Towards Multivariate Regression using Quantum Neural Networks

Dissertation Manuscript

Submitted to National University

School of Technology and Engineering

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

by

LEO M. RAVELO

San Diego, California

September 2023

Approval Page

Towards Multivariate Regression using Quantum Neural Networks

By

LEO M. RAVELO

Approved by the Doctoral Committee:

DocuSigned by: John Enamait F381584A61AE418...	PhD	12/02/2023 | 11:55:09 MST
Dissertation Chair: John Enamait	Degree Held	Date
DocuSigned by: David Bouvin B55DC3A06360469...	PhD, DBA	11/29/2023 | 19:59:39 MST
Committee Member: David Bouvin	Degree Held	Date
DocuSigned by: William Tribbey 1240E67C9457406...	PhD	11/30/2023 | 06:55:24 MST
Committee Member: William Tribbey	Degree Held	Date

Abstract

Quantum computing is an emerging field in computer science that can offer advantages by exploiting the principles of quantum mechanics. This investigation focused on the design and construction of a quantum neural network model for multivariate multiple regression problems. The problem addressed by this study was the lack of understanding on how to train a multivariate multiple regression model using quantum neural networks to predict multivariate responses. Using a constructive research methodology, this investigation incorporated elements of classical computing and quantum computing to produce a novel approach that leverages quantum neural networks to predict multiple targets when trained with a multivariate multiple regression dataset. Experiments conducted in this investigation explored optimal configurations of different quantum circuits and quantum gates to encode the classical data into effective feature maps and weights. This manuscript introduced a multiple target quantum neural network regressor class as a Python module researchers can use for multivariate multiple regression problems. This new module provides methods for encoding and scaling classical data, along with methods for evaluating the performance of the trained model. The performance of this approach was compared to the performance of classical neural networks by training both models with the same datasets. The results revealed that the quantum neural network approach when run in a simulation can achieve similar performance metrics as a classical model, with the exception of the total runtime. Due to current hardware constraints, running the quantum model on a quantum processor is cost prohibitive. Future research should explore incorporating graphics processing units to improve the training time of the models. Additional studies should also explore optimizing the training process by parallelizing the training of the underlying neural networks as well as alternative combinations of quantum circuits and gate operations.

Acknowledgements

To my eternal partner and wife, Rosemary, whose unconditional love and support has helped me overcome the most difficult challenges in my life, thank you. You inspire me to become a better version of myself every day, my love. Thank you for always supporting me and following me to the ends of the Earth. I love you.

To my children, Roselea, Lily, and Leo. Thank you for making me the proudest dad in the world for having the most understanding and supportive children for whom I could ever hope. Daddy did this for you, my babies. We will make up for lost time effective immediately.

In loving memory of my dad, Alex, whom we lost unexpectedly last year. I still cannot put into words how much I miss you and wish you were still here. You sacrificed your dreams and worked tirelessly to provide for our family. This is your achievement too, Dad. I am forever proud of you.

To my mother, Leslie. Thank you, Mom, for instilling in me the value of hard work and education. I was not the easiest son to raise, and I thank you for always believing in me and helping me realize my full potential. This is your achievement too, Mom. I love you.

To my grandparents, brother, sister, family, and my colleagues at Northrop Grumman, thank you for your continued support and faith in me.

A special thank you to my doctoral committee, Dr. John Enamait, Dr. David Bouvin, and Dr. Will Tribbey. Dr. Enamait, thank you for your empathy while helping me navigate this final chapter of my studies during the most difficult moments of my life.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Quantum computation is a growing branch of computer science that exploits the laws of quantum mechanics in hopes of achieving exponential speed-ups (Bozzo-Rey et al., 2019). Potential applications of quantum computing include the domains of cryptography, chemistry, physics, simulation, optimization, secure communications, and machine learning (Rieffel et al., 2019). Quantum versions of machine learning algorithms such as linear regression, k-means clustering, principal component analysis, support vector machines, and neural networks are now possible on quantum hardware with the availability of quantum libraries and cloud-based quantum services (Pattanayak, 2021). Quantum neural networks for classification problems are a popular research theme, especially in image classification experiments (Farhi & Neven, 2018; Jiang et al., 2021; Potempa & Porebski, 2022). However, the study of quantum neural networks for regression problems has received far less attention.

Classical computers use the basic unit of bits to represent two possible states, 0 or 1. In quantum computers, the basic units of information are called *qubits* and are represented using Dirac, or *bra-ket*, notation as $|0\rangle$ and $|1\rangle$ (Nielsen & Chuang, 2010). The primary distinction of a qubit from a classical bit is that the qubit can be in a *superposition* state formed by a linear combination of the two possible states $|0\rangle$ and $|1\rangle$. Representing multiple states simultaneously could enable a quantum algorithm to exhibit an exponential speed-up compared to a classical algorithm (Moran, 2019). This advantage, called *quantum supremacy*, may be possible with 50-100 qubits (Bozzo-Rey et al., 2019). A recent publication by scientists on the Google AI Quantum team claims to have achieved quantum supremacy (Arute et al., 2019). The team's experiment involved using a quantum computer with 53 qubits representing a state space of 2^{53} ($\approx 10^{16}$) dimensions to sample a quantum circuit a million times in 200 seconds. This same task

was estimated to take 10,000 years on a state-of-the-art supercomputer. This exponential computing power increase is what makes quantum computing particularly attractive for practical investigations.

The computational capabilities of quantum computers has attracted machine learning researchers to investigate implementations of these algorithms on quantum simulators and quantum hardware (Schuld et al., 2015). The widespread availability of quantum programming languages, libraries, and cloud services such as IBM's *Qiskit* and *IBMQ Experience*, Microsoft's *Q#* and *Azure Quantum*, Google's *Cirq*, and Amazon's *Braket* have made advancements in this domain possible without the need for researchers to purchase quantum hardware (Garhwal et al., 2021; McCaig, 2021). Several exponential speed-ups of machine learning algorithms on cloud-based quantum hardware have already been realized in recent experiments that involved training support vector machines (Saini et al., 2020). Further experimentation could help accomplish the same for quantum neural networks.

Artificial neural networks are popular for classification tasks but have also demonstrated excellent performance and prediction accuracies for linear, non-linear, and multivariate statistical regression problems (Carvalho et al., 2013; Goyal & Goyal, 2012). In the past, multivariate regression techniques were seldom used and instead employed different univariate regression methods to predict multiple dependent variables (Breiman & Friedman, 1997). Since then, various techniques including statistical (Borchani et al., 2015) and machine learning techniques (Alahassa, 2021; Xu et al., 2019) for approaching multivariate regression problems have emerged. Further research is needed to determine if a quantum neural network could offer any advantages when applied to multivariate regression problems.

Statement of the Problem

This dissertation report introduced an approach to creating a multivariate multiple regression model using quantum neural networks. The problem this investigation addressed was the lack of guidance available to researchers on how to prepare and train a quantum neural network model so that it can effectively predict multivariate responses. Quantum machine learning has seen significant advances with quantum versions of algorithms such as *k-nearest neighbors* (Sharma, 2020), *support vector machines* (Rebentrost et al., 2014; Saini et al., 2020), *generative adversarial networks* (Dallaire-Demers & Killoran, 2018; Lloyd & Weedbrook, 2018), *Boltzmann machines* (Amin et al., 2018), and *deep neural networks* (C. Zhao & Gao, 2021). Recent experiments also included classification problems (Asolkar, 2020; Macaluso et al., 2020b; Patel & Tiwari, 2020; Schuld et al., 2020), generative adversarial problems (Li et al., 2021; Niu et al., 2021a; S. A. Stein et al., 2021), and univariate regression problems (Kairon & Bhattacharyya, 2020; Quoc et al., 2021; Reddy & Bhattacherjee, 2021). Thus far, research is limited on the usage of quantum neural networks for multivariate regression problems.

Multivariate (or multi-output) regression is the task of training a model to simultaneously predict multiple response variables given a set of prediction variables (Xu et al., 2019). Multivariate regression is also referred to as *multi-target*, *multi-response*, *multi-objective*, or *multi-output* regression and can be approached with problem transformation techniques, algorithm adaptation methods, statistical methods, or machine learning techniques (Borchani et al., 2015). Artificial neural networks have been demonstrated to perform well with multivariate regression problems with large datasets (Moosavi et al., 2017). The results of this investigation helped answer how to train a multivariate multiple regression model using quantum neural networks and how the performance of the model compares to a classical version of the model.

Purpose of the Study

The purpose of this investigation was to create a quantum neural network model capable of predicting multivariate responses. This research helped establish a comprehensive strategy for training a quantum neural network for multivariate multiple regression problems along with techniques for assessing the model's performance. The investigation was conducted using the Python programming language and the Qiskit quantum software development kit to implement the quantum neural network model.

Through experimentation, various hyperparameters were tested to find the optimal configuration (i.e., loss function, optimizer, number of iterations) for training the quantum neural network to perform the regression task. The experiments used a synthetic multivariate regression dataset for training and testing. The data was split using various split ratios, including the conventional 80/20 split rule (Shao et al., 2021) to train the neural network using 80% of the dataset and validate the results using the remaining 20%. Additional models were also trained using 70/30 and 60/40 split ratios to identify the model with the best performance metrics.

The experimental code was tested within Qiskit's built-in quantum simulator running on an enterprise-grade cloud server. A classical neural network was also trained using the same regression data to compare its performance with the quantum model. Performance metrics included the *mean absolute error* (MAE), *mean squared error* (MSE), *root mean squared error* (RMSE), and *R-squared* (R^2) values.

This investigation contributed to existing methods by establishing a formal process of training a quantum neural network model to predict multivariate responses and defining formal methods on how to assess the model's performance. The final artifact was a multitarget quantum

neural network regression model implemented as a Python class with which other researchers can experiment.

Introduction to Conceptual Framework

Previous research using quantum neural networks for regression problems involved training the models to predict univariate responses (Bhattacharyya et al., 2015; Reddy & Bhattacherjee, 2021; Watabe, Shiba, Chen, et al., 2021). These approaches involved encoding input feature vectors into quantum states to be used as the input for the neural network. Since the quantum models were trained to only predict univariate responses, it inspired the question of how to train a quantum neural network to predict multiple response variables.

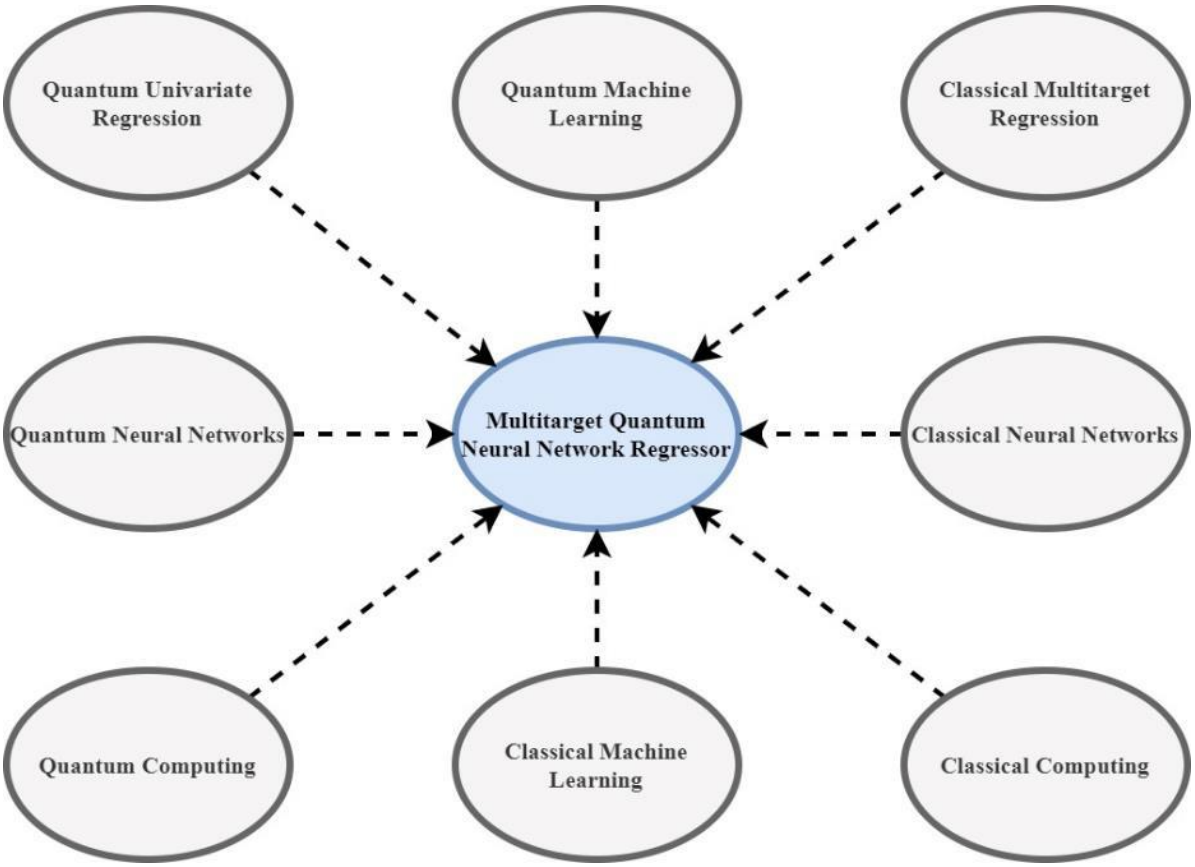
A survey of current literature revealed a gap in research on how to train a quantum neural network for multivariate multiple regression tasks. For this reason, the research involved constructing a multivariate multiple regression quantum neural network model using quantum and classical guidance in its design (Figure 1). This investigation helped answer how to create a multivariate multiple regression model using a quantum neural network to effectively predict multivariate responses.

Techniques for training classical neural network multivariate multiple regression models helped aid in the design of a quantum version of the model. The research adopted elements of classical neural networks combined with a quantum approach as a unified strategy for preparing and training a quantum neural network to predict multivariate responses. A multivariate regression classical neural network was also trained on the same dataset to compare the performance of the two models. The experiments helped further understand how to design a quantum neural network model for multivariate multiple regression problems. The results of the

experiments also helped answer how the performance of the quantum model compares to that of the classical model.

Figure 1

Conceptual Framework for a Multivariate Regression Quantum Neural Network



Introduction to Research Methodology and Design

This investigation applied the *design science research* approach (Hevner & Chatterjee, 2010; March & Smith, 1995; Peffers et al., 2007). Design science, also known as *constructive research* (Crnkovic, 2010), is the process of developing, extending, or applying solutions to a problem in an innovative way. This research resulted in the creation of a quantum neural network model for multivariate multiple regression problems through iterative experimentation and

testing. The final artifacts included strategies for training a quantum neural network model to predict multivariate responses and formal methods for evaluating the model's performance.

Although performance was assessed using quantitative metrics, the innovative nature of this research made the design science research approach a suitable methodology over a strictly quantitative one. The design science research methodology includes the flexibility of defining quantitative objectives, qualitative objectives, or a combination of both in the design of an artifact. The selected design science research approach, as suggested by Peffers et al. (2007), consisted of six steps shown in Table 1.

This investigation helped answer how to design a quantum neural network model for multivariate regression problems. This process involved training quantum neural networks using a dataset with multidimensional regression data. For designing and training the neural network, the Qiskit (Qiskit contributors, 2023) quantum software development kit was used to accomplish this task. This research also investigated the type of neural network most suitable for the task, such as an *EstimatorQNN* or a *SamplerQNN*, along with the optimal parameters used to train them. The neural network models were iteratively trained and refined for optimal performance. The goal of this investigation was to design an approach for creating a quantum neural network model for predicting multiple response variables along with techniques for assessing the model's performance. The research resulted in the construction of a new multitarget quantum neural network regression approach and the identification of the models that achieved the highest performance metrics.

Table 1

Design Science Research Steps

Step	Description
Problem Identification	An Investigation into Quantum Neural Networks for Multivariate Regression Problems
Definition of Objectives	Identify the type of Neural Network best suited for the task. Identify a strategy for creating a quantum neural network for multivariate multiple regression problems.
Design and Development	Establish a process for training a neural network using a quantum development toolkit.
Demonstration	Use the trained quantum neural network to predict multidimensional values.
Evaluation	Assess the statistical performance of the neural network using metrics such as MAE, RMSE, and R^2 scores. Compare the performance of a classical neural network and the quantum neural network using the same dataset.
Communication	Present the findings in a manuscript and oral dissertation defense.

The quantitative metrics included the mean absolute error (MAE), mean squared error (MSE), root mean squared error (RMSE), and r squared (R^2) score. The evaluation process also involved measuring and comparing the performance metrics of classical neural networks trained using the same dataset on classical hardware. The findings of the experiments are available in Chapter 4 along with the source code for the new Python module for training models on quantum simulators or quantum processors.

Research Questions

Due to recent advances made with quantum neural networks on near-term quantum processors, it is essential to identify the challenges in developing a quantum multivariate

multiple regression neural network model and measure its performance. This dissertation research answered the following research questions.

RQ1

How can a multivariate multiple regression model be created using a quantum neural network to predict multivariate responses effectively?

RQ2

How does the training and prediction performance of a quantum multivariate multiple regression neural network model compare to the performance of a classical one?

Answering RQ1 will help establish best practices for preparing training datasets, creating quantum circuits, and running the training algorithms in quantum simulators and quantum hardware. This iterative task involved refining the process by identifying the optimal hyperparameters, loss function, and programming procedures. The same actions were performed using a classical neural network to evaluate the performance of both models to help us answer RQ2.

Significance of the Study

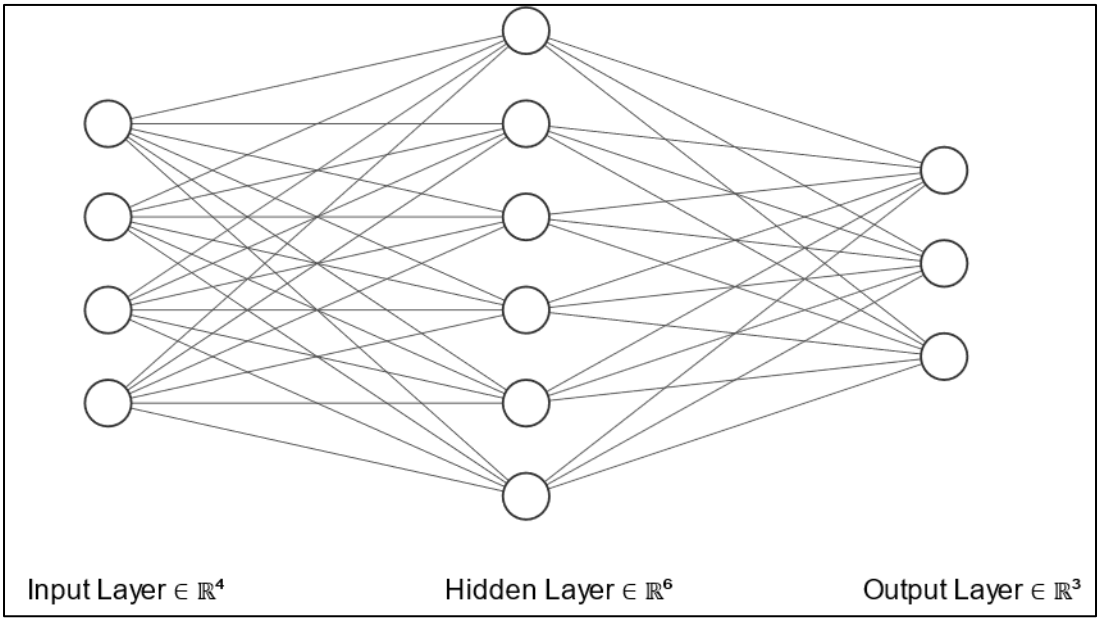
Multivariate regression is the task of simultaneously predicting multiple response variables from the same set of independent variables (Figure 2). Industry applications of multivariate regression include cancer research (Lee et al., 2012; Lee & Liu, 2012), agriculture (Johnson & Wichern, 2014), and aviation (Rencher & Christensen, 2012). Multiple approaches (Borchani et al., 2015; Breiman & Friedman, 1997; Johnson & Wichern, 2007) for creating multivariate regression models that attempt to explain the relationship between predictor variables and dependent variables have been put forward and continue to be a research topic of interest. Recent approaches using artificial neural networks to predict multivariate responses

have demonstrated favorable performance and higher accuracy than conventional approaches (Alahassa, 2021; Stangierski et al., 2019). Further research should explore using quantum neural networks for predicting multivariate responses.

This investigation addressed gaps in current research on the use of quantum neural networks for multivariate regression problems. Recent approaches (Kairon & Bhattacharyya, 2020; Quoc et al., 2021; Reddy & Bhattacherjee, 2021) focused on using quantum neural networks to predict univariate responses. The results of this investigation yielded techniques to effectively train future quantum neural networks to predict multiple response variables.

Figure 2

Multi-target regression neural network example



Definitions of Key Terms

Bloch Sphere

A geometric approach to visualizing the state of a single qubit (Nielsen & Chuang, 2010).

Bra-ket Notation

Also referred to as Dirac notation. Used for representing row vectors (*bra*) and column vectors (*ket*) of quantum states. Often represented using the kets $|0\rangle$ and $|1\rangle$ (Bernhardt, 2019).

Classical Computing

Refers to computation performed on ordinary computers using binary bits to represent the states of zero or one (Mermin, 2007).

Neural Network

Artificial neural network. A computational model inspired by the human brain programmed to recognize complex patterns and make intelligent decisions (Keijsers, 2010).

State Space

A linear space representing all the possible points a state vector can point to. Often represented by a Bloch sphere (Kitaev et al., 2002).

State Vector

A unit vector in a quantum system's state space (Nielsen & Chuang, 2010).

Quantum Circuit

An arrangement of wires and elementary quantum gates to carry and manipulate quantum information (Nielsen & Chuang, 2010).

Quantum Computing

Refers to the processing of information performed on quantum mechanical systems (Nielsen & Chuang, 2010).

Quantum Entanglement

Quantum states that cannot be factored into two product states (Wong, 2022).

Quantum Gate

Operations that change the states of a qubit (DiVincenzo, 1998).

