.0 %
Model Parameters $\vec{\omega}$	(2.79292, 2.54097, 0.37259, 2.08065)
Accuracy	0.927461
Precision	0.930108
True Positive Rate	**0.856436**
Specificity	0.965517
False Positive Rate	**0.034483**

C. *MNIST Dataset Training*

Here, we allocate 68% of the MNIST data for training and the remaining 32% for testing data, resulting in $10,000$ training samples and $4,780$ testing samples.

To train the QNN on MNIST, we take a different approach compared to the two previous training examples since the MNIST dataset is so large. We take the following approach using *Incremental Learning*:

Algorithm 1: Incremental Learning Algorithm

```
Data: Initial model parameters $\vec{\omega}$, MNIST dataset $D$
Result: Trained neural network parameters $\vec{\omega}$
Initialize neural network parameters $\vec{\omega}$;
foreach mini-batch $B$ in dataset $D$ do
    foreach epoch do
        foreach point $x$ in mini-batch $B$ do
            Compute gradient of the cost function on
              point $x$ using current parameters $\vec{\omega}$;
            Use gradient descent to update model
              parameters $\vec{\omega}$;
        end
    end
    if mini-batch $B = D$ and model is trained then
        break;
    end
    else
        Select the next progressively larger mini-batch
          $B \in D$;
    end
end
return Trained neural network parameters $\vec{\omega}$
```

Here were my choices for the *mini-batch* B while training the QNN on the MNIST training data set D:

1) Train model on data $B_0 = (D_0, .., D_{199})$ for 30 epochs at $lr = 1$.
2) Train model on data $B_1 = (D_0, .., D_{599})$ for 30 epochs at $lr = 1$.
3) Train model on data $B_2 = (D_0, .., D_{1499})$ for 30 epochs at $lr = 1$.
4) Train model on data D for 10 epochs at $lr = 180$.

After training, the model's predictions on the test data are given by the following confusion matrix in Figure 10. We see that the QNN classifier doesn't detect the C-shaped cluster and the shape cluster as two separate classes but rather appears to draw two horizontal linear boundaries that encapsulate the True negatives in the oval cluster and intersect the C-shaped cluster, resulting in the majority of false negative errors.

The binary classifier performance is given by the table below:

TABLE III
QNN Performance on MNIST Data

Cost Function	0.04681
Training Model Error	12.11%
Model Parameters $\vec{\omega}$	(1.37732, -0.94510, 7.40563, 7.52189)
Accuracy	0.87992
Precision	0.99797
True Positive Rate	**0.77488**
Specificity	0.99822
False Positive Rate	**0.00178**

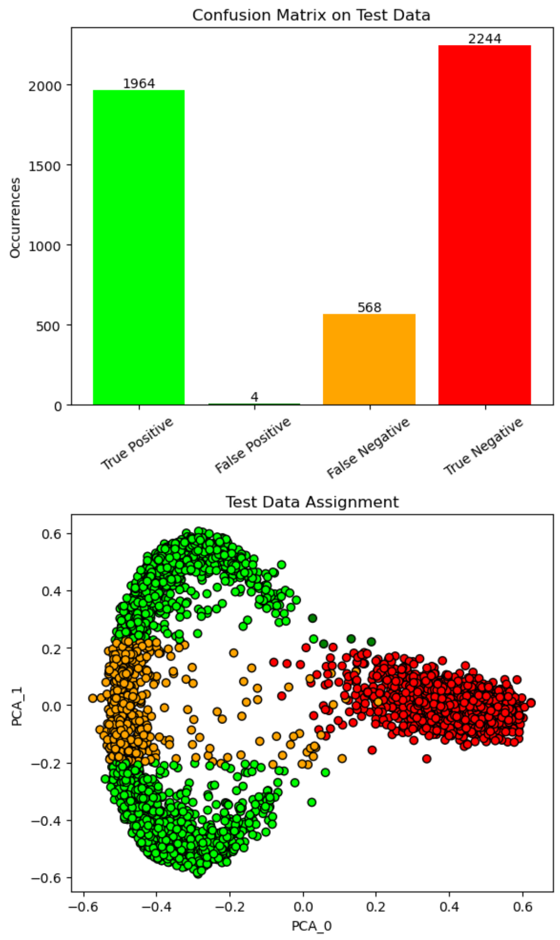


Figure: 10 QNN MNIST Classifier Confusion Matrix

V. Demo

A *recorded demo* of the code running can be found here on Youtube: https://www.youtube.com/watch?v=VPK-RhlEyBI