Quantum Mechanics

The mathematical framework for constructing physical theories (Nielsen & Chuang, 2010).

Quantum Neural Network

An artificial neural network that combines elements of quantum theory with the properties of a traditional neural network (Schuld et al., 2014a).

Qubit

Quantum bit. The basic unit of quantum information with possible states of $|0\rangle$, $|1\rangle$, or a linear combination of both (Kitaev et al., 2002).

Superposition

The quantum state resulting from the linear combination of $|0\rangle$ and $|1\rangle$ (Mermin, 2007).

Wavefunction

Also known as the quantum state. Denoted by the uppercase or lowercase representation of the Greek letter psi: Ψ or ψ (Lundeen et al., 2011).

Summary

This chapter discussed the investigation on the construction and performance evaluation of a quantum neural network regression model for multivariate problems. The problem this investigation addressed was the gap in research on how to construct a quantum neural network architecture for predicting multiple targets in regression problems. This research extended

previous methods of building models for univariate regression problems by incorporating classical techniques of building multivariate neural network regressors. The answers to the research questions established in this chapter help guide future researchers in constructing multivariate regression models using quantum neural networks. The significance of this study is that it could have measurable impacts in the areas of aviation, agriculture, and cancer research, should future quantum processors prove to possess a quantum advantage. Finally, the definitions of key terms were provided to help the reader understand the quantum computational concepts discussed throughout the manuscript.

Chapter 2: Literature Review

The purpose of this study was to design a strategy for creating a multivariate multiple regression model using quantum neural networks. The investigation also provided formal methods for measuring the model's performance. This chapter reviews and discusses the literature leading up to the research topic. The path begins with an overview of multivariate multiple regression followed by an introduction to quantum computing. The chapter concludes with a review of related works and a discussion on the research gaps that inspired the research questions.

Literature Search Strategies

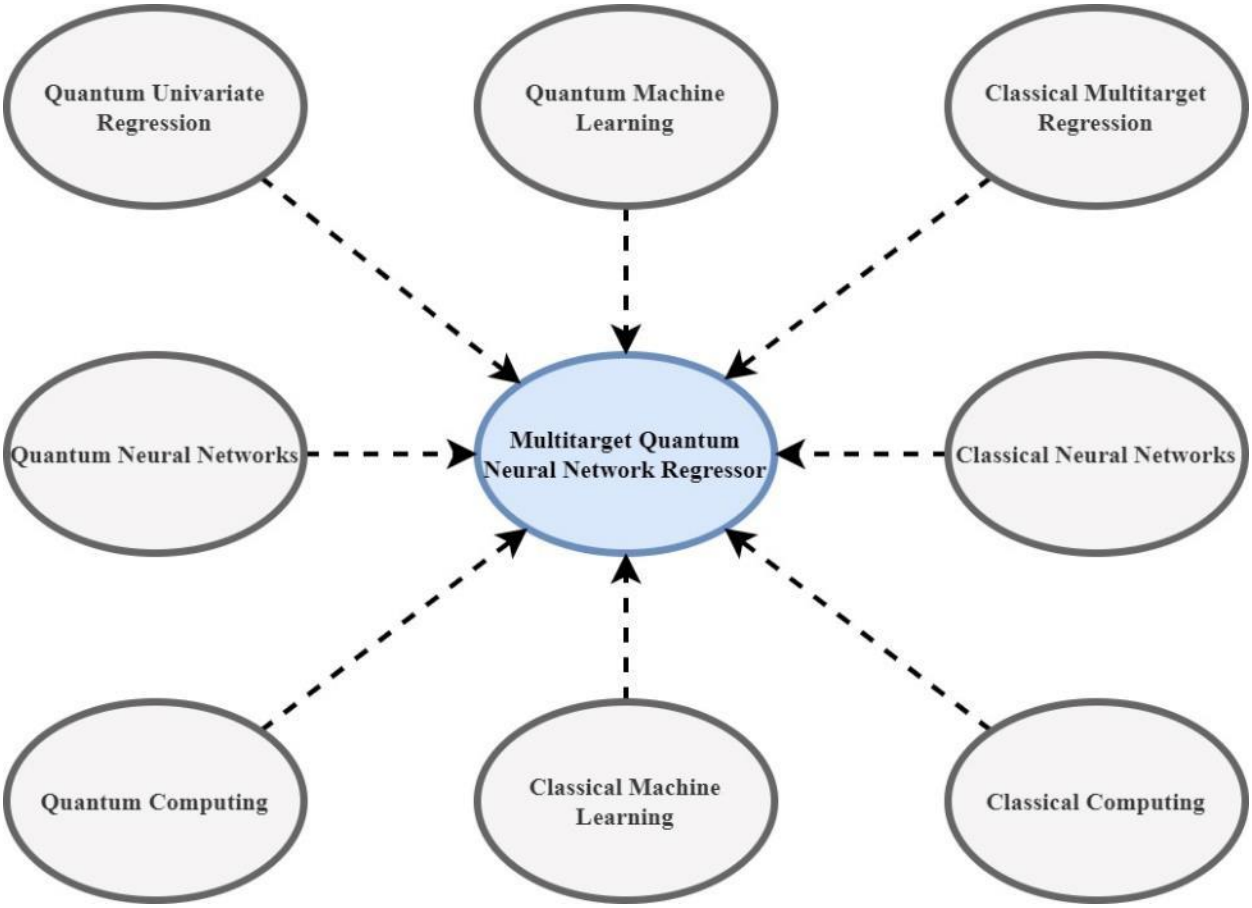
The databases searched for this investigation included IEEE Xplore, ACM Digital Library, the Cornell University Library's ArXiv website, Google Scholar, and the EBSCOHost research platform. The search terms used were *multivariate multiple regression, quantum neural network, quantum multivariate regression, neural network regression, multilayer perceptron regressor, and quantum machine learning*. The literature review also included references to several technical books covering statistical concepts, quantum mechanics, quantum information science, and quantum computing.

Conceptual Framework

The conceptual framework in Figure 3 depicts the combination of quantum and classical machine learning concepts used in the design of the new multiple target quantum neural network regression model. This investigation explored the use of the Qiskit's EstimatorQNN and SamplerQNN (Chen, 2023; Markidis, 2023; J. Stein et al., 2023; Tehrani et al., 2023) for the underlying quantum neural networks in the new model. The next section reviews concepts of regression analysis using conventional approaches and classical neural networks.

Figure 3

Conceptual Framework for a Multivariate Regression Quantum Neural Network



Regression Analysis

Simple regression is the task of estimating the relationship between a single variable, Y (dependent variable), and a single predictor variable, X (independent variable). Multiple regression is the task of establishing a relationship between a single dependent variable Y and multiple independent variables X_1, $X_{2,}$…, X_n (Rencher & Christensen, 2012). The concept central to this research is multivariate multiple regression, which is the task of estimating the relationship between multiple response variables $Y_{1,}Y_{2,}$…,Y_n and a set of prediction variables X_1, X_2 ,…,X_n (Xu et al., 2019). Regression analysis can be linear or non-linear. Linear regression models often use a least-squares approach to estimate the relationship. Non-linear regression models use exponential functions, logarithmic functions, Gaussian functions, and other fitting methods to estimate the relationship between the dependent and independent variables.

Multivariate Multiple Regression

Multivariate regression is also referred to as multiple-target, multi-response, multi-objective, or multi-output regression and can be approached with problem transformation techniques, algorithm adaptation methods, statistical methods, or machine learning techniques (Borchani et al., 2015). Industry applications of multivariate regression include cancer research (Lee et al., 2012; Lee & Liu, 2012), agriculture (Johnson & Wichern, 2014), and aviation (Rencher & Christensen, 2012). In the defense industry, the multivariate regression problem is central to computing the launch acceptability region of a guided bomb unit in air-to-surface attacks (Filgöz et al., 2021; Kang, 2021; Yoon et al., 2010), and air-to-air attacks (Birkmire & Gallagher, 2012). The vast applications of multivariate multiple regression underscore the importance of making advances in finding better techniques or optimizing existing ones.

Researchers have presented numerous approaches for creating multivariate regression models that attempt to explain the relationship between predictor variables and the dependent variables. This topic continues to be an active research area across different domains. Table 2 contains a summary of the different approaches for multivariate multiple regression discussed in this section.

The original approach to the multivariate multiple regression problem was creating a separate regression model for each response variable (Johnson & Wichern, 2007). Another conventional approach was to perform a least squares regression (Abdi & Williams, 2013; Höskuldsson, 1988; Rencher & Christensen, 2012; H. Wold, 1975; S. Wold et al., 1984). To challenge the conventional approach of not combining multiple responses, the Curds and Whey method (Breiman & Friedman, 1997) was introduced and found to improve predictive accuracy, especially when the response variables were correlated.

Another approach, multivariate adaptive regression splines (Friedman, 1991), also challenged conventional approaches and continues to be used in present day research (Akin et al., 2020; Bera & Mukherjee, 2018; Gerling et al., 2021; Sun et al., 2021). Rimal et al. (2019) provided a comparison of additional statistical approaches including, partial least squares (Helland, 1990), principal component regression (Jolliffe, 2002), and simultaneous envelopes (Cook & Zhang, 2015). Their analysis concluded that the simultaneous envelopes approach outperformed the partial least squares and principal component regression approaches.

Table 2

Multivariate Multiple Regression Approaches

Method	Relevant Works
Partial Least Squares Regression	Abdi & Williams, 2013 Helland, 1990 Höskuldsson, 1988 Rencher & Christensen, 2012 H. Wold, 1975 S. Wold et al., 1984
Linear Regression	Johnson & Wichern, 2007 Rencher & Christensen, 2012
Curds and Whey	Breiman & Friedman, 1997
Multivariate Regression Splines	Akin et al., 2020 Bera & Mukherjee, 2018 Friedman, 1991 Gerling et al., 2021 Sun et al., 2021
Principal Component Regression	Jolliffe, 2002
Simultaneous Envelopes	Cook & Zhang, 2015
Artificial Neural Networks	Alahassa, 2021 Bravo & Moreno, 2019 Moosavi et al., 2017 Stangierski et al., 2019

Borchani et al. (2015) suggested that approaches for multivariate multiple regression problems can be categorized as a problem transformation technique or as an algorithm adaptation method. The problem transformation techniques identified included single target regression, random linear target combinations, separate ridge regression, multi-target regressor stacking, regressor chains, and multioutput support vector regression. Among the algorithm adaptation techniques were statistical methods, other multioutput support vector regression methods, kernel methods, multitarget regression trees, and rule methods. The problem transformation techniques performed less favorably than the algorithm adaptation methods since some of the transformation algorithms failed to exploit the relationships between the target variables. Borchani et al. arrived at the same conclusion as Breiman and Friedman (1997), and found that exploiting the correlation between the output variables can improve the predictive accuracy as well as reduce computational costs.

Recent approaches using artificial neural networks to predict multivariate responses have demonstrated favorable performance and higher accuracy than conventional approaches (Alahassa, 2021; Bravo & Moreno, 2019; Moosavi et al., 2017; Stangierski et al., 2019). Experiments performed by Stangierski et al. (2019) found that the artificial neural networks resulted in higher determination coefficients and lower root mean squared error (RMSE) values. Alahassa (2021) established that neural networks perform better than conventional approaches when there are nonlinearities in the multivariate multiple regression problem.

Classical Neural Networks

A neural network is a two-stage nonlinear statistical model suitable for regression or classification problems (Hastie et al., 2009). Other names for a neural network are hidden layer back-propagation network, single layer perceptron, or a multilayer perceptron. Neural networks

attempt to mimic the human brain and are represented as a network-like structure with artificial synapses containing associated weights.

A typical neural network configuration has an input layer with nodes for each independent variable, one or multiple hidden layers for data processing, and an output layer with nodes for each dependent variable (Carvalho et al., 2013). The independent variables, also called the input features, are first passed into an input function to compute their weighted sum. These results are then passed on to an activation function, usually a sigmoid function (Hastie et al., 2009), to generate the results that get passed to the next layer until they reach the output layer (Menzies et al., 2015). This iterative process gets repeated and uses the backpropagation algorithm (Rumelhart et al., 1986) to adjust the weights assigned to each input feature with the goal of minimizing errors and not overfitting the training data. The multilayer perceptron regressor used by Bravo and Moreno (2019) is available as the *MLPRegressor* class in the Sci-kit Learn library (Pedregosa et al., 2011) and provides methods for optimizing the model and avoiding overfitting. The MLPRegressor class provides three optimization algorithms to solve for the weights: the *limited-memory Broyden-Fletcher-Goldfarb-Shanno* (LBFGS) method (Liu & Nocedal, 1989), the *stochastic gradient descent* (SGD) method (Robbins & Monro, 1951), and the *Adam* optimization method (Kingma & Lei, 2015). The training and optimization techniques of the MLPRegressor class were used as guidance in the design and optimization of the quantum neural network regressor presented in this manuscript.

The advantages of neural networks naturally inspired the quest for a quantum neural network (Kak, 1995; Vlasov, 1997). Multiple researchers have developed models for a quantum perceptron (Altaisky, 2001; Lewenstein, 1994; Siomau, 2014) and the requirements for designing a meaningful quantum neural network (Schuld et al., 2014b). The next section includes an

introduction to quantum computing and a review of current work involving quantum neural networks.

Introduction to Quantum Computing

Quantum computers were initially submitted as a potential solution for simulating quantum systems (Feynman, 1982, 1986). Near-term quantum computers attempt to achieve exponential speed-ups in computation by exploiting the laws of quantum mechanics (Bozzo-Rey et al., 2019). Theoretically, all computations that can be performed on a classical computer can also be done on a quantum computer (Bernhardt, 2019). This new era of computing presents opportunities for advancements in optimization, simulations, artificial intelligence, chemistry, finance, cryptography, cybersecurity, manufacturing, and medical sciences.

Some of the pioneering works in quantum computing started with a paper describing a quantum Turing machine that leverages quantum parallelism to perform faster computations (Deutsch & Penrose, 1985). Shortly after, the first designs and notations of quantum circuits were introduced (Feynman, 1986). This paved the way for the design of numerous quantum algorithms including the *Deutsch-Josza algorithm* (Deutsch & Jozsa, 1992), one of the first examples of a quantum deterministic algorithm that is exponentially faster than a classical version; *Shor's algorithm* (Shor, 1994) for finding prime factors of an integer at a polynomial speedup by using quantum gates; *Grover's algorithm* (Grover, 1996), which provides a quadratic speedup over classical search algorithms in performing an unstructured search; and many others that have been indexed on the *Quantum Algorithm Zoo* website (Jordan, 2022). The next section discusses the growing interest in quantum computing and initiatives to explore its potential applications in industry.

Quantum Computing Initiatives

Quantum computing has attracted interest from multiple industries and most recently received a considerable public investment as a result of the National Quantum Initiative Act, passed unanimously in 2018 by both houses of the United States Congress (Raymer & Monroe, 2019). The emergence of viable and efficient quantum computers has increased the calls for quantum-resistant cryptography and recently resulted in four candidate algorithms that would meet the definition set by the National Institute of Standards and Technology (NIST, 2022). Quantum computing has also received attention from the NASA (Biswas et al., 2017; Guillaume et al., 2022; Rieffel et al., 2019), the Department of Energy (Awschalom et al., 2021; Ho et al., 2018), and numerous agencies under the Department of Defense (J. E. Christensen et al., 2020; Keeler, 2022; Koch et al., 2021; Malhotra, 2020; Savchenkov et al., 2020; Wolf et al., 2019). The financial services industry is also exploring quantum applications in portfolio management, credit scoring, risk modeling, and derivative pricing, and is expected to exceed $630 million in expenditures by 2028 (IQT, 2022). The next section discusses investment in quantum computing and the key investors involved in its expansion.

Investment in Quantum Computing

Over 600 technology companies have invested in the development of quantum computing hardware, software, and cloud-based solutions for researchers and commercial clients (Dargan, 2022). While not an exhaustive list, key contributors are summarized in Table 3. The major technology companies invested in both quantum hardware and software solutions are IBM, Google, Microsoft, and Amazon.

IBM’s quantum services include cloud-based access to its 27-qubit, 65-qubit, and 127-qubit quantum computers for commercial customers, and 5-qubit and 7-qubit quantum computers

for researchers and enthusiasts (IBM, 2022). Google's quantum computer is not currently available to the public (Google, 2022a), the company has however made the 11-qubit IonQ quantum computer available through its cloud services, as well as a quantum virtual machine that can closely simulate the results that their Sycamore quantum computer would produce (Google, 2022b). Microsoft currently provides cloud-based access to third-party quantum computers via its Azure service. These include Quantinuum, IonQ, Quantum Circuits, Inc., Rigetti, and PASQAL (Microsoft, 2022). Amazon's cloud-based AWS Braket service provides access to third-party quantum computers from D-Wave, IonQ, Rigetti, Oxford Quantum Circuits, and Xanadu (Amazon, 2022). Most of the service providers allow simulating quantum circuits so that users can approximate the expected runtime on quantum hardware.

Table 3

Quantum Computing Companies

Company	Services and Products
IBM	• Quantum hardware • Quantum software development kit • Cloud-based access to proprietary quantum hardware • Quantum simulator
Google	• Quantum hardware • Quantum software development kit • Cloud-based access to third-party quantum hardware • Quantum Virtual Machines
Microsoft	• Quantum software development kit • Cloud-based access to third-party quantum hardware • Quantum Simulator
Amazon	• Quantum software development kit • Cloud-based access to third-party quantum hardware • Quantum Simulator
Intel	• Quantum software development kit • Quantum Simulator
D-Wave	• Quantum Hardware • Cloud-based access to proprietary quantum hardware • Quantum SDK
IonQ	• Quantum Hardware • Cloud-based access to proprietary quantum hardware

	• Quantum SDK
Pasqal	• Quantum Hardware • Cloud-based access to proprietary quantum hardware
Quantum Circuits Inc.	• Quantum Hardware • Cloud-based access through Azure
Oxford Quantum Circuits	• Quantum Hardware • Cloud-based access to proprietary quantum hardware
Rigetti	• Quantum Hardware • Cloud-based access to proprietary quantum hardware • Quantum SDK • Quantum Virtual Machine
Xanadu	• Quantum Hardware • Cloud-based access to proprietary quantum hardware • Quantum SDKs • Quantum Simulators

Quantum Computing Concepts

This section provides an overview of the key quantum computing concepts central to the research proposal. The concepts covered include *superposition*, *quantum gates*, *quantum circuits*, *quantum entanglement*, *quantum encoding*, and *quantum error correction*. Following this section is a review of current research on quantum machine learning and related works.

Superposition

One of the key features that makes quantum computing advantageous is the concept of superposition. In a classical computer, the basic units of information are called bits and represent the possible states of 0 or 1. In quantum computing, quantum bits (or qubits) can be represented by the special state vectors of $|0\rangle$, $|1\rangle$, or a linear combination of both, referred to as a superposition (Nielsen & Chuang, 2010). The state of a qubit $|\psi\rangle$ is a unit vector in a two-dimensional complex vector space. The state vectors $|0\rangle$ and $|1\rangle$ are known as computational basis states and form the orthonormal basis for the vector space. Using complex coefficients, a linear combination of the basis states can be expressed as so:

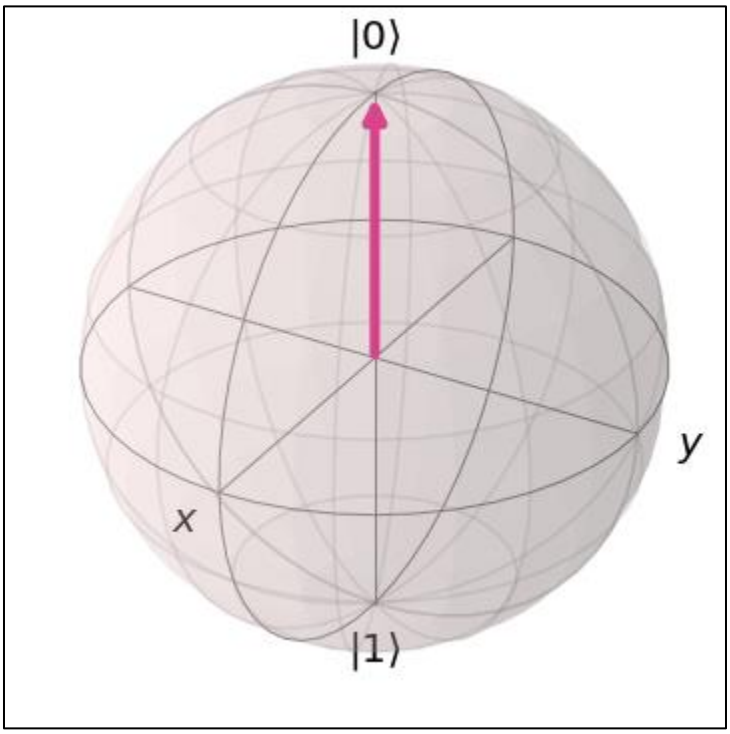
$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

The complex coefficients α and β are specifically called the probability amplitudes and require $|\psi\rangle$ to be a unit vector in the complex vector space (Mermin, 2007). Therefore, the sums of the probabilities $|\alpha|^2 + |\beta|^2$ must equal 1 (Nielsen & Chuang, 2010). The qubit unit vector can be geometrically represented using a Bloch sphere (Figure 4). Since there are an infinite number of points on the sphere, it can help the reader imagine how a qubit can be in an infinite number of states while in a superposition. However, once the qubit is measured, its wavefunction collapses so that its state can only be $|0\rangle$ or $|1\rangle$ (Nielsen & Chuang, 2010). Measuring the qubit causes its wavefunction to collapse for reasons that remain unknown (Griffiths & Schroeter,

2018). Ergo, measurements on quantum computers should only be used when an output needs to be extracted, often at the end of a quantum circuit.

Figure 4

Bloch Sphere



Physical Quantum Computers

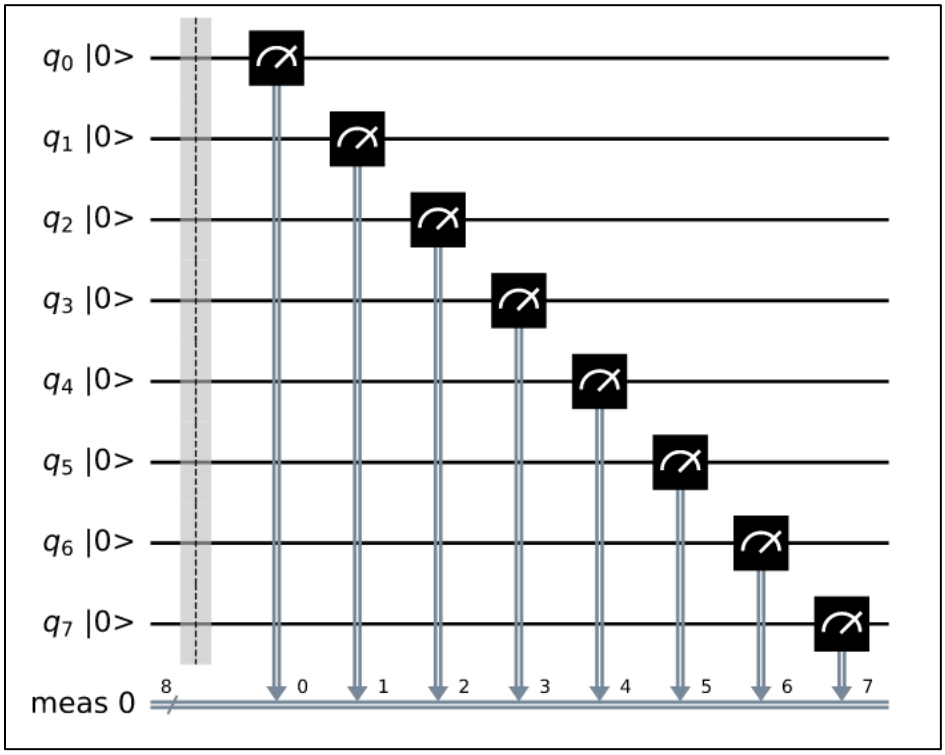
Any physical quantum system with two distinct states can be used to represent a qubit (Wong, 2022). Examples of physical quantum systems include ones that use photons, trapped ions, cold atoms, nuclear magnetic resonance, quantum dots (electrons), defect qubits, or superconductors. Many companies, such as IBM and Google, have invested in quantum computers that use superconducting qubits (Bernhardt, 2019). Cloud services provided by those companies have made physical quantum computers available to researchers and commercial clients.

Quantum Gates and Circuits

Like classical computers, quantum computation is performed using circuits composed of logic gates. Quantum gates are operations that can be described by orthogonal matrices (Bernhardt, 2019). The inputs and outputs of a one-qubit quantum gate are discrete quantum variables, and the operations are rotations around the Bloch sphere. Quantum circuits (Figure 5) provide the advantage of evaluating a function $f(x)$ for many different values of x simultaneously (Nielsen & Chuang, 2010). To achieve classical parallelism, multiple circuits are executed simultaneously to perform the same operation. With quantum parallelism, a single quantum circuit can be used to evaluate multiple values simultaneously. Quantum gates can also perform operations on two qubits at the same time. One of the most essential quantum gates involving two qubits is the CNOT (controlled-NOT) gate, which inverts the target (right side) qubit if the control (left side) qubit is 1 (Wong, 2022). Using two or more qubits also allows building more complex gates to build quantum algorithms or leverage quantum entanglement.

Figure 5

Example Quantum Circuit with 8 Qubits



Quantum Entanglement

Quantum entanglement plays a key role in the most interesting applications of quantum computing (Nielsen & Chuang, 2010). A group of qubits whose states cannot be independently described are said to be entangled. Measuring one of the entangled qubits instantaneously tells us the state of the other qubit and collapses the superposition regardless of the distance between the qubits (Wong, 2022). Multiple qubits can be entangled, as demonstrated in recent experiments involving six entangled qubits (Zhong et al., 2021). Such a configuration can pave the way for quantum communication networks. The CNOT gate is also referred to as the entangling gate, and is used for entangling two unentangled qubits (Bernhardt, 2019). This concept is important to know since it plays a role in many quantum circuits.

Quantum Encoding and Error Correction

To work with quantum computers, classical information must be encoded into quantum states (Park et al., 2019). This is made possible with quantum circuits to encode classical data into qubits. A problem that arises when executing quantum circuits is interference (noise) from the environment. It is common practice to execute quantum circuits thousands of times (shots) to get a probability distribution (Anaya-Morales & Delgado, 2023). For this reason, the current period of quantum computing is referred to as the Noisy Intermediate Scale Quantum (NISQ) era due to limitations on the size of quantum circuits that can be executed reliably (Preskill, 2018). At present, state-of-the-art quantum systems produce noise at an error rate above 1% on superconducting qubits, thus reducing the quality of the qubits. These errors result from quantum decoherence, where the environment interacts with the qubits and alters their states. To mitigate this problem, quantum error correction attempts to preserve the encoded information by constantly running the physical qubits through a highly entangling circuit (DiVincenzo, 1998; Zhong et al., 2021). Error correction becomes more difficult as the size of the quantum system grows and continues to be one of the prohibitive factors in creating large-scale quantum systems. Researchers recently suggested that the key to building large-scale quantum systems may involve dynamically shuffling qubits in the system to make them highly entangled and thus self-correcting (Bluvstein et al., 2022). In the meantime, designing algorithms during the NISQ era helps prepare researchers to run them on fault-tolerant systems in the future.

Quantum Software Development Kits

The rise of quantum computers has necessitated moving beyond mathematical equations to computer code that can run on quantum computers. A considerable number of quantum development kits have emerged in recent years and are summarized in Table 4**.** Among the most

popular quantum development kits are Qiskit, Cirq, PyQuil, and Microsoft's Quantum SDK (Carrascal et al., 2021). This investigation used IBM's Qiskit SDK (Qiskit contributors, 2023) and quantum simulators running on enterprise-grade cloud servers.

Table 4

Quantum Software Development Kits

Quantum SDK	Supported Programming Languages	Provider
Qiskit	Python and OpenQASM	IBM
Cirq	Python	Google
Quantum Development Kit	Q#, C#, F#, Python	Microsoft
Braket	Python	Amazon
PyQuil	Python	Rigetti
NVIDIA cuQuantum	C and Python	NVIDIA
Forest	Python	Rigetti
Strawberry Fields	Python	Xanadu
Pennylane	Python	Xanadu
Intel Quantum SDK	C++	Intel
Quantum Inspire SDK	Python	QuTech
Ocean	Python	D-Wave
Torch Quantum	Python	PyTorch
QuTiP	Python	QuTiP
Blueqat	Python	Blueqat
Mitiq	Python	Unitary Fund
Orquestra	Python	Zapata
Paddle Quantum	Python	Baidu
Pytket	Python	Quantinuum
Qrack	C++	Unitary Fund
Quantum++	C++	SoftwareQ
TensorFlow Quantum	Python	TensorFlow
Qulacs	C++ and Python	Qulacs
Qibo	Python	Qibo

Quantum Machine Learning

Quantum machine learning is a research area that incorporates ideas from quantum computing and classical machine learning (Wiebe, 2020). The goal is to eventually run machine learning algorithms on quantum processors to improve the computational speed of the algorithms. Classical machine learning algorithms that have been implemented on quantum computers include k-nearest neighbors, support vector machines, and k-means (Schuld et al., 2015). The next section discusses additional quantum algorithms and a software development kit (SDK) that implements them.

Quantum versions of neural networks (Beer et al., 2020; Kamruzzaman et al., 2020; Schuld et al., 2014b), Boltzmann machines (Amin et al., 2018), and generative adversarial networks (Huang et al., 2020; Lloyd & Weedbrook, 2018; Niu et al., 2021b) have also been implemented. The IBM Qiskit SDK (Qiskit contributors, 2023) provides quantum neural network classes for classification and regression tasks. The aim of the research was to leverage the quantum neural network regressor class to build a multivariate multiple regression model.

Quantum Neural Networks

Following the initial proposal for quantum computers and the design of the first quantum algorithms, designs for a quantum neural network were introduced not too long after (Kak, 1995; Vlasov, 1997). Multiple researchers designed models for a quantum perceptron (Altaisky, 2001; Lewenstein, 1994; Siomau, 2014) and the requirements for designing a meaningful quantum neural network (Schuld et al., 2014b). A modified definition of a quantum perceptron has also been presented to allow for the construction of a deep quantum neural network (Beer et al., 2020). The next section discusses emerging applications of quantum neural networks in various domains.

In recent years, quantum neural networks have been applied to problems involving disease prediction, image processing, and speech recognition (R. Zhao & Wang, 2021). The Qiskit library (Qiskit contributors, 2023) provides two quantum neural networks, the EstimatorQNN, based on the evaluation of quantum mechanical observables, and the SamplerQNN, based on the samples that result from measuring a quantum circuit. Recent experiments using the EstimatorQNN and SamplerQNN include applications in deep learning (Chen, 2023), cybersecurity (Tehrani et al., 2023), and pharmaceutical development (J. Stein et al., 2023). Expansions to existing machine learning libraries, such as TensorFlow, have also made it possible to extend its features to support quantum neural networks (Pattanayak, 2021). These findings suggest that a shift in machine learning from classical computers to quantum computers is imminent and requires more research to uncover potential use cases.

The EstimatorQNN and SamplerQNN quantum neural network classes provide different optimization algorithm choices to solve for the weights. This investigation tested the *LBFGSB* method (Zhu et al., 1997), which is an extension of the LBFGS method improved to deal with bounds on variables, the *Adam* optimization method (Kingma & Lei, 2015), the *constrained optimization by linear approximation* (COBYLA) method (Powell, 1994), and the *sequential least squares programming* (SLSQP) method (Kraft, 1988) during the training of the quantum neural networks. The classes also provide the flexibility of encoding the training inputs, i.e. creating the *feature map*, using a choice between a standard quantum circuit (QuantumCircuit), a Pauli expansion circuit, a first-order Pauli-Z evolution circuit (ZFeatureMap), or a second-order Pauli-Z evolution (ZZFeatureMap) circuit (Havlicek et al., 2019). The feature mapping process enables the encoding of the classical data as qubits. The Qiskit library provides a mechanism for assigning tunable weights to each of the features in the feature map. The optimization algorithms

optimize these weights to minimize the objective function, such as the mean squared error or the mean absolute error when dealing with regression problems.

The weights for each feature are represented by a parameterized quantum circuit, also known as an *ansatz*. The Qiskit circuits that can be used for the ansatz include the NLocal, TwoLocal, RealAmplitudes, and the EfficientSU2 circuits. These circuits include the capability of generating entangled states, which can provide the advantage of efficiently representing the solution space and capturing correlations in quantum data (Sim et al., 2019). This investigation used the EstimatorQNN for the quantum neural network, a standard quantum circuit for the feature map, and the RealAmplitudes circuit for the ansatz.

Related Works

This section includes a review of related works surrounding linear regression, nonlinear regression, and univariate regression using quantum machine learning. Several techniques include quantum linear regression (Quoc et al., 2021), quantum support vector regression (Dalal et al., 2021), variational quantum regression (Kairon & Bhattacharyya, 2020), variational quantum linear solver (Chakraborty et al., 2018; Tilly et al., 2022), the Harrow Hassidim Lloyd (HHL) algorithm (Dutta et al., 2020; Harrow et al., 2009), the quantum sampling regression algorithm (Rivero et al., 2020), a quantum backpropagation multilayer perceptron (Bhattacharyya et al., 2015), a quantum backpropagation algorithm for circuit-based learning (Watabe, Shiba, Chen, et al., 2021; Watabe, Shiba, Sogabe, et al., 2021), and a single hidden layer quantum neural network (Macaluso et al., 2020a). A summary of these approaches is available in Table 5.

Table 5

Quantum Regression Approaches

Methods	Relevant Works
Quantum Linear Regression	Quoc et al., 2021
Quantum Support Vector Regression	Dalal et al., 2021
Variational Quantum Regression	Kairon & Bhattacharyya, 2020
Variational Quantum Linear Solver	Chakraborty et al., 2018 Tilly et al., 2022
Harrow Hassidim Lloyd (HHL) Algorithm	Dutta et al., 2020 Harrow et al., 2009
Quantum Backpropagation Multilayer Perceptron	Bhattacharyya et al., 2015
Quantum Backpropagation Algorithm for Circuit-Based Learning	Watabe, Shiba, Chen, et al., 2021 Watabe, Shiba, Sogabe, et al., 2021
Single Hidden Layer Quantum Neural Network	Macaluso et al., 2020a
Quantum Sampling Regression Algorithm	Rivero et al., 2020

Quantum Neural Networks for Regression

Quantum neural networks have been successfully used to create regression models that predict univariate responses (Bhattacharyya et al., 2015; Reddy & Bhattacherjee, 2021; Watabe, Shiba, Chen, et al., 2021). A survey of the current literature revealed a gap in research on how to train a quantum neural network for multivariate multiple regression tasks. Current techniques involve training quantum neural networks to predict a single dependent variable given one or more independent variables. The Qiskit library provides the NeuralNetworkRegressor class, which can accept an EstimatorQNN or a SamplerQNN, to help perform regression tasks (Markidis, 2023). Current literature and tutorials to-date have only demonstrated how to use the NeuralNetworkRegressor to predict single targets. This investigation used the EstimatorQNN and the NeuralNetworkRegressor classes in the design of a new artifact capable of predicting multiple response variables.

Summary

The advances made in the quantum computing domain over the last decade have significant implications for the current challenges hindered by the limitations of classical computers. Quantum computers with hundreds or thousands of qubits are expected to bring about a new era in areas such as cryptography, chemistry, physics, simulation, optimization, secure communications, and machine learning (Rieffel et al., 2019). This chapter reviewed the quantum mechanical ideas of superposition and entanglement that make the exponential computing speed-ups theoretically possible (Bernhardt, 2019). Quantum machine learning and quantum neural networks are popular research topics made possible by the availability of quantum programming languages and quantum software development kits. Implementations of quantum neural networks for regression tasks have been, thus far, used for predicting univariate responses. The literature

reviewed in this chapter revealed a gap in current research addressing how to train a quantum neural network to predict multivariate responses. The research extended current techniques for training quantum neural networks by developing a comprehensive approach for training a model to predict multiple response variables. The next chapter presents the research method and design of the investigation along with the evaluation procedures and performance metrics.

Chapter 3: Research Method

This chapter discusses the research method and design of the investigation. Using the design science research methodology, the contributions of this investigation included a strategy on how to train a multivariate multiple regression model using quantum neural networks and how to assess the performance of the model. This research leveraged a quantum neural network class provided by the Qiskit quantum software development kit as the foundation for the new regression model. The EstimatorQNN and the NeuralNetworkRegressor classes were adopted in the design of the new artifact.

The problem this investigation addressed was the lack of understanding on how to train a multivariate multiple regression model using quantum neural networks to predict multivariate responses effectively. A review of current literature found limited research on the usage of quantum neural networks for multivariate multiple regression problems. The purpose of this investigation was to design a quantum neural network model capable of predicting multivariate responses. The results of the research provided a strategy on using quantum neural networks for multivariate multiple regression problems, along with techniques for evaluating the performance of the model.

Research Methodology and Design

For this research, the design science research approach (Hevner & Chatterjee, 2010; March & Smith, 1995; Peffers et al., 2007) was adopted. Design science, also referred to as constructive research (Crnkovic, 2010), is the process of developing, extending, or applying solutions to a problem in an innovative way. The research study involved creating a quantum neural network model for multivariate multiple regression problems through iterative experimentation and testing. The artifacts included a new Python module that researchers can use

for training a quantum neural network model to predict multivariate responses along with formal methods for evaluating the model's performance.

Research Method Selection

Denzin and Lincoln (1994) defined qualitative research as an interpretive research method for investigating phenomena in their natural settings. Such research involves the use of case studies, personal experiences, interviews, observations, and other materials that can help provide a better understanding of the subject under investigation. The nature of qualitative research heavily relies on the subjective perspectives of the participants in a study (L. B. Christensen et al., 2014). This research study is an unbiased and objective investigation involving no human participants. Therefore, a qualitative research method is not appropriate for this investigation.

Quantitative research is defined as a research method that collects numerical data to answer a research question (L. B. Christensen et al., 2014). A quantitative study attempts to establish a causal relationship between independent and dependent variables. This research study had quantitative elements, but it did not solely involve investigating a causal relationship; it involved designing artifacts that were applicable to research or practice. Design science research provides a framework for designing an artifact with quantitative objectives (Hevner & Chatterjee, 2010). The quantitative objectives of this study involved maximizing the performance metrics of the trained quantum neural network models. The artifacts included a new Python module for training a multivariate regression quantum neural network model and methods for evaluating the performance of trained models. For this reason, the design science research approach was used instead of a strictly quantitative approach.

The innovative nature of this research made the design science research approach an appropriate methodology for creating artifacts with quantitative objectives. The design science research methodology includes the flexibility of defining quantitative objectives, qualitative objectives, or a combination of both in the design of an artifact. The selected design science research approach, as suggested by Peffers et al. (2007), consists of six steps:

1. Problem identification and motivation.
2. Definition of the objectives for a solution.
3. Design and development.
4. Demonstration.
5. Evaluation.
6. Communication.

The first step in the design science research approach was defining the research problem and justifying the value of developing a solution (Hevner & Chatterjee, 2010; Peffers et al., 2007). The problem this investigation addressed was the lack of guidance available to researchers on how to train a multivariate multiple regression model using quantum neural networks to effectively predict multivariate responses. Since quantum computers are predicted to have a computational advantage over classical computers (Bozzo-Rey et al., 2019) and have recently demonstrated examples of quantum supremacy (Arute et al., 2019), the value this research offers is a framework for using the new artifact for practical uses in the real world.

The second step of the design science research approach was defining the objectives of the solution. The objectives of the research were to answer how to best train quantum neural networks for multivariate regression problems and how to measure a trained model's performance.

The third step of the design science research approach was designing and developing the new artifact. For designing and training the neural network, the Qiskit (Qiskit contributors, 2023) quantum software development kit was used to accomplish this task. The quantum neural network models were trained using a dataset with multidimensional regression data and were iteratively trained and refined for optimal performance.

The fourth step was to demonstrate the use of the artifact through experimentation, simulation, case study, proof, or other activities (Hevner & Chatterjee, 2010). Through experimentation, the quantum neural network models were each used to predict multivariate responses and iteratively retrained to improve their performance.

For the fifth step, each neural network model was evaluated using descriptive statistics such as the mean absolute error, mean squared error, root mean squared error, and the r-squared metric, discussed in further detail in the data analysis section.

The sixth, and final step was communicating the problem, the artifact, its design, and its effectiveness. This information was included in Chapter 4 of this manuscript following the experiments and data analyses.

Materials or Instrumentation

The experiments of this investigation were performed on multiple systems. The classical neural networks and quantum neural networks were first designed on a computer with an Intel Core i7 2.60 GHz 6-core CPU and 16 GB RAM. The classical neural network and the simulated quantum neural networks were then trained and tested on an enterprise-grade cloud server with an Intel Xeon Platinum 3.70 GHz 16-core CPU and 32 GB RAM. The source code for these experiments was written with the Python programming language and the Scikit-learn (Pedregosa

et al., 2011) machine learning module and the Qiskit (Qiskit contributors, 2023) quantum software development kit.

Study Procedures

The first question this investigation aimed to answer was: How can a multivariate multiple regression model be created using a quantum neural network to effectively predict multivariate responses? To answer this question, the Python programming language and the Qiskit quantum software development kit were leveraged to implement the quantum neural network model. The investigation's variables included the neural network architecture and its hyperparameters as the independent variables and the performance scores, such as the mean absolute error (MAE), mean squared error (MSE), root mean squared error (RMSE), and r-squared (R^2) score as the dependent variables, discussed in further detail in the data analysis section.

The study procedures were divided into three experiments. The first experiment involved generating a synthetic multidimensional regression dataset using Scikit-Learn, training and optimizing the classical neural network models, and collecting the performance metrics for each model. The second experiment involved constructing a new artifact: a Python module that can be used by researchers to construct a multiple target regression model using quantum neural networks from the Qiskit library. In addition, the second experiment explored the automatic generation of feature maps, ansatz circuits, and methods for assessing a trained model's performance. The final experiment involved using the synthetic multidimensional regression data to validate the new multiple target quantum neural network class, train and optimize different models, and collect the performance metrics for each model.

Experiment 1: Training a Multiple Target Classical Neural Network Regression Model

The first step in this investigation was generating synthetic training datasets containing multidimensional regression data. The Scikit-learn (Pedregosa et al., 2011) machine learning module provides a synthetic dataset generation submodule where the researcher can specify the number of samples, features, and targets. The generated datasets were used for training the classical and the quantum neural network models to establish a basis for comparison. The Qiskit documentation recommends normalizing the features by scaling the data into an interval between 0 and 1 for better numerical stability and convergence of an algorithm. Therefore, the datasets were scaled down to this interval for all classical and quantum experiments. This investigation explored the effects of different sample sizes for the training datasets and thus used three generated datasets with 100 samples, 500 samples, and 1000 samples. Each synthetic dataset included three input variables and two output (target) variables as shown in Table 6. The complete normalized datasets are available in Appendix A.

Table 6

Example Dataset with Three Input Variables and Two Target Variables

	x_1	x_2	x_3	y_1	y_2
0	0.299688	0.537735	0.645746	0.399578	0.600950
1	0.331838	0.644123	0.293120	0.253450	0.325425
2	0.295416	0.358360	0.811418	0.457989	0.717744
3	0.536015	0.356220	0.477311	0.476763	0.551227
4	0.233822	0.359595	0.549973	0.268055	0.458148
5	0.424951	0.127427	0.504187	0.366132	0.484670

The next step was training a classical neural network regression model to predict multiple response variables. This task used the multi-layer perceptron regressor (MLPRegressor) class from Scikit-Learn. The MLPRegressor class provides three optimization algorithms to solve for the weights: the limited-memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS) method (Liu &

Nocedal, 1989), the stochastic gradient descent (SGD) method (Robbins & Monro, 1951), and the Adam optimization method (Kingma & Lei, 2015). The 100, 500, and 1000 sample datasets were split using 80/20, 70/30, and 60/40 split ratios for use with the MLPRegressor during training. Each optimization algorithm (LBGFS, Adam, and SGD) was used to train the models using the different split ratios. The experiments also explored if the number of epochs, or training iterations, influenced the accuracy of the model. Therefore, additional experiments involved training the MLPRegressor using 100, 1000, 10000, and 100000 iterations. In all, there were 108 distinct trials in the training of the classical neural network models. For each model, the R^2, MAE, MSE, RMSE, training time, and prediction time performance metrics were collected.