VI. Summary

In this paper, we explore the applications of Quantum Neural Networks (QNNs) for binary classification tasks. Our research aims to develop a simple QNN model capable of achieving binary classification across 3 benchmarking datasets, including non-linearly separable data. Although we rely on the assumption that QNN's truly hold an advantage on error-corrected hardware, contemporary quantum devices are noise-dominated, which complicates being able to physically realize a QNN. Our model being deliberately simplified is also a constraint. Our QNN proposed in this paper is a 2-input, 1-output network. However, [1] provides instruction on how to generalize this network to handle multidimensional data to classify k number of classes instead of only 2. Although several quantum hardware issues pose significant challenges to fully utilize quantum computing, this research demonstrates the potential of a functioning QNN to solve real-world binary classification problems.

References

[1] A. Daskin, "A simple quantum neural net with a periodic activation function," 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Oct. 2018. doi:10.1109/smc.2018.00491

[2] R. A. Fisher, "Iris," UCI Machine Learning Repository, https://archive.ics.uci.edu/dataset/53/iris (accessed Apr. 29, 2024).

[3] Wi. Wolberg, "Breast cancer Wisconsin (original)," UCI Machine Learning Repository, https://archive.ics.uci.edu/dataset/15/breast+cancer+wisconsin+original (accessed Apr. 29, 2024).

[4] K. Team, "Keras Documentation: MNIST Digits Classification Dataset," Keras, https://keras.io/api/datasets/mnist/ (accessed Apr. 29, 2024).

[5] H. Y. WONG, Introduction to Quantum Computing: From a Layperson to a Programmer in 30 Steps. S.l.: SPRINGER INTERNATIONAL PU, 2023.

[6] D. Koch, L. Wessing, and P. M. Alsing, "Introduction to coding quantum algorithms: A tutorial series using Qiskit," arXiv.org, https://arxiv.org/abs/1903.04359 (accessed Apr. 29, 2024).

[7] D. Koch, S. Patel, L. Wessing, and P. M. Alsing, "Fundamentals in quantum algorithms: A tutorial series using Qiskit continued," arXiv.org, https://arxiv.org/abs/2008.10647 (accessed Apr. 20, 2024).

VII. Appendix A: Quantum Gate Definitions and Dirac Notation

TABLE IV
The Pauli Spin Matrices, their eigenvectors, and eigenvalues

$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	$\sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$	$\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
$\lvert + \rangle = \frac{1}{\sqrt{2}} \lvert 0 \rangle + \frac{1}{\sqrt{2}} \lvert 1 \rangle$ $\lvert - \rangle = \frac{1}{\sqrt{2}} \lvert 0 \rangle - \frac{1}{\sqrt{2}} \lvert 1 \rangle$	$\lvert +i \rangle = \frac{1}{\sqrt{2}} \lvert 0 \rangle + \frac{i}{\sqrt{2}} \lvert 1 \rangle$ $\lvert -i \rangle = \frac{1}{\sqrt{2}} \lvert 0 \rangle - \frac{i}{\sqrt{2}} \lvert 1 \rangle$	$\lvert 0 \rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ $\lvert 1 \rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$
$\sigma_x \lvert + \rangle = (+1) \lvert + \rangle$ $\sigma_x \lvert - \rangle = (-1) \lvert - \rangle$	$\sigma_y \lvert +i \rangle = (+1) \lvert +i \rangle$ $\sigma_y \lvert -i \rangle = (-1) \lvert -i \rangle$	$\sigma_z \lvert 0 \rangle = (+1) \lvert 0 \rangle$ $\sigma_z \lvert 1 \rangle = (-1) \lvert 1 \rangle$

TABLE V
Other Common 1 & 2 Qubit Gates

Hadamard Gate	$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$
Phase Gate	$P(\theta) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}$
CNOT Gate	$CX = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$
Control Phase Gate	$CP(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\theta} \end{pmatrix}$

The state of a qubit can be represented by the arbitrary wavefunction $\lvert \psi \rangle = a \lvert 0 \rangle + b \lvert 1 \rangle$ where $a, b \in \mathbb{C}$. This wavefunction state is not a physical state of the qubit, but rather a state of information. We extract information about $\lvert \psi \rangle$ by examining probability amplitudes using the Born Rule. In particular $P_0 = \lvert \langle 0 \vert \psi \rangle \rvert^2 = \lvert a \rvert^2$ and $P_1 = \lvert \langle 1 \vert \psi \rangle \rvert^2 = \lvert b \rvert^2$ where a and b are constrained by $\lvert a \rvert^2 + \lvert b \rvert^2 = 1$ due to being probability amplitudes.