Experiment 2: Constructing a Multiple Target Quantum Neural Network Regressor

In this experiment, the researcher investigated how to build a multiple target regression model using quantum neural networks. This task involved identifying the appropriate quantum neural network class, quantum circuits for encoding classical data into a quantum feature map, and quantum circuits for representing the model's weights. The Qiskit library provides the option between an EstimatorQNN and a SamplerQNN for the quantum neural network architecture. The EstimatorQNN predicts expectation values based on the evaluation of quantum mechanical observables. The SamplerQNN predicts the probabilities of measuring the integer index of a bitstring. Since this investigation's regression models involved predicting real numbers instead of probabilities, the EstimatorQNN was the most appropriate neural network for this experiment. The Qiskit library's options for the feature map circuits include a standard quantum circuit (QuantumCircuit), the Pauli expansion circuit, the first-order Pauli-Z evolution circuit (ZFeatureMap), and the second-order Pauli-Z evolution (ZZFeatureMap) circuit (Havlicek et al.,

2019). For a regression model using multiple inputs and hundreds of samples, the QuantumCircuit class provided the most flexibility to automate the creation of the feature map circuit with multiple circuit parameters.

The Qiskit library provides a mechanism for assigning tunable weights to each of the features in the feature map. Optimization algorithms optimize these weights to minimize the objective function, such as the mean squared error or the mean absolute error when dealing with regression problems. The weights for each feature are represented by a parameterized quantum circuit, also known as an ansatz. The Qiskit circuits that can be used for the ansatz include the NLocal, TwoLocal, RealAmplitudes, and the EfficientSU2 circuits. These circuits include the capability of generating entangled states, which can provide the advantage of efficiently representing the solution space and capturing correlations in quantum data (Sim et al., 2019). The ansatz for this experiment used the RealAmplitudes circuit with repeated layers with the goal of strongly entangling the circuits (Havlíček et al., 2019). This experiment also explored automating the creation of the ansatz to simplify the parameterization process.

A limitation uncovered during this experiment revealed that the loss function used by Qiskit's neural networks does not currently support working with multiple response variables. This is a fundamental requirement for neural networks and this limitation has resulted in numerous support requests in Qiskit's GitHub repository. To address this challenge, an alternative approach was adopted, involving training a separate model for each target response variable. This experiment explored this approach by creating a new class that automates the generation of individual models encapsulated in the new class. This approach provided a higher level of abstraction and the capability for each internal model to predict the corresponding target response variable.

Qiskit recommends normalizing feature maps to a scale between 0 and 1. Therefore, this experiment explored the inclusion of methods for data normalization and denormalization. This experiment focused on providing users of the new class with the flexibility to select between the LBFGSB method (Zhu et al., 1997), the Adam optimization method (Kingma & Lei, 2015), the constrained optimization by linear approximation (COBYLA) method (Powell, 1994), and the sequential least squares programming (SLSQP) method (Kraft, 1988) for the quantum neural network's optimization algorithm. This experiment concluded with the implementation of methods that provide the R^2, MAE, MSE, and RMSE performance metrics.

Experiment 3: Training a Multiple Target Quantum Neural Network Regression Model

This experiment used the new multiple target quantum neural network regressor artifact constructed during the second experiment. The experiments were conducted using Qiskit's quantum simulator running on a classical processor. The experiments used the same synthetic normalized datasets generated during the classical neural network experiments. The 100, 500, and 1000 sample datasets were split using 80/20, 70/30, and 60/40 split ratios for use with the quantum neural network model during training. Each optimization algorithm (LBGFSB, COBYLA, Adam, and SLSQP) was used to train the models using the different split ratios. The experiments tested different training iterations using 100, 300, and 500 iterations. These lower iteration numbers were used since the quantum neural networks appear to converge sooner than classical neural networks but conversely run much longer during simulations. In all, there were 108 distinct trials in the training of the quantum neural network models. For each model, the R^2, MAE, MSE, RMSE, training time, and prediction time performance metrics were collected.

The source code for these experiments was tested on an enterprise-grade cloud server. This investigation contributed to existing methods by establishing a formal process of training a

quantum neural network model to predict multivariate responses and defining formal methods on how to assess the model's performance.

Data Analysis

The second question this investigation aimed to answer was: How does the training and prediction performance of a quantum multivariate multiple regression neural network model compare to the performance of a classical one? To answer this question, the performance metrics for each classical and quantum neural network model were collected. These metrics included statistics such as the R^2, MAE, MSE, RMSE, training time, and prediction time.

The R^2 score (coefficient of determination) is a metric for measuring how well a regression model fits the dataset and how likely future events fall within the predicted outcomes (Ennouri et al., 2017; Mohamed, 2019). The R^2 score is computed by subtracting the quotient of the sum of residuals squared (SSR) and the sum of squares total (SST) from 1. The SSR is calculated by finding the sum of square differences between the actual values and predicted values. The SST is calculated by finding the sum of the square differences between the actual values and the mean.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}\left(Y_i - \widehat{Y}_i\right)^2}{\sum_{i=1}^{n}(Y_i - \bar{Y})^2} \quad (1)$$

The MAE is often used to assess the model accuracy of multivariate regression models (Qi et al., 2020). A lower MAE typically indicates better long-term prediction performance of the neural network model (Doreswamy & Vastrad, 2013). This metric was measured by computing the average magnitude of the absolute differences between the predicted values produced by the quantum neural network and the true values from the training dataset.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|Y_i - \widehat{Y}_i\right|$$

(2)

The MSE is calculated by finding the mean of the squared differences between the actual values and predicted values produced by the model. It is important to note that the MSE is more sensitive to outliers than the MAE (Xu et al., 2019). The RMSE also penalizes significant errors and is measured in the same units as the target variable (Sahlol et al., 2017). It is calculated by computing the square root of the Mean Square Error (MSE). The performance metrics of the experiments were discussed in Chapter 4 along with the source code for generating the scores.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \widehat{Y}_i\right)^2$$

(3)

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \widehat{Y}_i\right)^2}$$

(4)

Assumptions and Limitations

The researcher made two assumptions concerning the investigation. The first was that the Scikit-learn library correctly implemented neural network models free of software bugs and errors. The second assumption was that the Qiskit quantum software development kit also correctly implemented neural network models free of software bugs and errors. Quantum computers by their very nature are susceptible to errors due to interference or noise from the environment (Preskill, 2018). Examples of interference include cosmic rays, neutrons from the earth's atmosphere, or radiation from wireless devices, therefore measurement results may not be

entirely accurate (Wong, 2022). However, the Qiskit quantum software development kit provides error mitigation techniques to minimize the effects of noise (Qiskit contributors, 2023). These techniques may increase the number of times a calculation is performed with the tradeoff of improving the accuracy of the calculation.

A notable limitation of this investigation were the technical constraints imposed by IBM's quantum cloud service. The service provides researchers with complimentary access to 5-qubit and 7-qubit quantum processors. These quantum processors do not have sufficient memory to process large datasets. Commercial access plans provide access to 27-qubit, 65-qubit, and 127-qubit processors. The commercial plans bill the usage of the quantum processors at a rate of $1.60 per second. Performing computations on quantum processors requires executing quantum circuits thousands of times (shots) to get a probability distribution of the measurement outcomes. The default number of shots on the quantum cloud is 4000, meaning algorithms are executed 4000 times during each training iteration. The complimentary and commercial plans also limit the amount of time a user can reserve a processor. Due to these constraints and the number of shots required for reliable measurements, the experiments attempted on the quantum processors exceeded the time allotted to users and resulted in the execution terminating before the models could successfully be trained. Therefore, the experiments were conducted on quantum simulators.

Delimitations

This investigation used synthetic regression data generated by the Scikit-learn library. The use of synthetic data is a common practice in machine learning research and eliminates the need to collect raw data and label it accordingly (Nikolenko, 2021). The use of synthetic data

helped generalize the methodology resulting from this investigation so that it can be applied across different domains.

Ethical Assurances

The researcher attests that the investigation did not use any human subjects. The researcher completed formal training and certification through the Collaborative Institutional Training Initiative website and sought formal approval from the Northcentral University Institutional Review Board (IRB) using the IRBManager website. Following this request, the Northcentral University IRB made the official determination that the investigation was considered Not Human Subjects Research (NHSR) and provided a certification to the researcher. A copy of this certification was made available in Appendix B.

Summary

This chapter included a review of the research method and design of the investigation. The design science research methodology was selected for the investigation and helped answer the research questions on how to construct a quantum neural network for multivariate multiple regression problems and how the performance of the quantum model compares to the performance of a classical model. The materials for this investigation involved computing hardware along with the software development kits necessary to construct quantum and classical neural networks. The study procedures included an iterative experimentation process where multiple quantum neural network architectures were trained using multivariate regression data. The data analysis section included detailed information on how the performance of the classical and quantum neural networks were measured. The chapter concluded with assumptions, limitations, delimitations, and the ethical assurances about the research study. The results of this investigation were presented in Chapter 4 along with a detailed analysis of the findings.

Chapter 4: Findings

The results for the experimental research are presented in this chapter. The problem addressed by this investigation was the lack of understanding on how to train a multivariate multiple regression model using quantum neural networks to predict multivariate responses effectively. A review of current literature found limited research on the usage of quantum neural networks for multivariate multiple regression problems. The purpose of this investigation was to design a quantum neural network model capable of predicting multivariate responses.

The study procedures were divided into three experiments. The first experiment involved generating a synthetic multidimensional regression dataset using Scikit-Learn, training and optimizing the classical neural network models, and collecting the performance metrics for each model.

The second experiment involved constructing a new artifact: a Python module that can be used by researchers to construct a multiple target regression model using quantum neural networks from the Qiskit library. The second experiment also explored the automatic generation of feature maps, ansatz circuits, and methods for assessing a trained model's performance.

The final experiment involved using the synthetic multidimensional regression data to validate the new multiple target quantum neural network class, train and optimize different models, and collect the performance metrics for each model. All experiments were conducted on an enterprise-grade cloud server with an Intel Xeon Platinum 3.70 GHz 16-core CPU and 32 GB RAM. The results of the research provided a strategy on using quantum neural networks for multivariate multiple regression problems, along with techniques for evaluating the performance of the model.

Experiment 1: Training a Multiple Target Classical Neural Network Regression Model

The goal of the first experiment was to identify the best performing classical neural network models after testing different combinations of hyperparameters and sample sizes. For this task, the researcher used the multilayer perceptron neural network regressor class (MLPRegressor) from the Scikit-Learn library. The Scikit-Learn's *make_regression*() utility was used to generate a synthetic multivariate multiple regression dataset with three input features and two target outputs (Figure 6). Three datasets were generated with 100 samples, 500 samples, and 1000 samples to assess the impact the sample size has on the performance of the models. All values in the datasets were normalized to values between 0 and 1 using Scikit-Learn's *MinMaxScaler*() utility since the quantum neural networks in Experiment 3 required the data to be normalized to offer better performance. The normalized datasets were made available in Appendix A.

Figure 6

Scikit-Learn's make_regression utility

```
1  X, Y = make_regression(n_samples=n_samples, n_features=n_features, n_targets=n_targets, random_state=42)
```

Top Ten Classical Neural Network Results

The MLPRegressor class provided the limited-memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS) method (Liu & Nocedal, 1989), the stochastic gradient descent (SGD) method (Robbins & Monro, 1951), and the Adam optimization method (Kingma & Lei, 2015). The 100, 500, and 1000 sample datasets were split using 80/20, 70/30, and 60/40 split ratios for use with the MLPRegressor during training. Each optimization algorithm (LBGFS, Adam, and SGD) was used to train the models using the different split ratios. The experiments also explored if the number of epochs, or training iterations, influenced the accuracy of the model. Therefore,

additional experiments involved training the MLPRegressor using 100, 1000, 10000, and 100000 iterations. In all, there were 108 distinct trials in the training of the classical neural network models. For each model, the R^2, MAE, MSE, RMSE, training time, and prediction time performance metrics were collected. The results for the 108 models were ranked by R^2 score since it is the metric that measures how well a model fits the dataset. The top ten models (Table 7) used the LBFGS optimizer and had R^2 scores between 0.9525 and 0.9812. The mean absolute error (MAE) scores for the top ten models ranged from 0.0262 to 0.0446. The mean squared error (MSE) scores ranged from 0.0011 to 0.0028 and the root mean squared error (RMSE) scores ranged from 0.0328 to 0.0514.

Table 7

Classical Neural Network Models Top 10 by R^2 Score

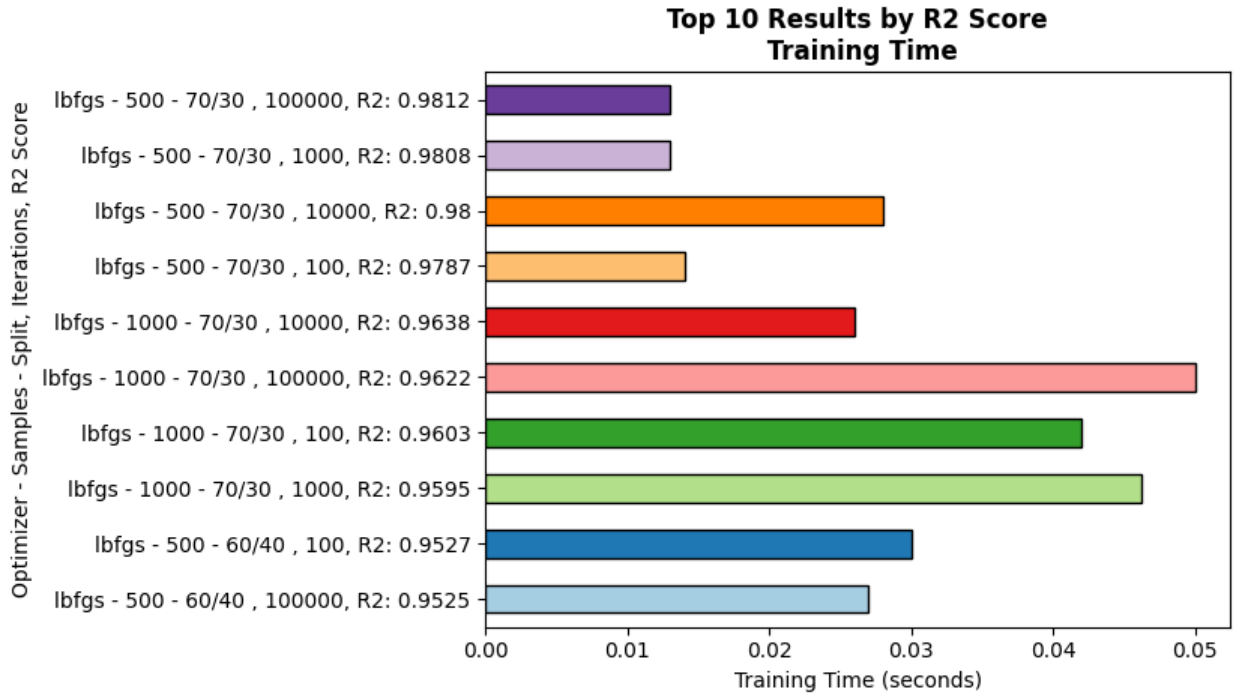
Rank	Features	Targets	Samples	Test Size	Optimizer	Iterations	Training Time	Prediction Time	R2	MAE	MSE	RMSE
1	3	2	500	30	lbfgs	100000	0.0130	0.000	0.9812	0.0262	0.0011	0.0328
2	3	2	500	30	lbfgs	1000	0.0130	0.000	0.9808	0.0270	0.0011	0.0333
3	3	2	500	30	lbfgs	10000	0.0280	0.000	0.9800	0.0268	0.0012	0.0336
4	3	2	500	30	lbfgs	100	0.0140	0.000	0.9787	0.0277	0.0013	0.0352
5	3	2	1000	30	lbfgs	10000	0.0260	0.000	0.9638	0.0397	0.0024	0.0479
6	3	2	1000	30	lbfgs	100000	0.0500	0.000	0.9622	0.0404	0.0025	0.0490
7	3	2	1000	30	lbfgs	100	0.0420	0.001	0.9603	0.0413	0.0026	0.0504
8	3	2	1000	30	lbfgs	1000	0.0462	0.000	0.9595	0.0416	0.0027	0.0506
9	3	2	500	40	lbfgs	100	0.0300	0.001	0.9527	0.0446	0.0028	0.0514
10	3	2	500	40	lbfgs	100000	0.0270	0.000	0.9525	0.0446	0.0028	0.0511

Note. Training time and prediction time are measured in seconds.

The training time for the top ten models by R^2 score is presented in Figure 7. The model with the best performance used 500 samples with a 70/30 training and test split. This model was trained using 100,000 iterations and also had the shortest training time with 0.0130 seconds. The full training results and performance metrics for all 108 experiments are available in Appendix C.

Figure 7

Training Time of Top 10 Classical Neural Network Models



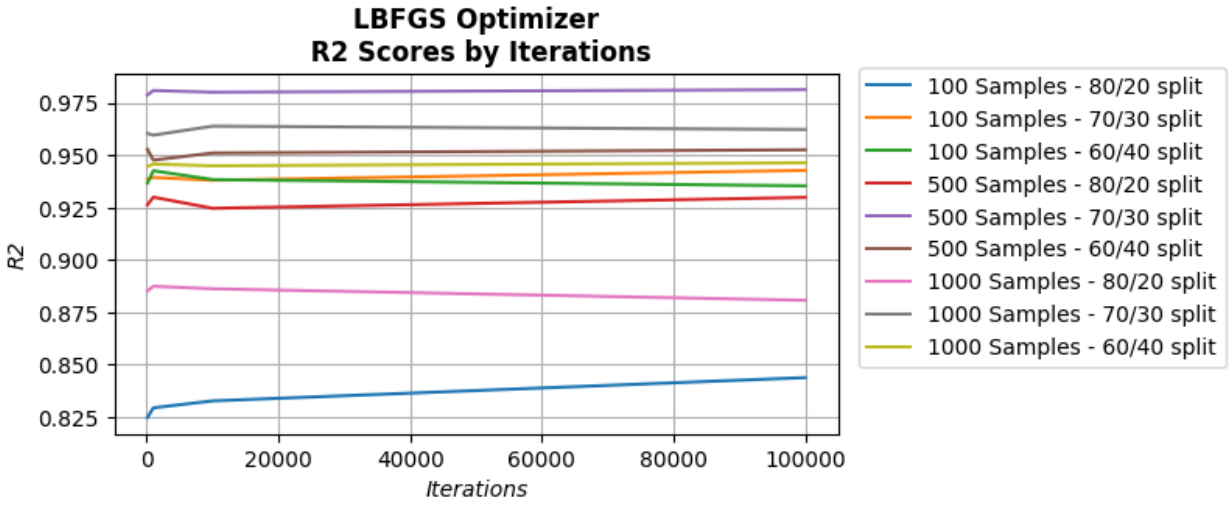
LBFGS Results

For the models trained with the LBFGS optimizer, the split ratios appeared to have a greater influence on the performance of the model compared to the number of iterations and sample sizes, as shown in Figure 8. The model with the lowest R^2 performance used 100 samples with an 80/20 split ratio. The model's R^2 scores started from approximately 0.825 and gradually improved with the number of iterations to just below 0.85. The other 100 sample models used

70/30 and 60/40 split ratios which resulted in performance scores between 0.925 and 0.95. The model with the best performance was also the best model of all classical neural network experiments and used 500 samples with a 70/30 split ratio and achieved an R^2 score of 0.9812.

Figure 8

Classical Neural Network LBFGS R^2 Scores



SGD Results

For the models trained with the SGD optimizer, the top ten results are presented in Table 8. The R^2 scores for these models ranged from 0.3869 to 0.5421. The model with the highest R^2 score used 500 samples with a 70/30 split ratio, and 1000 iterations. The MAE scores for the top ten models ranged from 0.1415 to 0.1841. The MSE scores ranged from 0.0277 to 0.0453 and the RMSE scores ranged from 0.1666 to 0.2128.