VIII. Appendix B: Gate Decomposition of $\mathcal{U}(\mathbf{x}, \vec{\omega})$

In this section, I will attach a Mathematica Notebook showing that $\mathcal{U}(\mathbf{x}, \vec{\omega})$ can be written in terms of control-phase gates and NOT gates. (See embedded PDF below)

Constructing U with Matrices

In[1]:=
```
(*Re-labeling (ω00,ω01,ω10,ω11) → (ω0,ω1,ω2,ω3) *)
(*Matrix Composition for U given by [1]*)

Ux1 = {{Exp[ⅈ ω0 x0], 0},
       {0, Exp[ⅈ ω1 x0]}};
Ux2 = {{Exp[ⅈ ω2 x1], 0},
       {0, Exp[ⅈ ω3 x1]}};
UMatrix = KroneckerProduct[Ux1, Ux2];
UMatrix // MatrixForm
```

Out[4]//MatrixForm=

$$\begin{pmatrix} e^{i\, x0\, \omega 0 + i\, x1\, \omega 2} & 0 & 0 & 0 \\ 0 & e^{i\, x0\, \omega 0 + i\, x1\, \omega 3} & 0 & 0 \\ 0 & 0 & e^{i\, x0\, \omega 1 + i\, x1\, \omega 2} & 0 \\ 0 & 0 & 0 & e^{i\, x0\, \omega 1 + i\, x1\, \omega 3} \end{pmatrix}$$

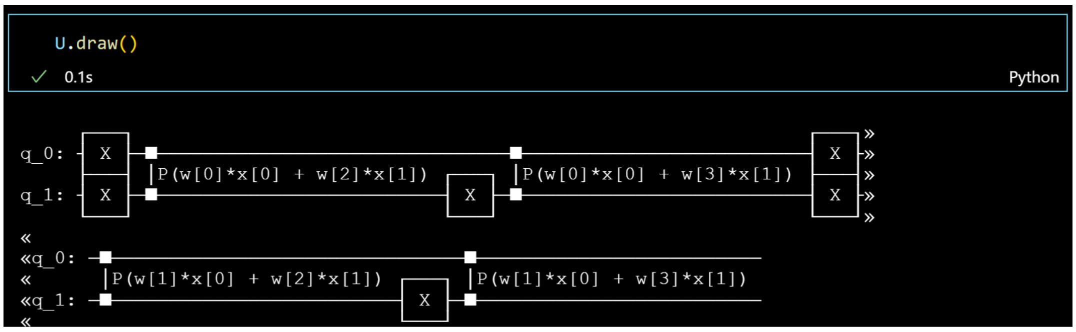
Constructing U From Gates

In[16]:=
```
(*Writing the Matrix form of Quantum Gates*)
X = {{0, 1}, {1, 0}}; (*NOT Gate*)
Iden = IdentityMatrix[2]; (*Identity*)
II = IdentityMatrix[4]; (*2-Qubit Identity*)
XX = KroneckerProduct[X, X]; (*2-Qubit NOT Gate*)
XI = KroneckerProduct[X, Iden]; (*NOT0 Identity1*)
IX = KroneckerProduct[Iden, X]; (*Identity0 NOT1*)

CP[θ_] := {{1, 0, 0, 0},
           {0, 1, 0, 0},
           {0, 0, 1, 0},
           {0, 0, 0, Exp[ⅈ θ]}} (*Control - Phase Gate*)

(*Writing the Phases of the Eigenvalues of U*)
α0 = x0 * ω0 + x1 * ω2;
α1 = x0 * ω0 + x1 * ω3;
α2 = x0 * ω1 + x1 * ω2;
α3 = x0 * ω1 + x1 * ω3;
```