Table 8

Classical Neural Networks SGD Top 10 by R^2 Score

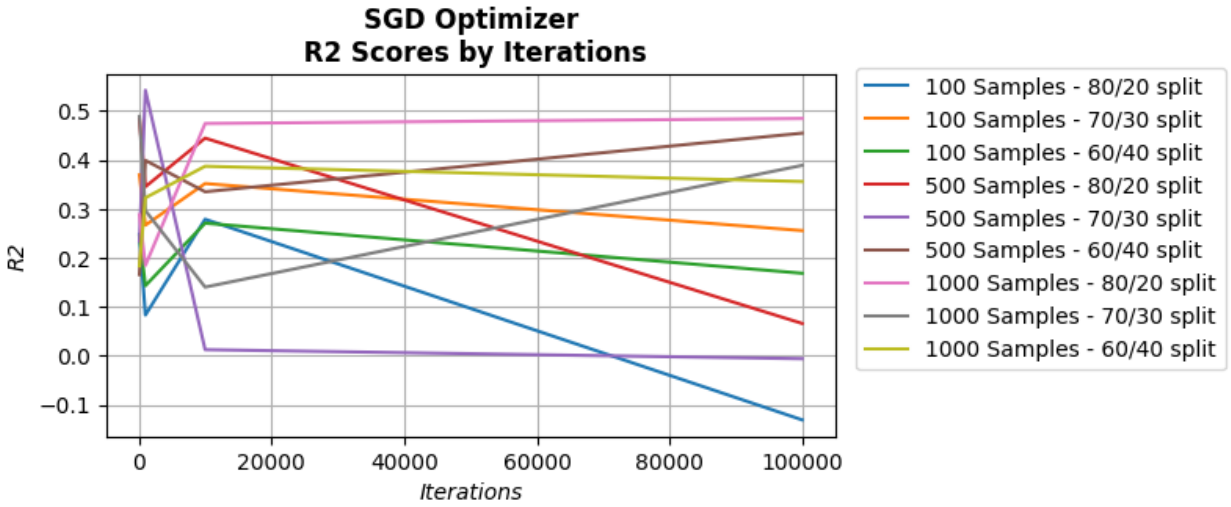
Rank	Features	Targets	Samples	Test Size	Optimizer	Iterations	Training Time	Prediction Time	R2	MAE	MSE	RMSE
1	3	2	500	30	sgd	1000	0.025	0.000	0.5421	0.1415	0.0277	0.1666
2	3	2	1000	30	sgd	100	0.031	0.000	0.4886	0.1454	0.0336	0.1832
3	3	2	1000	20	sgd	100000	0.038	0.000	0.4844	0.1585	0.0362	0.1898
4	3	2	500	20	sgd	100	0.029	0.000	0.4830	0.1808	0.0419	0.2048
5	3	2	1000	20	sgd	10000	0.042	0.000	0.4742	0.1623	0.0372	0.1919
6	3	2	500	40	sgd	100000	0.028	0.000	0.4545	0.1468	0.0336	0.1824
7	3	2	500	20	sgd	10000	0.025	0.000	0.4443	0.1841	0.0453	0.2128
8	3	2	500	40	sgd	1000	0.027	0.000	0.3988	0.1543	0.0366	0.1914
9	3	2	1000	30	sgd	100000	0.035	0.000	0.3887	0.1593	0.0402	0.2001
10	3	2	1000	40	sgd	10000	0.034	0.001	0.3869	0.1575	0.0388	0.1968

Note. Training time and prediction time are measured in seconds.

The SGD R^2 performance metrics by iterations are presented in Figure 9. The model with the highest R^2 score, 0.5421, achieved this score using 500 samples with a 70/30 split ratio and 1000 iterations. This model's R2 performance dropped significantly when trained for 10000 and 100000 iterations, with scores near 0. Most SGD models in the top ten realized a drop in R^2 scores with the exception of the 1000-sample model with a 70/30 split and the 500-sample model with a 60/40 split.

Figure 9

Classical Neural Network SGD R^2 Scores



ADAM Results

For the models trained with the ADAM optimizer, the top ten results are presented in Table 9. The R^2 scores for these models ranged from 0.9174 to 0.9379. The model with the highest score used 500 samples with a 60/40 split ratio and 1000 iterations. The MAE scores for the top ten models ranged from 0.0433 to 0.0596. The MSE scores ranged from 0.0039 to 0.0052 and the RMSE scores ranged from 0.0581 to 0.0719.

Table 9

Classical Neural Networks Adam Top 10 by R^2 Score

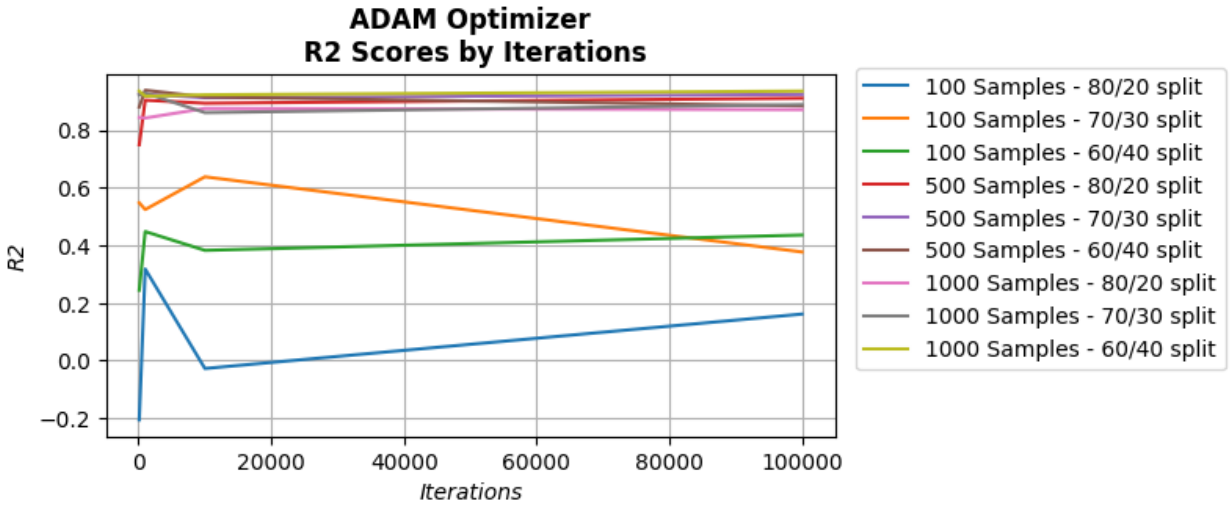
Rank	Features	Targets	Samples	Test Size	Optimizer	Iterations	Training Time	Prediction Time	R2	MAE	MSE	RMSE
1	3	2	500	40	adam	1000	0.066	0.000	0.9379	0.0484	0.0039	0.0611
2	3	2	1000	40	adam	100000	0.069	0.000	0.9349	0.0534	0.0041	0.0641
3	3	2	1000	40	adam	100	0.071	0.000	0.9348	0.0521	0.0041	0.0638
4	3	2	1000	30	adam	1000	0.073	0.001	0.9278	0.0533	0.0047	0.0676
5	3	2	500	30	adam	100	0.049	0.000	0.9267	0.0433	0.0042	0.0581
6	3	2	500	30	adam	1000	0.043	0.000	0.9239	0.0489	0.0044	0.0627
7	3	2	500	30	adam	100000	0.060	0.000	0.9234	0.0457	0.0044	0.0598
8	3	2	1000	40	adam	10000	0.079	0.000	0.9221	0.0578	0.0049	0.0702
9	3	2	1000	30	adam	100	0.061	0.000	0.9215	0.0559	0.0051	0.0710
10	3	2	1000	40	adam	1000	0.061	0.001	0.9174	0.0596	0.0052	0.0719

Note. Training time and prediction time are measured in seconds.

The ADAM R^2 performance metrics by iterations are presented in Figure 10. The model with the highest R^2 score, 0.9379, achieved this score using 500 samples with a 60/40 split ratio and 1000 iterations. Most models trained with the ADAM optimizer demonstrated no significant improvement in R^2 scores when trained over more iterations. Models using 100 samples performed less favorably than models using 500 or 1000 samples. Split ratios did not appear to have a significant influence on the performance of the models.

Figure 10

Classical Neural Network ADAM R^2 Scores



Experiment 2: Constructing a Multiple Target Quantum Neural Network Regressor

This section presents the results of Experiment 2 along with a detailed description of the new artifact, its composition, and its functionality.

Research Question 1. How can a multivariate multiple regression model be created using a quantum neural network to effectively predict multivariate responses?

This experiment helped answer this question with the design and implementation of a quantum neural network model for multivariate multiple regression problems through iterative experimentation and testing. The new artifact includes a multitarget quantum neural network regression class (Figure 6) to predict multivariate responses. This class was constructed in the Python programming language and includes formal methods for evaluating the model's performance. The full source code for this artifact was made available in Appendix D.

Model Composition

The new MultiTargetQuantumNNRegressor Python class accepts six parameters in its constructor to define the number of features, the number of targets, the solver for weight

optimization, the maximum number of training iterations, the option to use quantum cloud services, and the quantum hardware or simulator on which to perform the operations. The number of qubits to use during computation is determined by the number of input features the regression model uses.

Figure 11

New MultiTargetQuantumNNRegressor Python Class

```
class MultiTargetQuantumNNRegressor:
    r"""MultiTargetQuantumNNRegressor

    This class creates a multivariate multiple regression model by instantiating a separate quantum
    neural network for each target output. This class automatically constructs the necessary
    quantum circuits for the feature map and an ansatz for the trainable weights to encode
    classical data into a quantum state.
    """
    def __init__(
            self,
            n_features: int,
            n_targets: int,
            optimizer: Literal['cobyla', 'adam', 'lbfgsb', 'slsqp'],
            maxiter: int = 50,
            use_quantum_cloud: bool = False,
            backend: str = None
            ) -> None:
        r"""
        Parameters
        ----------
        n_features: int
            The number of features in the regression model.

        n_targets: int
            The number of regression targets i.e. the dimension of the y output vector.

        optimizer: {'cobyla', 'adam', 'lbfgsb','slsqp'}, default = 'slsqp'
            The solver for weight optimization.

        maxiter: int, default = 50
            Maximum number of iterations. The solver iterates until convergence.

        use_quantum_cloud: bool
            Determines whether to run on CPU or IBM Quantum Cloud.

        backend: str, default = None
            If running on quantum cloud, name of quantum hardware or simulator instance.
        """
```

Model Functionality

This new model provides users of the new class with the flexibility to select between the LBFGSB method (Zhu et al., 1997), the Adam optimization method (Kingma & Lei, 2015), the

constrained optimization by linear approximation (COBYLA) method (Powell, 1994), and the sequential least squares programming (SLSQP) method (Kraft, 1988) for the quantum neural network's optimization algorithm. To help encode the classical data into quantum states, the class generates a parameterized quantum circuit to serve as a feature map for the input variables (Figure 12). An ansatz is also generated to define trainable weights for each of the input features.

Figure 12

Quantum Circuit, Feature Map, and Ansatz Creation

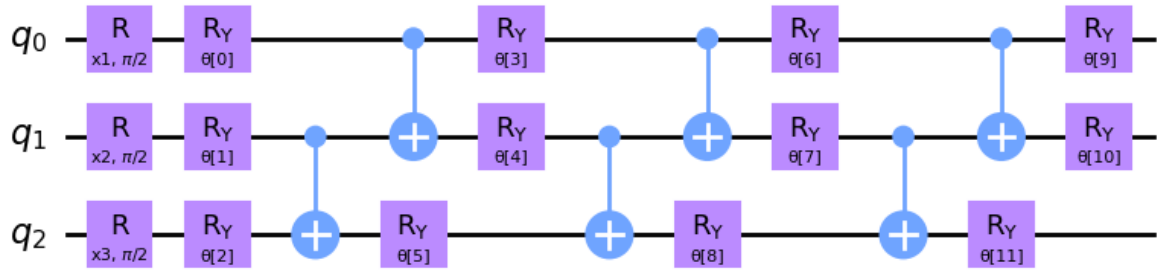
```
def _create_feature_map(self):
        """
        Creates a parameterized quantum circuit based on the number of features in the regression model.
        """
        feature_map = QuantumCircuit(self._n_features)
        parameters = []
        for i in range(self._n_features):
            parameter = Parameter(f"x{i}")
            parameters.append(parameter)
            feature_map.ry(parameter, i)
        return feature_map

    def _create_ansatz(self):
        """
        Creates an ansatz for the trainable weights.
        """
        ansatz = RealAmplitudes(self._n_features, reps=self._n_features)
        return ansatz

    def _create_quantum_circuit(self, feature_map: QuantumCircuit, ansatz: RealAmplitudes):
        """
        Builds the quantum circuit that will be used to encode the classical data.
        """
        qc = QuantumCircuit(self._n_features)
        qc.compose(feature_map, inplace=True)
        qc.compose(ansatz, inplace=True)
        return qc
```

The feature map circuit and the ansatz are used to create an instance of Qiskit's EstimatorQNN class for each target variable the user wishes to predict. An example quantum circuit for the feature map and ansatz used during Experiment 3 is shown in Figure 13. In this example, three input features were provided in the class constructor, so a three-qubit quantum circuit was automatically created. Features are encoded with an RGate placeholder followed by

an RYGate representing a single qubit rotation about the Y-axis. These gates represent the angles of rotation which are adjusted according to the value of the input feature. The qubits then have a CX Gate (CNOT) applied in reverse order to introduce an entangled state between them. In addition to the EstimatorQNN, the Qiskit library also provides a SamplerQNN data structure that outputs probabilities of measuring the integer index of a bitstring. Since the investigation involved predicting real numbers instead of probabilities, the EstimatorQNN was the most appropriate data structure in the Qiskit library for a quantum neural network regressor since it outputs expectation values instead of probabilities. The model's internal collection of EstimatorQNNs can be trained on a classical computer, a cloud-based quantum simulator, or on cloud-based quantum processors.

Figure 13

Quantum Circuit for Three Input Feature Map and Ansatz



The Qiskit documentation recommends users to scale training data to a range between 0 and 1. To help users normalize training data into this range, the MultiTargetQuantumNNRegressor class provides static methods for data normalization or denormalization, as shown in Figure 14. Once models are trained, their predicted values can be passed to the denormalization method to restore the values to an unscaled state.

Figure 14

Static Methods for Data Normalization or Denormalization

```
 1
 2  @staticmethod
 3  def get_scaled_data(X: np.ndarray, y: np.ndarray):
 4      """
 5      Scales the dataset down to values between 0 and 1.
 6      Note: Model training convergence is better when the transformation is applied.
 7
 8      Parameters
 9      ----------
10      X: numpy array of shape (n_samples, n_features)
11          The input data.
12
13      y: numpy array of shape (n_samples, n_targets)
14          The target values.
15      """
16      scaler = MinMaxScaler()
17      X = scaler.fit_transform(X)
18      if y is not None:
19          y = scaler.fit_transform(y)
20      return X, y
21
22  @staticmethod
23  def get_unscaled_data(X: np.ndarray, y: np.ndarray):
24      """
25      Reverses the scaling of the X and y values to original values.
26
27      Parameters
28      ----------
29      X: numpy array of shape (n_samples, n_features)
30          The input data.
31
32      y: numpy array of shape (n_samples, n_targets)
33          The target values.
34      """
35      scaler = MinMaxScaler()
36      X = scaler.inverse_transform(X)
37      if y is not None:
38          y = scaler.inverse_transform(y)
39      return X, y
```

The MultiTargetQuantumNNRegressor provides a *fit()* method to train the new quantum model (Figure 15). The method provides the option to normalize the data if the user has not

already done so. The fit method accepts feature input arrays of any dimension and target output arrays of any dimension, effectively allowing for multivariate multiple regression with any combination of the number of independent and dependent variables. The method verifies that the dimension shapes of the training data match what the user provided when the class was instantiated. Once the validation is completed, a separate quantum neural network for each output variable is trained and optimized for the number of iterations specified by the user. Each trained model is encapsulated within the MultiTargetQuantumNNRegressor class and does not require the user's involvement in separating the training or prediction data for the corresponding quantum neural network.

Figure 15

Model Fitting Method

```

def fit(self, X: np.ndarray, y: np.ndarray, scale_data: bool = False):
    """
    Begins training the neural network corresponding to each output variable.

    Parameters
    ----------
    X: numpy array of shape (n_samples, n_features)
        The input data.

    y: numpy array of shape (n_samples, n_targets)
        The target values.

    scale_data: bool, default = False
        Set this value to true if the dataset is not already scaled down to values between 0 and 1.
        Note: Model training convergence is better when the transformation is applied.
    """
    if X.shape[1] != self._n_features:
        raise ValueError(f"Shapes don't match, X features: {X.shape[1]}, n_features: {self._n_features}!")

    if y.shape[1] != self._n_targets:
        raise ValueError(f"Shapes don't match, y targets: {y.shape[1]}, n_targets: {self._n_targets}!")

    if X.shape[0] != y.shape[0]:
        raise ValueError(f"Shapes don't match, X samples: {X.shape[0]}, y samples: {y.shape[0]}!")

    if scale_data:
        X, y = self.get_scaled_data(X, y)

    self._fit(X,y)

def _fit(self, X: np.ndarray, y: np.ndarray):
    """
    Internally called from self.fit() to begin training each model.
    """
    for i in range(self._n_targets):
        self._neural_networks[i].fit(X, y[:, i])

```

Like the *fit()* method, the *predict()* method does require the user to invoke multiple calls to predict multiple target values using the separate quantum neural networks encapsulated in the MultiTargetQuantumNNRegressor class. When a user provides an input feature array, it is used by each internal quantum neural network to predict the corresponding target variable (Figure 16). The predicted values are stacked along the first axis to effectively form columns containing the multiple response values. This new array is returned to the user in the same fashion that the classical MLPRegressor does when predicting multiple response variables.

Figure 16

Prediction Method

```
def predict(self, X: np.ndarray, scale_data: bool = False):
    """
    Predict the target values given the input vector.

    Parameters
    ----------
    X: numpy array of shape (n_samples, n_features)
        The input data.

    Returns
    ----------
    y: numpy array of shape (n_samples, n_targets)
        The target values.
    """
    if scale_data:
        X, _ = self.get_scaled_data(X, None)
    if X.shape[1] != self._n_features:
        raise ValueError(f"Shapes don't match, X features: {X.shape[1]}, n_features: {self._n_features}!")
    return self._predict(X)

def _predict(self, X: np.ndarray):
    """
    Internally called from self.predict().
    """
    predictions = []
    for i in range(self._n_targets):
        prediction = self._neural_networks[i].predict(X)
        predictions.append(prediction)
    final_result = np.reshape(np.stack((predictions), axis= 1), (X.shape[0], len(predictions)))
    return final_result

```

For measuring the R^2 score of the trained model, the MultiTargetQuantumNNRegressor provides the *score()* method (Figure 17) that accepts the validation input values and the expected output values. This method automatically separates the expected output values and passes them to the corresponding trained model for scoring. The mean R^2 score of the collective models is returned to the user when scoring is completed. The same operations are performed for the MAE, MSE, and RMSE scoring methods as shown in Figure 18.

Figure 17

R^2 Score Method

```python
def score(self, X: np.ndarray, y: np.ndarray, scale_data: bool = False):
        """
        Compute the coefficient of determination i.e. the R-squared (R2) score.

        Parameters
        ----------
        X: numpy array of shape (n_samples, n_features)
            The input test data.

        y: numpy array of shape (n_samples, n_targets)
            The true target values.

        Returns
        ----------
        r2: float
            The r-squared (R2) score of the predictions.
        """
        if X.shape[1] != self._n_features:
            raise ValueError(f"Shapes don't match, X features: {X.shape[1]}, n_features: {self._n_features}!")

        if y.shape[1] != self._n_targets:
            raise ValueError(f"Shapes don't match, y targets: {y.shape[1]}, n_targets: {self._n_targets}!")

        if X.shape[0] != y.shape[0]:
            raise ValueError(f"Shapes don't match, X samples: {X.shape[0]}, y samples: {y.shape[0]}!")
        if scale_data:
            X, y = self.get_scaled_data(X, y)

        return self._score(X, y)

    def _score(self, X: np.ndarray, y: np.ndarray):
        """
        Internally called from self.score().
        """
        scores = []

        for i in range(self._n_targets):
            performance_score = self._neural_networks[i].score(X, y[:,i])
            scores.append(performance_score)

        return np.mean(scores)
```

Figure 18

MAE, MSE, and RMSE Methods

```
def mae(self, y_true, y_pred):
        """
        Computes the Mean Absolute Error.

        Parameters
        ----------
        y_true: numpy array of shape (n_samples, n_targets)
            The true target values.

        y_pred: numpy array of shape (n_samples, n_targets)
            The predicted target values.

        Returns
        ----------
        mae: float
            The mean absolute error regression loss.
        """
        return mean_absolute_error(y_true=y_true, y_pred=y_pred)

    def mse(self, y_true, y_pred):
        """
        Computes the Mean Squared Error.

        Parameters
        ----------
        y_true: numpy array of shape (n_samples, n_targets)
            The true target values.

        y_pred: numpy array of shape (n_samples, n_targets)
            The predicted target values.

        Returns
        ----------
        mae: float
            The mean squared error regression loss.
        """
        return mean_squared_error(y_true=y_true, y_pred=y_pred, squared=True)

    def rmse(self, y_true, y_pred):
        """
        Computes the Root Mean Squared Error.

        Parameters
        ----------
        y_true: numpy array of shape (n_samples, n_targets)
            The true target values.

        y_pred: numpy array of shape (n_samples, n_targets)
            The predicted target values.

        Returns
        ----------
        mae: float
            The root mean squared error regression loss.
        """
        return mean_squared_error(y_true=y_true, y_pred=y_pred, squared=False)
```

The MultiTargetQuantumNNRegressor class provides users with the ability to save or load trained models (Figure 19). The *save_model()* method allows users to serialize any instance of the MultiTargetQuantumNNRegressor class along with all internally trained quantum neural networks for later use. The saved model preserves all training weights in each of the quantum neural networks, allowing the user to reuse or distribute the trained model. The *load_model()* allows a user to open a serialized instance of the MultiTargetQuantumNNRegressor class and resume using the trained model to predict new target variables.

Figure 19

Save and Load Functionality

```
1
2  def save_model(self, model_name: str = 'model'):
3      """
4      Saves the model to disk.
5
6      Parameters
7      ----------
8      model_name: str, default = 'model'
9
10     """
11     with open(f'{model_name}', 'wb') as file:
12         pickle.dump(self, file)
13
14 @staticmethod
15 def load_model(model_name: str = 'model'):
16     """
17     Loads the saved model from disk.
18
19     Parameters
20     ----------
21     model_name: str, default = 'model'
22
23     Returns
24     ----------
25     model: MultiTargetQuantumNNRegressor
26         The saved instance of the MultiTargetQuantumNNRegressor.
27     """
28     with open(model_name, 'rb') as file:
29         model = pickle.load(file)
30         class_type = MultiTargetQuantumNNRegressor
31         if isinstance(model, class_type):
32             return model
33         else:
34             raise Exception(f"The model loaded is not an instance of the {class_type.__name__} class.")
35
```

Experiment 3: Training a Multiple Target Quantum Neural Network Regression Model

The goal of the third experiment was to identify the best performing quantum neural network regression models after testing different combinations of hyperparameters and sample sizes. For this task, the researcher used the new MultiTargetQuantumNNRegressor class created during Experiment 2 and the same training datasets used with the classical neural networks in Experiment 1.