Quantum Circuit Decomposition of the Unitary U



```
U.draw()
✓ 0.1s                                                                          Python
```

```
     ┌───┐                                                             ┌───┐»
q_0: ┤ X ├─■───────────────────────────────────■───────────────────────┤ X ├»
     ├───┤ │P(w[0]*x[0] + w[2]*x[1]) ┌───┐ │P(w[0]*x[0] + w[3]*x[1]) ├───┤»
q_1: ┤ X ├─■─────────────────────────┤ X ├─■─────────────────────────┤ X ├»
     └───┘                           └───┘                           └───┘»
«
«q_0: ─■─────────────────────────────────■─────────────────────────
«      │P(w[1]*x[0] + w[2]*x[1]) ┌───┐ │P(w[1]*x[0] + w[3]*x[1])
«q_1: ─■─────────────────────────┤ X ├─■─────────────────────────
«                                └───┘
```

In[27]:=
```
(*Constructing U in terms of Quantum Gates*)
Ugate = CP[α3].IX.CP[α2].XX.CP[α1].IX.CP[α0].XX // FullSimplify;
Ugate // MatrixForm
```

Out[28]//MatrixForm=

$$\begin{pmatrix} e^{i\,(x0\,\omega 0+x1\,\omega 2)} & 0 & 0 & 0 \\ 0 & e^{i\,(x0\,\omega 0+x1\,\omega 3)} & 0 & 0 \\ 0 & 0 & e^{i\,(x0\,\omega 1+x1\,\omega 2)} & 0 \\ 0 & 0 & 0 & e^{i\,(x0\,\omega 1+x1\,\omega 3)} \end{pmatrix}$$

Confirming that UMatrix = Ugate

In[29]:=
```
Print["UMatrix = ", UMatrix // MatrixForm]
Print["Ugate=", Ugate // MatrixForm]

Print["Does UMatrix = UGate?: ", UMatrix == Ugate // FullSimplify]
```

$$\text{UMatrix} = \begin{pmatrix} e^{i\,x0\,\omega 0+i\,x1\,\omega 2} & 0 & 0 & 0 \\ 0 & e^{i\,x0\,\omega 0+i\,x1\,\omega 3} & 0 & 0 \\ 0 & 0 & e^{i\,x0\,\omega 1+i\,x1\,\omega 2} & 0 \\ 0 & 0 & 0 & e^{i\,x0\,\omega 1+i\,x1\,\omega 3} \end{pmatrix}$$

$$\text{Ugate=}\begin{pmatrix} e^{i\,(x0\,\omega 0+x1\,\omega 2)} & 0 & 0 & 0 \\ 0 & e^{i\,(x0\,\omega 0+x1\,\omega 3)} & 0 & 0 \\ 0 & 0 & e^{i\,(x0\,\omega 1+x1\,\omega 2)} & 0 \\ 0 & 0 & 0 & e^{i\,(x0\,\omega 1+x1\,\omega 3)} \end{pmatrix}$$

```
Does UMatrix = UGate?: True
```

IX. Appendix C: Quantum Neural Network and Classical Perceptron Performances

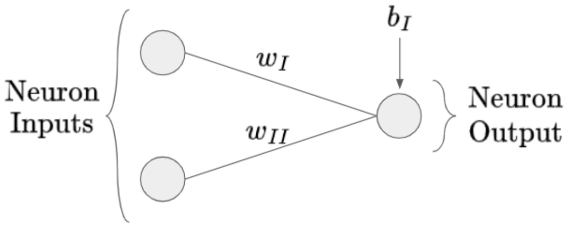


Figure: 9 Perceptron Neural Network Architecture

TABLE VI
QNN vs Perceptron Performance on Iris Data

Metric	Quantum Neural Network	Perceptron
Accuracy	1	1
Precision	1	1
True Positive Rate	**1**	**1**
Specificity	1	1
False Positive Rate	**0**	**0**

TABLE VII
QNN vs Perceptron Performance on Breast Cancer Data

Metric	Quantum Neural Network	Perceptron
Accuracy	0.927461	0.927461
Precision	0.930108	0.965116
True Positive Rate	**0.856436**	**0.821782**
Specificity	0.965517	0.984085
False Positive Rate	**0.034483**	**0.015915**

TABLE VIII
QNN vs Perceptron Performance on MNIST Data

Metric	Quantum Neural Network	Perceptron
Accuracy	0.879916	0.996444
Precision	0.997965	0.997626
True Positive Rate	**0.774882**	**0.995656**
Specificity	0.998221	0.997331
False Positive Rate	**0.001779**	**0.002669**