Top Ten Quantum Neural Network Results

The 100, 500, and 1000 sample datasets were split with the same 80/20, 70/30, and 60/40 split ratios used with the classical neural networks during Experiment 1. Each optimization algorithm (LBGFSB, SLSQP, COBYLA, and ADAM) was used to train the models using the different split ratios. The experiments also explored if the number of training iterations influenced the accuracy of the model. An observation made during initial experimentation was that the quantum neural network optimization algorithms converge with fewer iterations than the optimization algorithms used with the classical neural networks. Therefore, these experiments involved training the MultiTargetQuantumNNRegressor using 100, 300, and 500 iterations. In all, there were 108 distinct trials in the training of the quantum neural network models. For each model, the R^2, MAE, MSE, RMSE, training time, and prediction time performance metrics were collected. The results for the 108 models were ranked by R^2 score.

The top ten models (Table 10) used the LBFGSB and SLSQP optimizers and had R^2 scores between 0.9337 and 0.9504. The mean absolute error (MAE) scores for the top ten models ranged from 0.0377 to 0.0497. The mean squared error (MSE) scores ranged from 0.0029 to 0.0041 and the root mean squared error (RMSE) scores ranged from 0.0539 to 0.0641. The

training times ranged from 0.6677 hours to 4.7603 hours. The prediction times ranged from 0.3037 seconds and 0.4342 seconds.

Table 10

Quantum Neural Network Models Top 10 by R^2 Score

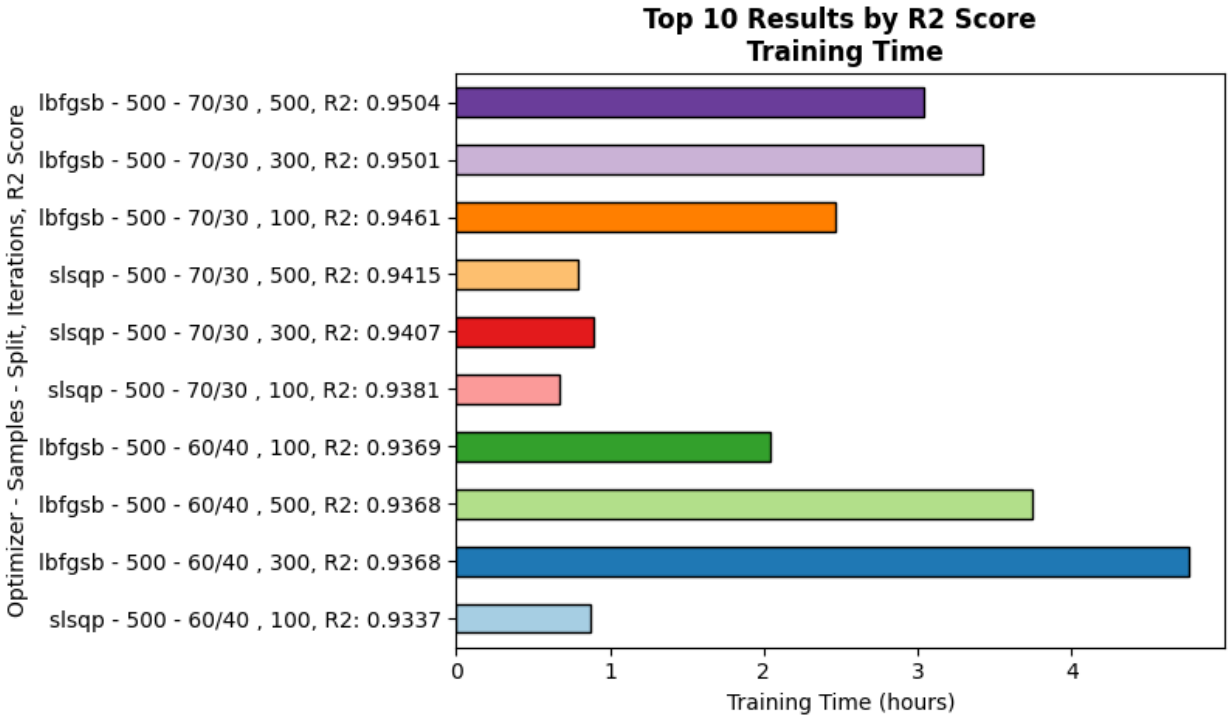
Rank	Features	Targets	Samples	Test Size	Optimizer	Iterations	Training Time	Prediction Time	R2	MAE	MSE	RMSE
1	3	2	500	30	lbfgsb	500	3.0381	0.3317	0.9504	0.0377	0.0029	0.0539
2	3	2	500	30	lbfgsb	300	3.4232	0.3060	0.9501	0.0378	0.0029	0.0540
3	3	2	500	30	lbfgsb	100	2.4656	0.3058	0.9461	0.0398	0.0031	0.0561
4	3	2	500	30	slsqp	500	0.7934	0.3114	0.9415	0.0438	0.0034	0.0586
5	3	2	500	30	slsqp	300	0.8991	0.3037	0.9407	0.0418	0.0035	0.0588
6	3	2	500	30	slsqp	100	0.6677	0.3150	0.9381	0.0423	0.0036	0.0601
7	3	2	500	40	lbfgsb	100	2.0474	0.4342	0.9369	0.0496	0.0039	0.0624
8	3	2	500	40	lbfgsb	500	3.7445	0.3938	0.9368	0.0497	0.0039	0.0625
9	3	2	500	40	lbfgsb	300	4.7603	0.4112	0.9368	0.0497	0.0039	0.0625
10	3	2	500	40	slsqp	100	0.8704	0.3931	0.9337	0.0496	0.0041	0.0641

Note. Training time is measured in hours. Prediction time is measured in seconds.

The training time for the top ten models by R^2 score is presented in Figure 20. The model with the best performance used 500 samples with a 70/30 training and test split. This model was trained using 500 iterations and had a training time of 3.0381 hours. The full training results and performance metrics for all 108 experiments were made available in Appendix E.

Figure 20

Training Time of Top 10 Quantum Neural Network Models



LBFGSB Results

For the models trained with the LBFGSB optimizer, the top ten results are presented in Table 11. The R^2 scores for these models ranged from 0.9133 to 0.9504. The model with the highest R^2 score used 500 samples with a 70/30 split ratio, and 500 iterations. The MAE scores for the top ten models ranged from 0.0377 to 0.0638. The MSE scores ranged from 0.0029 to 0.0076 and the RMSE scores ranged from 0.0532 to 0.0874. The training times ranged from 0.333 hours to 6.6972 hours. The prediction times ranged from 0.2034 seconds and 0.4342 seconds.

Table 11

Quantum Neural Networks LBFGSB Top 10 by R^2 Score

Rank	Features	Targets	Samples	Test Size	Optimizer	Iterations	Training Time	Prediction Time	R2	MAE	MSE	RMSE
1	3	2	500	30	lbfgsb	500	3.0381	0.3317	0.9504	0.0377	0.0029	0.0539
2	3	2	500	30	lbfgsb	300	3.4232	0.3060	0.9501	0.0378	0.0029	0.0540
3	3	2	500	30	lbfgsb	100	2.4656	0.3058	0.9461	0.0398	0.0031	0.0561
4	3	2	500	40	lbfgsb	100	2.0474	0.4342	0.9369	0.0496	0.0039	0.0624
5	3	2	500	40	lbfgsb	300	4.7603	0.4112	0.9368	0.0497	0.0039	0.0625
6	3	2	500	40	lbfgsb	500	3.7445	0.3938	0.9368	0.0497	0.0039	0.0625
7	3	2	100	40	lbfgsb	300	0.5409	0.3906	0.9249	0.0414	0.0028	0.0532
8	3	2	100	40	lbfgsb	100	0.3330	0.4075	0.9228	0.0418	0.0029	0.0539
9	3	2	100	40	lbfgsb	500	0.4880	0.4122	0.9161	0.0442	0.0032	0.0564
10	3	2	500	20	lbfgsb	500	6.6972	0.2034	0.9133	0.0638	0.0076	0.0874

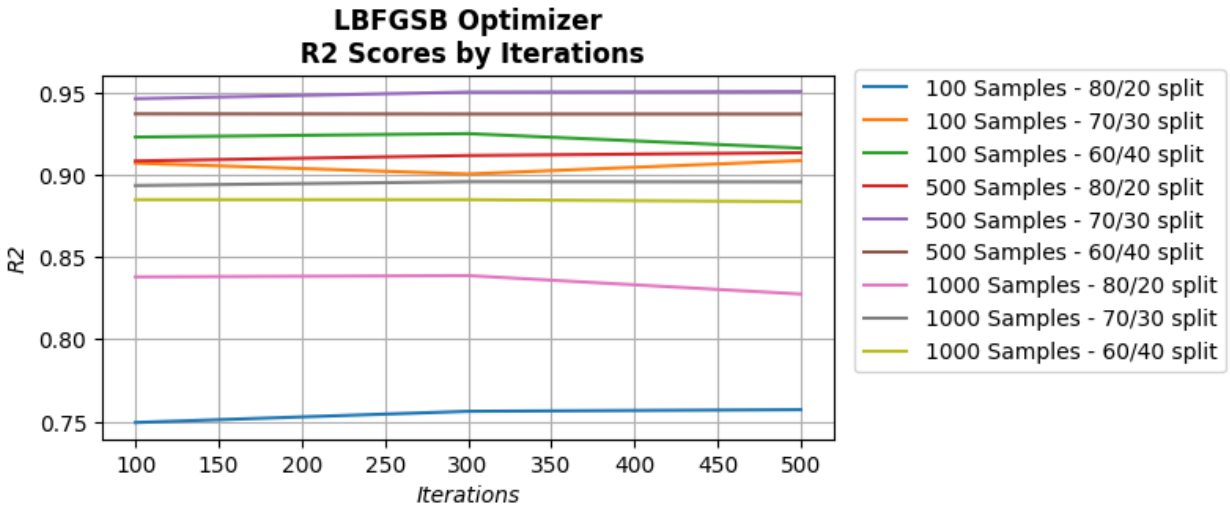
Note. Training time is measured in hours. Prediction time is measured in seconds.

The R^2 scores by iterations for all models trained with the LBFGSB optimizer are presented in Figure 21. The R^2 scores did not show a significant improvement as the number of training iterations increased. The models trained with 500 samples achieved higher scores than model models trained with 1000 samples. The model with the lowest R^2 performance used 100 samples with an 80/20 split ratio. The model's R^2 scores started from approximately 0.75 and gradually improved with the number of iterations to just below 0.77. The other 100 sample models used 70/30 and 60/40 split ratios which resulted in performance scores between 0.90 and 0.93. The model with the best performance was also the best model of all quantum neural

network experiments and used 500 samples with a 70/30 split ratio and achieved an R^2 score of 0.9504.

Figure 21

Quantum Neural Network LBFGSB R^2 Scores



SLSQP Results

For models trained with the SLSQP optimizer, the top ten results are presented in Table 12. The R^2 scores for these models ranged from 0.8889 to 0.9419. The model with the highest R^2 score used 500 samples with a 70/30 split ratio, and 500 iterations. The MAE scores for the top ten models ranged from 0.0418 to 0.0638. The MSE scores ranged from 0.0029 to 0.0076 and the RMSE scores ranged from 0.0532 to 0.0725. The training times ranged from 0.1134 hours to 1.0996 hours. The prediction times ranged from 0.1955 seconds and 0.4366 seconds.

Table 12

Quantum Neural Networks SLSQP Top 10 by R^2 Score

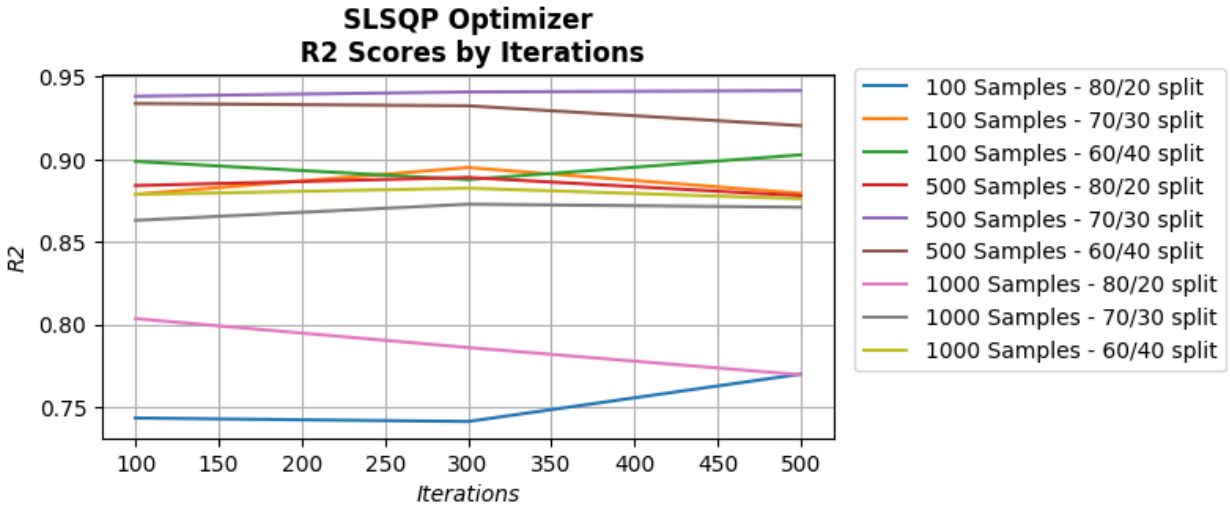
Rank	Features	Targets	Samples	Test Size	Optimizer	Iterations	Training Time	Prediction Time	R2	MAE	MSE	RMSE
1	3	2	500	30	slsqp	500	0.7934	0.3114	0.9415	0.0438	0.0034	0.0586
2	3	2	500	30	slsqp	300	0.8991	0.3037	0.9407	0.0418	0.0035	0.0588
3	3	2	500	30	slsqp	100	0.6677	0.3150	0.9381	0.0423	0.0036	0.0601
4	3	2	500	40	slsqp	100	0.8704	0.3931	0.9337	0.0496	0.0041	0.0641
5	3	2	500	40	slsqp	300	0.8486	0.4151	0.9323	0.0507	0.0042	0.0648
6	3	2	500	40	slsqp	500	0.6838	0.3874	0.9203	0.0537	0.0050	0.0706
7	3	2	100	40	slsqp	500	0.1134	0.4366	0.9026	0.0476	0.0037	0.0606
8	3	2	100	40	slsqp	100	0.1640	0.4337	0.8986	0.0483	0.0039	0.0623
9	3	2	100	30	slsqp	300	0.1685	0.3156	0.8949	0.0496	0.0045	0.0672
10	3	2	500	20	slsqp	300	1.0996	0.1955	0.8889	0.0725	0.0098	0.0992

Note. Training time is measured in hours. Prediction time is measured in seconds.

The R^2 scores by iterations for all models trained with the SLSQP optimizer are presented in Figure 22. Most of these models realized a decrease in R^2 scores after 300 iterations. The models trained with 500 samples achieved higher scores than model models trained with 1000 samples. The model with the lowest R^2 performance used 100 samples with an 80/20 split ratio. The model's R^2 scores started from approximately 0.74 and gradually improved with the number of iterations to just below 0.77. The other 100 sample models used 70/30 and 60/40 split ratios which resulted in performance scores between 0.87 and 0.91. The model with the best performance used 500 samples with a 70/30 split ratio and achieved an R^2 score of 0.9415.

Figure 22

Quantum Neural Network SLSQP R^2 Scores



COBYLA Results

For models trained with the COBYLA optimizer, the top ten results are presented in Table 13. The R^2 scores for these models ranged from 0.8219 to 0.9155. The model with the highest R^2 score used 500 samples with a 70/30 split ratio, and 500 iterations. The MAE scores for the top ten models ranged from 0.0553 to 0.0937. The MSE scores ranged from 0.0050 to 0.0156 and the RMSE scores ranged from 0.0705 to 0.1249. The training times ranged from 0.0545 hours to 1.4713 hours. The prediction times ranged from 0.2839 seconds and 0.5211 seconds.

Table 13

Quantum Neural Networks COBYLA Top 10 by R^2 Score

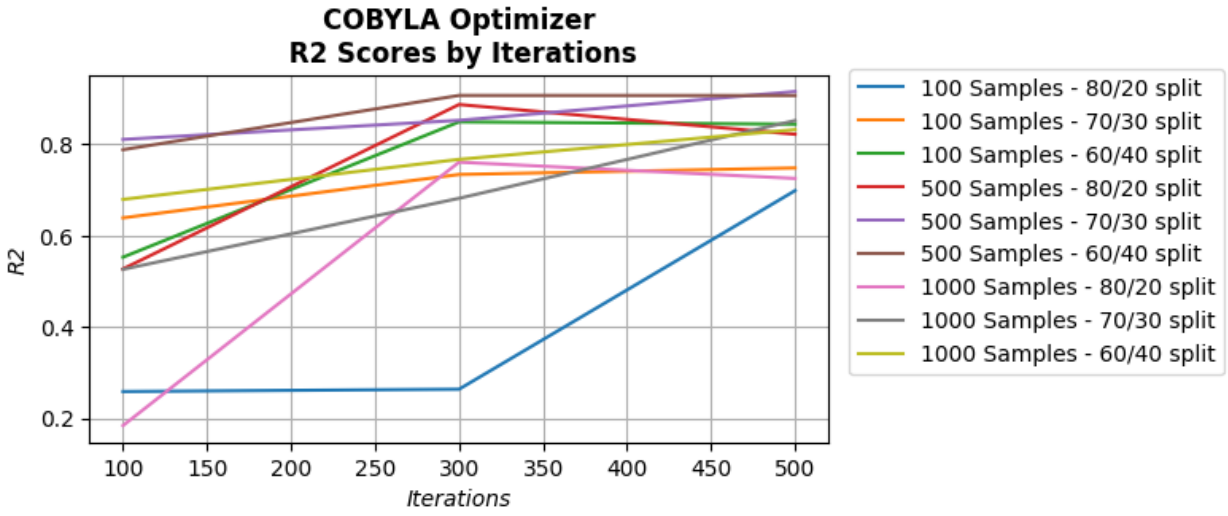
Rank	Features	Targets	Samples	Test Size	Optimizer	Iterations	Training Time	Prediction Time	R2	MAE	MSE	RMSE
1	3	2	500	30	cobyla	500	0.7005	0.3038	0.9155	0.0553	0.0050	0.0705
2	3	2	500	40	cobyla	300	0.4179	0.4090	0.9070	0.0572	0.0057	0.0756
3	3	2	500	40	cobyla	500	0.6882	0.5211	0.9068	0.0561	0.0058	0.0764
4	3	2	500	20	cobyla	300	0.4323	0.2839	0.8871	0.0725	0.0100	0.1001
5	3	2	500	30	cobyla	300	0.4249	0.3126	0.8524	0.0744	0.0088	0.0936
6	3	2	1000	30	cobyla	500	1.4713	0.3214	0.8519	0.0781	0.0098	0.0988
7	3	2	100	40	cobyla	300	0.0545	0.4193	0.8490	0.0585	0.0057	0.0753
8	3	2	100	40	cobyla	500	0.0901	0.4129	0.8440	0.0600	0.0060	0.0774
9	3	2	1000	40	cobyla	500	1.4608	0.5129	0.8320	0.0844	0.0106	0.1031
10	3	2	500	20	cobyla	500	0.7145	0.3106	0.8219	0.0937	0.0156	0.1249

Note. Training time is measured in hours. Prediction time is measured in seconds.

The R^2 scores by iterations for all models trained with the COBYLA optimizer are presented in Figure 23. Most of these models realized a gradual increase in R^2 scores with more training iterations. The model with the lowest R^2 performance used 100 samples with an 80/20 split ratio. The model's R^2 scores started from approximately 0.28 and improved with the number of iterations to just below 0.70. The other 100 sample models used 70/30 and 60/40 split ratios which resulted in performance scores between 0.65 and 0.80. The model with the best performance used 500 samples with a 70/30 split ratio and achieved an R^2 score of 0.9155.

Figure 23

Quantum Neural Network COBYLA R^2 Scores



ADAM Results

For models trained with the ADAM optimizer, the top ten results are presented in Table 14. The R^2 scores for these models ranged from -0.3058 to 0.5527. The model with the highest R^2 score used 500 samples with an 80/20 split ratio, and 500 iterations. The MAE scores for the top ten models ranged from 0.1483 to 0.2302. The MSE scores ranged from 0.0320 to 0.0829 and the RMSE scores ranged from 0.1710 to 0.2880. The training times ranged from 1.0901 hours to 22.1203 hours. The prediction times ranged from 0.1697 seconds and 0.4419 seconds.

Table 14

Quantum Neural Networks ADAM Top 10 by R^2 Score

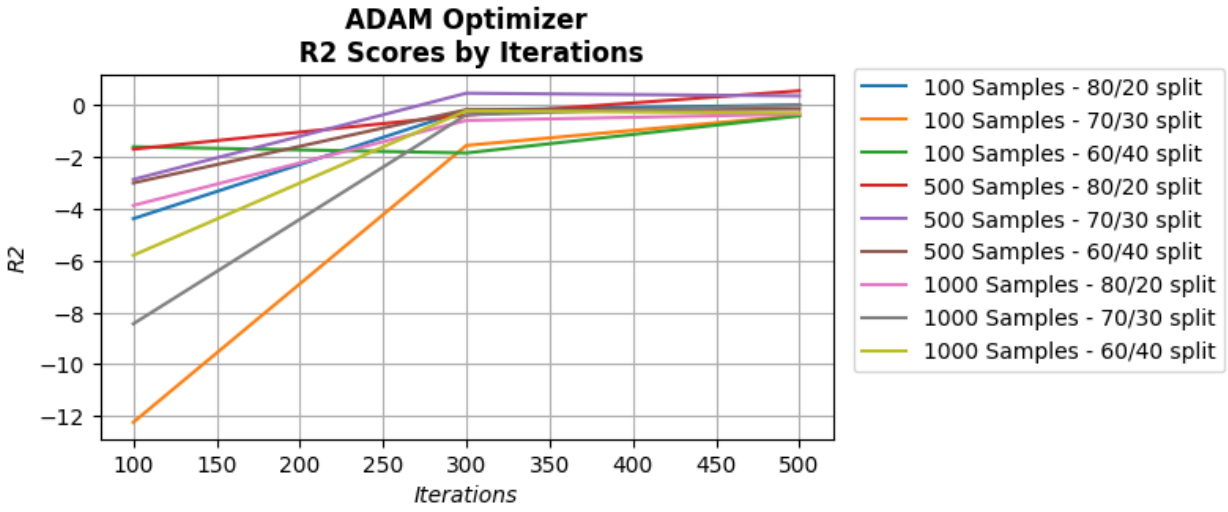
Rank	Features	Targets	Samples	Test Size	Optimizer	Iterations	Training Time	Prediction Time	R2	MAE	MSE	RMSE
1	3	2	500	20	adam	500	10.9021	0.1697	0.5527	0.1526	0.0384	0.1961
2	3	2	500	30	adam	300	6.4085	0.3127	0.4604	0.1483	0.0320	0.1790
3	3	2	500	30	adam	500	10.7049	0.3085	0.3561	0.1550	0.0380	0.1949
4	3	2	100	20	adam	500	1.8341	0.2299	0.0075	0.2152	0.0641	0.2532
5	3	2	1000	30	adam	500	22.1203	0.3292	0.0058	0.2070	0.0658	0.2565
6	3	2	500	40	adam	500	10.4428	0.4060	-0.1420	0.1825	0.0731	0.2704
7	3	2	500	40	adam	300	6.3144	0.4317	-0.1775	0.2187	0.0716	0.2675
8	3	2	100	20	adam	300	1.0901	0.2487	-0.1966	0.2149	0.0829	0.2880
9	3	2	1000	40	adam	300	13.0347	0.4095	-0.2178	0.2302	0.0772	0.2778
10	3	2	1000	40	adam	500	21.9940	0.4419	-0.3058	0.2281	0.0829	0.2879

Note. Training time is measured in hours. Prediction time is measured in seconds.

The R^2 scores by iterations for all models trained with the ADAM optimizer are presented in Figure 24. Most of these models realized an increase in R^2 scores with more training iterations. The model with the lowest R^2 performance used 100 samples with a 70/30 split ratio. The model's R^2 scores started from approximately -12.00 and improved with the number of iterations to just above -1.00. The other 100 sample models used 80/20 and 60/40 split ratios which resulted in performance scores between -4.00 and -1.00. The model with the best performance used 500 samples with an 80/20 split ratio and achieved an R^2 score of 0.5527.

Figure 24

Quantum Neural Network ADAM R^2 Scores



Evaluation of the Findings

Research Question 2. How does the training and prediction performance of a quantum multivariate multiple regression neural network model compare to the performance of a classical one?

A comparison of the best classical model and the best quantum model is presented in Table 15. Both models used a variation of the LBFGS optimizer, 500 samples, and a 70/30 split ratio. The classical model was trained in 0.0130 seconds and achieved an R^2 score of 0.9812 after 100000 iterations. The quantum model was trained in 3.0381 hours and achieved an R^2 score of 0.9504 after 500 iterations. The classical model's prediction time was below one millisecond while the quantum model's prediction time was 0.3317 seconds.

Table 15

Classical vs Quantum Multitarget Regression Models

Type	Features	Targets	Samples	Test Size	Optimizer	Iterations	Training Time	Prediction Time	R2	MAE	MSE	RMSE
Classical	3	2	500	30	lbfgs	100000	0.0130	0.000	0.9812	0.0262	0.0011	0.0328
Quantum	3	2	500	30	lbfgsb	500	3.0381	0.3317	0.9504	0.0377	0.0029	0.0539

Note. Training time for the classical model is measured in seconds. Training time for the quantum model is measured hours. Prediction time for both models is measured in seconds.

Summary

The results of the present research provided a strategy on using quantum neural networks for multivariate multiple regression problems along with techniques for evaluating the performance of the model to address the first research question. This chapter introduced the novel MultiTargetQuantumNNRegressor class as a Python module with which researchers can experiment and train multivariate multiple regression models. The experimental results of training quantum models helped answer the second research question and revealed that similar performance metrics comparable to those of classical models can be achieved, albeit with significant costs in training time. The next chapter concludes with a discussion on the implications of this investigation and recommendations for future research.

Chapter 5: Implications, Recommendations, and Conclusions

The problem addressed by this investigation was the lack of understanding on how to train a multivariate multiple regression model using quantum neural networks to predict multivariate responses effectively. A review of current literature found limited research on the usage of quantum neural networks for multivariate multiple regression problems. The purpose of this investigation was to design a quantum neural network model capable of predicting multivariate responses. The researcher applied a design science research methodology (Hevner & Chatterjee, 2010; March & Smith, 1995; Peffers et al., 2007) in the construction of a new artifact. The artifact presented in this manuscript extends previous methods by encapsulating multiple quantum neural networks in a Python module such that it can predict multiple targets when trained with a multivariate multiple regression dataset.

This research introduced the novel MultiTargetQuantumNNRegressor class as a Python module with which researchers can experiment and train multivariate multiple regression models. The results of the research provided a strategy on using quantum neural networks for multivariate multiple regression problems, along with techniques for evaluating the performance of the model. This investigation had limitations due to the technological constraints of near-term quantum processors. Therefore, experiments performed in this research were conducted with simulated quantum processors. Experiments attempted on real quantum processors exceeded the time allotted to users and resulted in the execution terminating before the models could successfully be trained. Should the resource constraints be lifted, the new MultiTargetQuantumNNRegressor module can be used with quantum processors. The remainder of this chapter discusses the implications of the study followed by recommendations for practice and future research.

Implications

Research Question 1. How can a multivariate multiple regression model be created using a quantum neural network to effectively predict multivariate responses?

With the construction of the new MultiTargetQuantumNNRegressor module, researchers can now train a multivariate multiple regression quantum neural network model, measure its performance, as well as save and load the model to and from a filesystem. This module has the benefit of automatically creating the quantum circuits, the ansatz, and performing the entangling operations on each qubit. This, in effect, enables a user to focus more on the objectives they wish to achieve and spend less time manually creating custom quantum circuits.

Research Question 2. How does the training and prediction performance of a quantum multivariate multiple regression neural network model compare to the performance of a classical one?

The best quantum neural network model was outperformed by the best classical neural model trained with the same synthetic multivariate multiple regression data (Table 16). The quantum model, nonetheless, achieved a favorable R^2 score of 0.9504 compared to the classical model's R^2 of 0.9812. A significant drawback of the quantum neural network model is its training time of 3.0381 hours. This translates to approximately 10,937 seconds of runtime. Training this model on a cloud-based quantum processor would exceed the allotted runtime on a complimentary access plan. Training this model with a paid-access plan would cost approximately $17,500. In experiments using Qiskit's EstimatorQNN as a binary classifier, Tehrani et al. (2023) trained their model with 1000 samples and the COBYLA optimizer to achieve an 84% accuracy and an execution time of 410 seconds on a simulator. This signifies that the findings of the current investigation were consistent with existing research. It is not

presently practical to use this quantum neural network model on the currently available quantum processors given the lower cost and higher speed offered by classical models that can be trained on personal computers. Once the memory and other hardware challenges of quantum processors are addressed, executing this quantum model on actual hardware may result in more favorable performance metrics.

Table 16

Classical vs Quantum Multitarget Regression Models

Type	Features	Targets	Samples	Test Size	Optimizer	Iterations	Training Time	Prediction Time	R2	MAE	MSE	RMSE
Classical	3	2	500	30	lbfgs	100000	0.0130	0.000	0.9812	0.0262	0.0011	0.0328
Quantum	3	2	500	30	lbfgsb	500	3.0381	0.3317	0.9504	0.0377	0.0029	0.0539

Note. Training time for the classical model is measured in seconds. Training time for the quantum model is measured hours. Prediction time for both models is measured in seconds.

Recommendations for Practice

Although the approach introduced by the current investigation is presently cost-prohibitive on quantum processors, it provides a theoretical framework that researchers can use once the physical hardware limitations are lifted. The MultiTargetQuantumNNRegressor enables users to leverage quantum technology to predict multiple response variables, especially in applications such as cancer research (Lee et al., 2012; Lee & Liu, 2012), agriculture (Johnson & Wichern, 2014), aviation (Rencher & Christensen, 2012), and defense (Filgöz et al., 2021; Kang, 2021; Yoon et al., 2010). Specialists can conduct experiments to assess how their models perform when trained with quantum neural networks.

Recommendations for Future Research

Future research on the usage of quantum neural networks for multivariate multiple regression problems should include optimizing the training process. One possible way to do this

is to parallelize the training of the internal quantum networks to reduce the training and prediction times. The parallel training of the quantum neural networks can be accomplished with a multiprocessing library. With the parallel training of the internal quantum neural networks, training models with extra dimensions becomes more feasible. While this study explored training models with three input features and two output variables, future research could explore models with higher dimensions, such as ten input features with ten output variables. The Qiskit library recently introduced an interface, TorchConnector, between its quantum library and the PyTorch framework to enable the training of quantum machine learning models using graphics processing units (GPUs). Recent deep learning experiments using TorchConnector allowed multiple layers of the quantum neural networks to be trained in parallel using a GPU (Chen, 2023). This hybrid approach may reduce the training and prediction time of the quantum models.

Another recommendation is to explore the effect on model accuracy improves when using a greater number of training iterations. The quantum models trained with the COBYLA and ADAM optimizers exhibited an improvement in R^2 scores when trained for more iterations, while several models trained with the SLSQP realized a decline in R^2 scores when trained with more iterations. Models trained with the LBFGSB optimizer showed marginal improvements in R^2 scores with more iterations. Further research can explore the effects of training with 1000 or more iterations.

The final research recommendation is to explore the optimization of the quantum circuits of the MultiTargetQuantumNNRegressor. What combination of quantum gates can help achieve better results with respect to training time, prediction time, and accuracy? The current investigation explored the automatic creation of quantum circuits with feature maps that applied an RYGate on the input variables and an ansatz using the RealAmplitudes circuit with a reverse

linear entanglement operation on the qubits. Further research should explore whether better results can be achieved with an alternative approach.

Conclusions

This study addressed the lack of understanding on how to train a multivariate multiple regression model using quantum neural networks to predict multivariate responses effectively. This research introduced the novel MultiTargetQuantumNNRegressor Python module researchers can use to train multivariate multiple regression models. The results of the research provided a strategy on using quantum neural networks for multivariate multiple regression problems along with techniques for evaluating the performance of the model.

The applications of machine learning and artificial intelligence have had remarkable impacts on society as well as scientific endeavors. The advent of openly available cloud-based quantum processors presents an important opportunity to explore how the advantages quantum computing offers can be leveraged to create better machine learning models. Multivariate multiple regression models are used in special applications such as cancer research (Lee et al., 2012; Lee & Liu, 2012), agriculture (Johnson & Wichern, 2014), aviation (Rencher & Christensen, 2012), and defense (Filgöz et al., 2021; Kang, 2021; Yoon et al., 2010). The approach presented in the present study enables research in these domains using quantum neural networks.

This chapter discussed the implications and the limitations of the study. The quantum neural network model achieved a similar R^2 score as the classical neural network model with the drawback of its lengthy training time when trained on a quantum simulator. Training the model on quantum hardware was not feasible due to technological constraints. Recommendations for future research included exploring methods to improve the training process to reduce the total

run time. Additional suggestions included exploring the use of GPUs, exploring the effects of greater training iterations, and optimizing the quantum circuits using alternative approaches. The full training dataset and the source code for the MultiTargetQuantumNNRegressor were made available in this manuscript to promote further research and innovation in this new quantum computing era we have entered.

References

Abdi, H., & Williams, L. J. (2013). Partial least squares methods: Partial least squares correlation and partial least square regression. *Methods in Molecular Biology (Clifton, N.J.)*, *930*, 549–579. https://doi.org/10.1007/978-1-62703-059-5_23

Akin, M., Eyduran, S. P., Eyduran, E., & Reed, B. M. (2020). Analysis of macro nutrient related growth responses using multivariate adaptive regression splines. *Plant Cell, Tissue and Organ Culture (PCTOC)*, *140*(3), 661–670. https://doi.org/10.1007/s11240-019-01763-8

Alahassa, N. K.-A. (2021). State-of-the-art Multivariate Regression with a General Nk Hidden Multi-Layer Feed Forward Neural Network Model. *2021 International Conference on Artificial Intelligence and Computer Science Technology (ICAICST)*, 31–36. https://doi.org/10.1109/ICAICST53116.2021.9497838

Altaisky, M. V. (2001). *Quantum neural network* (arXiv:quant-ph/0107012). arXiv. https://doi.org/10.48550/arXiv.quant-ph/0107012

Amazon. (2022). *Quantum Computing Service—Amazon Braket–Amazon Web Services*. Amazon Web Services, Inc. https://aws.amazon.com/braket/

Amin, M. H., Andriyash, E., Rolfe, J., Kulchytskyy, B., & Melko, R. (2018). Quantum Boltzmann Machine. *Physical Review X*, *8*(2), 021050. https://doi.org/10.1103/PhysRevX.8.021050

Anaya-Morales, A., & Delgado, F. (2023). Enquiring Electronic Structure Using Quantum Computers: Hands on Qiskit. *Journal of Physics: Conference Series*, *2448*(1), 012014. https://doi.org/10.1088/1742-6596/2448/1/012014

Arute, F., Arya, K., Babbush, R., Bacon, D., Bardin, J. C., Barends, R., Biswas, R., Boixo, S., Brandao, F. G. S. L., Buell, D. A., Burkett, B., Chen, Y., Chen, Z., Chiaro, B., Collins,

R., Courtney, W., Dunsworth, A., Farhi, E., Foxen, B., … Martinis, J. M. (2019). Quantum supremacy using a programmable superconducting processor. *Nature*, *574*(7779), Article 7779. https://doi.org/10.1038/s41586-019-1666-5

Asolkar, J. (2020). *Data classification using Microsoft quantum machine learning library* [M.S., University of Dublin]. https://www.scss.tcd.ie/publications/theses/diss/2020/TCD-SCSS-DISSERTATION-2020-092.pdf

Awschalom, D. D., Du, C. R., He, R., Heremans, F. J., Hoffmann, A., Hou, J., Kurebayashi, H., Li, Y., Liu, L., Novosad, V., Sklenar, J., Sullivan, S. E., Sun, D., Tang, H., Tyberkevych, V., Trevillian, C., Tsen, A. W., Weiss, L. R., Zhang, W., … Zollitsch, Ch. W. (2021). Quantum Engineering With Hybrid Magnonic Systems and Materials (Invited Paper). *IEEE Transactions on Quantum Engineering*, *2*, 1–36. https://doi.org/10.1109/TQE.2021.3057799

Beer, K., Bondarenko, D., Farrelly, T., Osborne, T. J., Salzmann, R., Scheiermann, D., & Wolf, R. (2020). Training deep quantum neural networks. *Nature Communications*, *11*(1), 808. https://doi.org/10.1038/s41467-020-14454-2

Bera, S., & Mukherjee, I. (2018). A Solution Framework for Response Surface-Based Multiple Quality Characteristics Optimization. *International Journal of Reliability, Quality and Safety Engineering*, *25*(05), 1850025. https://doi.org/10.1142/S0218539318500250

Bernhardt, C. (2019). *Quantum computing for everyone*. The MIT Press.

Bhattacharyya, S., Bhattacharjee, S., & Mondal, N. K. (2015). A quantum backpropagation multilayer perceptron (QBMLP) for predicting iron adsorption capacity of calcareous soil from aqueous solution. *Applied Soft Computing*, *27*, 299–312. https://doi.org/10.1016/j.asoc.2014.11.019

Birkmire, B., & Gallagher, J. (2012, August 13). Air-to-Air Missile Maximum Launch Range Modeling using a Multilayer Perceptron. *AIAA Modeling and Simulation Technologies Conference*. AIAA Modeling and Simulation Technologies Conference, Minneapolis, Minnesota. https://doi.org/10.2514/6.2012-4942

Biswas, R., Jiang, Z., Kechezhi, K., Knysh, S., Mandrà, S., O'Gorman, B., Perdomo-Ortiz, A., Petukhov, A., Realpe-Gómez, J., Rieffel, E., Venturelli, D., Vasko, F., & Wang, Z. (2017). A NASA perspective on quantum computing: Opportunities and challenges. *Parallel Computing*, *64*, 81–98. https://doi.org/10.1016/j.parco.2016.11.002

Bluvstein, D., Levine, H., Semeghini, G., Wang, T. T., Ebadi, S., Kalinowski, M., Keesling, A., Maskara, N., Pichler, H., Greiner, M., Vuletić, V., & Lukin, M. D. (2022). A quantum processor based on coherent transport of entangled atom arrays. *Nature*, *604*(7906), Article 7906. https://doi.org/10.1038/s41586-022-04592-6

Borchani, H., Varando, G., Bielza, C., & Larrañaga, P. (2015). A survey on multi-output regression: Multi-output regression survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *5*(5), 216–233. https://doi.org/10.1002/widm.1157

Bozzo-Rey, M., Longbottom, J., & Müller, H. A. (2019). Quantum computing: Challenges and opportunities. *Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering*, 393–394.

Bravo, S., & Moreno, Á. H. (2019). Prediction Model Based on Neural Networks for Microwave Drying Process of Amaranth Seeds. *Proceedings of the 2019 3rd International Conference on Compute and Data Analysis*, 88–93. https://doi.org/10.1145/3314545.3314551

Breiman, L., & Friedman, J. H. (1997). Predicting Multivariate Responses in Multiple Linear Regression. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *59*(1), 3–54. https://doi.org/10.1111/1467-9868.00054

Carrascal, G., del Barrio, A. A., & Botella, G. (2021). First experiences of teaching quantum computing. *Journal of Supercomputing*, *77*(3), 2770–2799.

Carvalho, N. B., Minim, V. P. R., Silva, R. de C. dos S. N., Della Lucia, S. M., & Minim, L. A. (2013). Artificial neural networks (ANN): Prediction of sensory measurements from instrumental data. *Food Science and Technology (Campinas)*, *33*(4), 722–729. https://doi.org/10.1590/S0101-20612013000400018

Chakraborty, S., Gilyén, A., & Jeffery, S. (2018). The power of block-encoded matrix powers: Improved regression techniques via faster Hamiltonian simulation. *arXiv:1804.01973 [Quant-Ph]*. https://doi.org/10.4230/LIPIcs.ICALP.2019.33

Chen, H.-Y. (2023). *Deep Reinforcement Learning Using Hybrid Quantum Neural Network* (arXiv:2304.10159). arXiv. https://doi.org/10.48550/arXiv.2304.10159

Christensen, J. E., Hucul, D., Campbell, W. C., & Hudson, E. R. (2020). High-fidelity manipulation of a qubit enabled by a manufactured nucleus. *Npj Quantum Information*, *6*(1), Article 1. https://doi.org/10.1038/s41534-020-0265-5

Christensen, L. B., Johnson, B., & Turner, L. A. (2014). *Research methods, design, and analysis*. Pearson.

Cook, R. D., & Zhang, X. (2015). Simultaneous Envelopes for Multivariate Linear Regression. *Technometrics*, *57*(1), 11–25. https://doi.org/10.1080/00401706.2013.872700

Crnkovic, G. D. (2010). Constructive Research and Info-computational Knowledge Generation. In L. Magnani, W. Carnielli, & C. Pizzi (Eds.), *Model-Based Reasoning in Science and*

Technology (Vol. 314, pp. 359–380). Springer Berlin Heidelberg. http://link.springer.com/10.1007/978-3-642-15223-8_20

Dalal, A., Bagherimehrab, M., & Sanders, B. C. (2021). *Quantum-Assisted Support Vector Regression for Detecting Facial Landmarks* (arXiv:2111.09304). arXiv. https://doi.org/10.48550/arXiv.2111.09304

Dallaire-Demers, P.-L., & Killoran, N. (2018). Quantum generative adversarial networks. *Physical Review A*, *98*(1), 012324. https://doi.org/10.1103/PhysRevA.98.012324

Dargan, J. (2022, September 5). Quantum Computing Companies: Ultimate List for 2022. *The Quantum Insider*. https://thequantuminsider.com/2022/09/05/quantum-computing-companies-ultimate-list-for-2022/

Denzin, N. K., & Lincoln, Y. S. (1994). *Handbook of qualitative research* (pp. xii, 643). Sage Publications, Inc.

Deutsch, D., & Jozsa, R. (1992). Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, *439*(1907), 553–558. https://doi.org/10.1098/rspa.1992.0167

Deutsch, D., & Penrose, R. (1985). Quantum theory, the Church–Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, *400*(1818), 97–117. https://doi.org/10.1098/rspa.1985.0070

DiVincenzo, D. P. (1998). Quantum Gates and Circuits. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, *454*(1969), 261–276. https://doi.org/10.1098/rspa.1998.0159

Doreswamy, & Vastrad, C. M. (2013). Performance Analysis of Neural Network Models for Oxazolines and Oxazoles Derivatives Descriptor Dataset. *International Journal of Information Sciences and Techniques*, *3*(6), 1–15. https://doi.org/10.5121/ijist.2013.3601

Dutta, S., Suau, A., Dutta, S., Roy, S., Behera, B. K., & Panigrahi, P. K. (2020). Quantum Circuit Design Methodology for Multiple Linear Regression. *IET Quantum Communication*, *1*(2), 55–61. https://doi.org/10.1049/iet-qtc.2020.0013

Ennouri, K., Ben Ayed, R., Triki, M. A., Ottaviani, E., Mazzarello, M., Hertelli, F., & Zouari, N. (2017). Multiple linear regression and artificial neural networks for delta-endotoxin and protease yields modelling of Bacillus thuringiensis. *3 Biotech*, *7*(3), 187. https://doi.org/10.1007/s13205-017-0799-1

Farhi, E., & Neven, H. (2018). Classification with Quantum Neural Networks on Near Term Processors. *arXiv:1802.06002 [Quant-Ph]*. http://arxiv.org/abs/1802.06002

Feynman, R. P. (1982). Simulating physics with computers. *International Journal of Theoretical Physics*, *21*(6–7), 467–488. https://doi.org/10.1007/BF02650179

Feynman, R. P. (1986). Quantum mechanical computers. *Foundations of Physics*, *16*(6), 507–531. https://doi.org/10.1007/BF01886518

Filgöz, A., Demirezen, G., & Demirezen, M. U. (2021). Applying Novel Adaptive Activation Function Theory for Launch Acceptability Region Estimation with Neural Networks in Constrained Hardware Environments: Performance Comparison. *2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC)*, 1–10. https://doi.org/10.1109/DASC52595.2021.9594334

Friedman, J. H. (1991). Multivariate Adaptive Regression Splines. *The Annals of Statistics*, *19*(1), 1–67. JSTOR.

Garhwal, S., Ghorani, M., & Ahmad, A. (2021). Quantum Programming Language: A Systematic Review of Research Topic and Top Cited Languages. *Archives of Computational Methods in Engineering*, *28*(2), 289–310. https://doi.org/10.1007/s11831-019-09372-6

Gerling, L., Wiedensohler, A., & Weber, S. (2021). Statistical modelling of spatial and temporal variation in urban particle number size distribution at traffic and background sites. *Atmospheric Environment*, *244*, 117925. https://doi.org/10.1016/j.atmosenv.2020.117925

Google. (2022a). *Google Quantum Computing Service | Cirq*. Google Quantum AI. https://quantumai.google/cirq/google/concepts

Google. (2022b, July 19). *Our new Quantum Virtual Machine will accelerate research and help people learn quantum computing*. Google. https://blog.google/technology/research/our-new-quantum-virtual-machine-will-accelerate-research-and-help-people-learn-quantum-computing/

Goyal, S., & Goyal, G. K. (2012). Shelf Life Estimation of Processed Cheese by Artificial Neural Network Expert Systems. *Journal of Advanced Computer Science & Technology*, *1*(1), 32–41. https://doi.org/10.14419/jacst.v1i1.10

Griffiths, D. J., & Schroeter, D. F. (2018). *Introduction to quantum mechanics* (Third edition). Cambridge University Press.

Grover, L. K. (1996). A fast quantum mechanical algorithm for database search. *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing - STOC '96*, 212–219. https://doi.org/10.1145/237814.237866

Guillaume, A., Goh, E. Y., Johnston, M. D., Wilson, B. D., Ramanan, A., Tibble, F., & Lackey, B. (2022). Deep Space Network Scheduling Using Quantum Annealing. *IEEE*

Transactions on Quantum Engineering, *3*, 1–13. https://doi.org/10.1109/TQE.2022.3199267

Harrow, A. W., Hassidim, A., & Lloyd, S. (2009). Quantum Algorithm for Linear Systems of Equations. *Physical Review Letters*, *103*(15), 150502. https://doi.org/10.1103/PhysRevLett.103.150502

Hastie, T., Tibshirani, R., & Friedman, J. H. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (2nd ed). Springer.

Havlicek, V., Córcoles, A. D., Temme, K., Harrow, A. W., Kandala, A., Chow, J. M., & Gambetta, J. M. (2019). Supervised learning with quantum enhanced feature spaces. *Nature*, *567*(7747), 209–212. https://doi.org/10.1038/s41586-019-0980-2

Havlíček, V., Córcoles, A. D., Temme, K., Harrow, A. W., Kandala, A., Chow, J. M., & Gambetta, J. M. (2019). Supervised learning with quantum-enhanced feature spaces. *Nature*, *567*(7747), Article 7747. https://doi.org/10.1038/s41586-019-0980-2

Helland, I. S. (1990). Partial Least Squares Regression and Statistical Models. *Scandinavian Journal of Statistics*, *17*(2), 97–114. JSTOR.

Hevner, A., & Chatterjee, S. (2010). *Design Research in Information Systems* (Vol. 22). Springer US. http://link.springer.com/10.1007/978-1-4419-5653-8

Ho, A., McClean, J., & Ong, S. P. (2018). The Promise and Challenges of Quantum Computing for Energy Storage. *Joule*, *2*(5), 810–813. https://doi.org/10.1016/j.joule.2018.04.021

Höskuldsson, A. (1988). PLS regression methods. *Journal of Chemometrics*, *2*(3), 211–228. https://doi.org/10.1002/cem.1180020306

Huang, H.-L., Du, Y., Gong, M., Zhao, Y., Wu, Y., Wang, C., Li, S., Liang, F., Lin, J., Xu, Y., Yang, R., Liu, T., Hsieh, M.-H., Deng, H., Rong, H., Peng, C.-Z., Lu, C.-Y., Chen, Y.-A.,

Tao, D., … Pan, J.-W. (2020). Experimental Quantum Generative Adversarial Networks for Image Generation. *arXiv:2010.06201 [Quant-Ph]*. http://arxiv.org/abs/2010.06201

IBM. (2022). *IBM Quantum Computing*. https://www.ibm.com/www.ibm.com/quantum

IQT. (2022, January 18). *IQT Research Report Projects Expenditures on Quantum Computing by Banks and other Financial Institution to Reach $631.8 million by 2028*. https://www.prnewswire.com/news-releases/iqt-research-report-projects-expenditures-on-quantum-computing-by-banks-and-other-financial-institution-to-reach-631-8-million-by-2028--301462292.html

Jiang, W., Xiong, J., & Shi, Y. (2021). When machine learning meets quantum computers: A case study. *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, 593–598. https://doi.org/10.1145/3394885.3431629

Johnson, R. A., & Wichern, D. W. (2007). *Applied multivariate statistical analysis* (6th ed). Pearson Prentice Hall.

Johnson, R. A., & Wichern, D. W. (2014). *Applied multivariate statistical analysis* (6. ed., new internat. ed). Pearson Education Limited.

Jolliffe, I. T. (2002). *Principal component analysis* (2nd ed). Springer.

Jordan, S. (2022). *Quantum Algorithm Zoo*. https://quantumalgorithmzoo.org/

Kairon, P., & Bhattacharyya, S. (2020). Comparative study of variational quantum circuit and quantum backpropagation multilayer perceptron for COVID-19 outbreak predictions. *arXiv:2008.07617 [Quant-Ph]*. http://arxiv.org/abs/2008.07617

Kak, S. C. (1995). Quantum Neural Computing. In P. W. Hawkes (Ed.), *Advances in Imaging and Electron Physics* (Vol. 94, pp. 259–313). Elsevier. https://doi.org/10.1016/S1076-5670(08)70147-2

Kamruzzaman, A., Alhwaiti, Y., Leider, A., & Tappert, C. C. (2020). Quantum Deep Learning Neural Networks. In K. Arai & R. Bhatia (Eds.), *Advances in Information and Communication* (Vol. 70, pp. 299–311). Springer International Publishing. https://doi.org/10.1007/978-3-030-12385-7_24

Kang, Y. (2021). Computation of Launch Acceptability Region of Air-to-Surface Guided Bomb for Moving Target. *Journal of the Korean Society for Aeronautical & Space Sciences*, *49*(7), 601–608. https://doi.org/10.5139/JKSAS.2021.49.7.601

Keeler, G. A. (2022). Heterogeneous Photonics and Optical Microsystem Applications at DARPA. *2022 27th OptoElectronics and Communications Conference (OECC) and 2022 International Conference on Photonics in Switching and Computing (PSC)*, 01–01. https://doi.org/10.23919/OECC/PSC53152.2022.9850038

Keijsers, N. L. W. (2010). Neural Networks. In K. Kompoliti & L. V. Metman (Eds.), *Encyclopedia of Movement Disorders* (pp. 257–259). Academic Press. https://doi.org/10.1016/B978-0-12-374105-9.00493-7

Kingma, D. P., & Lei, J. (2015). *Adam: A Method for Stochastic Optimization*. 15.

Kitaev, A. J., Šen, A. C., Vjalyj, M. N., & Kitaev, A. J. (2002). *Classical and quantum computation*. American Mathematical Society.

Koch, D., Samodurov, M., Projansky, A., & Alsing, P. M. (2021). *Gate-Based Circuit Designs For Quantum Adder Inspired Quantum Random Walks on Superconducting Qubits* (arXiv:2012.10268). arXiv. https://doi.org/10.48550/arXiv.2012.10268

Kraft, D. (1988). A software package for sequential quadratic programming. *Forschungsbericht-Deutsche Forschungs- Und Versuchsanstalt Fur Luft- Und Raumfahrt*.

Lee, W., Du, Y., Sun, W., Hayes, D. N., & Liu, Y. (2012). Multiple Response Regression for Gaussian Mixture Models with Known Labels. *Statistical Analysis and Data Mining*, *5*(6), 10.1002/sam.11158. https://doi.org/10.1002/sam.11158

Lee, W., & Liu, Y. (2012). Simultaneous Multiple Response Regression and Inverse Covariance Matrix Estimation via Penalized Gaussian Maximum Likelihood. *Journal of Multivariate Analysis*, *111*, 241–255. https://doi.org/10.1016/j.jmva.2012.03.013

Lewenstein, M. (1994). Quantum Perceptrons. *Journal of Modern Optics*, *41*(12), 2491–2501. https://doi.org/10.1080/09500349414552331

Li, J., Topaloglu, R., & Ghosh, S. (2021). Quantum Generative Models for Small Molecule Drug Discovery. *IEEE Transactions on Quantum Engineering*, 1–1. https://doi.org/10.1109/TQE.2021.3104804

Liu, D. C., & Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, *45*(1), 503–528. https://doi.org/10.1007/BF01589116

Lloyd, S., & Weedbrook, C. (2018). Quantum generative adversarial learning. *Physical Review Letters*, *121*(4), 040502. https://doi.org/10.1103/PhysRevLett.121.040502

Lundeen, J. S., Sutherland, B., Patel, A., Stewart, C., & Bamber, C. (2011). Direct measurement of the quantum wavefunction. *Nature*, *474*(7350), Article 7350. https://doi.org/10.1038/nature10120

Macaluso, A., Clissa, L., Lodi, S., & Sartori, C. (2020a). A Variational Algorithm for Quantum Neural Networks. In V. V. Krzhizhanovskaya, G. Závodszky, M. H. Lees, J. J. Dongarra, P. M. A. Sloot, S. Brissos, & J. Teixeira (Eds.), *Computational Science – ICCS 2020* (pp.

591–604). Springer International Publishing. https://doi.org/10.1007/978-3-030-50433-5_45

Macaluso, A., Clissa, L., Lodi, S., & Sartori, C. (2020b). Quantum Ensemble for Classification. *arXiv:2007.01028 [Quant-Ph, Stat]*. http://arxiv.org/abs/2007.01028

Malhotra, Y. (2020). *Communications: Making Quantum Computing Real for JADC2 With Qiskit: Quantum Enabling Technologies for Applications to Support Communication and Networking: White Paper Submitted for the Global Competition: AFRL Innovare Million Dollar International Quantum U Tech Accelerator* (SSRN Scholarly Paper 3682216). https://doi.org/10.2139/ssrn.3682216

March, S. T., & Smith, G. F. (1995). Design and natural science research on information technology. *Decision Support Systems*, *15*(4), 251–266. https://doi.org/10.1016/0167-9236(94)00041-2

Markidis, S. (2023). Programming Quantum Neural Networks on NISQ Systems: An Overview of Technologies and Methodologies. *Entropy*, *25*(4), Article 4. https://doi.org/10.3390/e25040694

McCaig, X. P. (2021). *Quantum Algorithms and Their Cryptographic Applications* [M.S., University of Nebraska at Omaha]. http://www.proquest.com/pqdtglobal/docview/2558112650/abstract/78B29180D1454BA5PQ/1

Menzies, T., Kocagüneli, E., Minku, L., Peters, F., & Turhan, B. (2015). Chapter 24—Using Goals in Model-Based Reasoning. In T. Menzies, E. Kocagüneli, L. Minku, F. Peters, & B. Turhan (Eds.), *Sharing Data and Models in Software Engineering* (pp. 321–353). Morgan Kaufmann. https://doi.org/10.1016/B978-0-12-417295-1.00024-2

Mermin, N. D. (2007). *Quantum computer science: An introduction*. Cambridge University Press.

Microsoft. (2022). *Azure Quantum—Quantum Cloud Computing Service | Microsoft Azure*. https://azure.microsoft.com/en-us/products/quantum/

Mohamed, Z. E. (2019). Using the artificial neural networks for prediction and validating solar radiation. *Journal of the Egyptian Mathematical Society*, *27*(1), 47. https://doi.org/10.1186/s42787-019-0043-8

Moosavi, A., Stefanescu, R., & Sandu, A. (2017). Multivariate predictions of local reduced-order-model errors and dimensions. *arXiv:1701.03720 [Cs]*. http://arxiv.org/abs/1701.03720

Moran, C. C. (2019). *Mastering quantum computing with IBM QX: Explore the world of quantum computing using the Quantum Composer and Qiskit*. Packt Publishing.

Nielsen, M. A., & Chuang, I. L. (2010). *Quantum computation and quantum information* (10th anniversary ed). Cambridge University Press.

Nikolenko, S. I. (2021). *Synthetic data for deep learning*. Springer. https://doi.org/10.1007/978-3-030-75178-4

NIST. (2022, July 5). *Announcing PQC Candidates to be Standardized, Plus Fourth Round Candidates | CSRC*. CSRC | NIST. https://csrc.nist.gov/News/2022/pqc-candidates-to-be-standardized-and-round-4

Niu, M. Y., Zlokapa, A., Broughton, M., Boixo, S., Mohseni, M., Smelyanskyi, V., & Neven, H. (2021a). Entangling Quantum Generative Adversarial Networks. *arXiv:2105.00080 [Quant-Ph]*. http://arxiv.org/abs/2105.00080

Niu, M. Y., Zlokapa, A., Broughton, M., Boixo, S., Mohseni, M., Smelyanskyi, V., & Neven, H. (2021b). Entangling Quantum Generative Adversarial Networks. *arXiv:2105.00080 [Quant-Ph]*. http://arxiv.org/abs/2105.00080

Park, D. K., Petruccione, F., & Rhee, J.-K. K. (2019). Circuit-Based Quantum Random Access Memory for Classical Data. *Scientific Reports*, *9*(1), Article 1. https://doi.org/10.1038/s41598-019-40439-3

Patel, T., & Tiwari, D. (2020). DisQ: A novel quantum output state classification method on IBM quantum computers using openpulse. *Proceedings of the 39th International Conference on Computer-Aided Design*, 1–9. https://doi.org/10.1145/3400302.3415619

Pattanayak, S. (2021). *Quantum Machine Learning with Python: Using Cirq from Google Research and IBM Qiskit*. Apress. https://doi.org/10.1007/978-1-4842-6522-2

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of Management Information Systems*, *24*(3), 45–77. https://doi.org/10.2753/MIS0742-1222240302

Potempa, R., & Porebski, S. (2022). Comparing concepts of quantum and classical neural network models for image classification task. *arXiv:2108.08875 [Cs]*, *255*, 61–71. https://doi.org/10.1007/978-3-030-81523-3_6

Powell, M. J. D. (1994). A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation. In S. Gomez & J.-P. Hennart (Eds.),

Advances in Optimization and Numerical Analysis (pp. 51–67). Springer Netherlands. https://doi.org/10.1007/978-94-015-8330-5_4

Preskill, J. (2018). Quantum Computing in the NISQ era and beyond. *Quantum*, *2*, 79. https://doi.org/10.22331/q-2018-08-06-79

Qi, J., Du, J., Siniscalchi, S. M., Ma, X., & Lee, C.-H. (2020). On Mean Absolute Error for Deep Neural Network Based Vector-to-Vector Regression. *IEEE Signal Processing Letters*, *27*, 1485–1489. https://doi.org/10.1109/LSP.2020.3016837

Qiskit contributors. (2023). *Qiskit: An Open-source Framework for Quantum Computing*. https://doi.org/10.5281/zenodo.2573505

Quoc, C. N., Ho, L. B., Tran, L. N., & Nguyen, H. Q. (2021). Qsun: An open-source platform towards practical quantum machine learning applications. *arXiv:2107.10541 [Quant-Ph]*. http://arxiv.org/abs/2107.10541

Raymer, M. G., & Monroe, C. (2019). The US National Quantum Initiative. *Quantum Science and Technology*, *4*(2), 020504. https://doi.org/10.1088/2058-9565/ab0441

Rebentrost, P., Mohseni, M., & Lloyd, S. (2014). Quantum Support Vector Machine for Big Data Classification. *Physical Review Letters*, *113*(13), 130503. https://doi.org/10.1103/PhysRevLett.113.130503

Reddy, P., & Bhattacherjee, A. B. (2021). A hybrid quantum regression model for the prediction of molecular atomization energies. *Machine Learning: Science and Technology*, *2*(2), 025019. https://doi.org/10.1088/2632-2153/abd486

Rencher, A. C., & Christensen, W. F. (2012). Multivariate Regression. In *Methods of Multivariate Analysis* (pp. 339–383). John Wiley & Sons, Inc. https://doi.org/10.1002/9781118391686.ch10

Rieffel, E. G., Hadfield, S., Hogg, T., Mandrà, S., Marshall, J., Mossi, G., O'Gorman, B., Plamadeala, E., Tubman, N. M., Venturelli, D., Vinci, W., Wang, Z., Wilson, M., Wudarski, F., & Biswas, R. (2019). From Ansätze to Z-gates: A NASA View of quantum computing. *arXiv:1905.02860 [Quant-Ph]*. https://doi.org/10.3233/APC190010

Rimal, R., Almøy, T., & Sæbø, S. (2019). Comparison of Multi-response Prediction Methods. *Chemometrics and Intelligent Laboratory Systems*, *190*, 10–21. https://doi.org/10.1016/j.chemolab.2019.05.004

Rivero, P., Cloët, I. C., & Sullivan, Z. (2020). *An optimal quantum sampling regression algorithm for variational eigensolving in the low qubit number regime* (arXiv:2012.02338). arXiv. https://doi.org/10.48550/arXiv.2012.02338

Robbins, H., & Monro, S. (1951). A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, *22*(3), 400–407. https://doi.org/10.1214/aoms/1177729586

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), Article 6088. https://doi.org/10.1038/323533a0

Sahlol, A., Hemdan, A. M., & Hassanien, A. E. (2017). Prediction of Antioxidant Status in Fish Farmed on Selenium Nanoparticles using Neural Network Regression Algorithm. In A. E. Hassanien, K. Shaalan, T. Gaber, A. T. Azar, & M. F. Tolba (Eds.), *Proceedings of the International Conference on Advanced Intelligent Systems and Informatics 2016* (Vol. 533, pp. 353–364). Springer International Publishing. https://doi.org/10.1007/978-3-319-48308-5_34

Saini, S., Khosla, P., Kaur, M., & Singh, G. (2020). Quantum driven machine learning. *International Journal of Theoretical Physics*, *59*(12), 4013–4024. https://doi.org/10.1007/s10773-020-04656-1

Savchenkov, A. A., Christensen, J. E., Hucul, D., Campbell, W. C., Hudson, E. R., Williams, S., & Matsko, A. B. (2020). Application of a self-injection locked cyan laser for Barium ion cooling and spectroscopy. *Scientific Reports*, *10*(1), Article 1. https://doi.org/10.1038/s41598-020-73373-w

Schuld, M., Bocharov, A., Svore, K. M., & Wiebe, N. (2020). Circuit-centric quantum classifiers. *Physical Review A*, *101*(3), 032308. https://doi.org/10.1103/PhysRevA.101.032308

Schuld, M., Sinayskiy, I., & Petruccione, F. (2014a). The quest for a Quantum Neural Network. *Quantum Information Processing*, *13*(11), 2567–2586. https://doi.org/10.1007/s11128-014-0809-8

Schuld, M., Sinayskiy, I., & Petruccione, F. (2014b). The quest for a Quantum Neural Network. *Quantum Information Processing*, *13*(11), 2567–2586. https://doi.org/10.1007/s11128-014-0809-8

Schuld, M., Sinayskiy, I., & Petruccione, F. (2015). An introduction to quantum machine learning. *Contemporary Physics*, *56*(2), 172–185. https://doi.org/10.1080/00107514.2014.964942

Shao, Y., Dietrich, F. M., Nettelblad, C., & Zhang, C. (2021). Training algorithm matters for the performance of neural network potential: A case study of Adam and the Kalman filter optimizers. *The Journal of Chemical Physics*, *155*(20), 204108. https://doi.org/10.1063/5.0070931

Sharma, S. (2020). *QEML: (Quantum Enhanced Machine Learning)*. 9.

Shor, P. W. (1994). Algorithms for quantum computation: Discrete logarithms and factoring. *Proceedings 35th Annual Symposium on Foundations of Computer Science*, 124–134. https://doi.org/10.1109/SFCS.1994.365700

Sim, S., Johnson, P. D., & Aspuru-Guzik, A. (2019). Expressibility and Entangling Capability of Parameterized Quantum Circuits for Hybrid Quantum-Classical Algorithms. *Advanced Quantum Technologies*, *2*(12), 1900070. https://doi.org/10.1002/qute.201900070

Siomau, M. (2014). A Quantum Model for Autonomous Learning Automata. *Quantum Information Processing*, *13*(5), 1211–1221. https://doi.org/10.1007/s11128-013-0723-5

Stangierski, J., Weiss, D., & Kaczmarek, A. (2019). Multiple regression models and Artificial Neural Network (ANN) as prediction tools of changes in overall quality during the storage of spreadable processed Gouda cheese. *European Food Research and Technology*, *245*(11), 2539–2547. https://doi.org/10.1007/s00217-019-03369-y

Stein, J., Poppel, M., Adamczyk, P., Fabry, R., Wu, Z., Kölle, M., Nüßlein, J., Schuman, D., Altmann, P., Ehmer, T., Narasimhan, V., & Linnhoff-Popien, C. (2023). *Quantum Surrogate Modeling for Chemical and Pharmaceutical Development* (arXiv:2306.05042). arXiv. http://arxiv.org/abs/2306.05042

Stein, S. A., Baheri, B., Chen, D., Mao, Y., Guan, Q., Li, A., Fang, B., & Xu, S. (2021). QuGAN: A Generative Adversarial Network Through Quantum States. *arXiv:2010.09036 [Quant-Ph]*. http://arxiv.org/abs/2010.09036

Sun, H., Burton, H. V., & Huang, H. (2021). Machine learning applications for building structural design and performance assessment: State-of-the-art review. *Journal of Building Engineering*, *33*, 101816. https://doi.org/10.1016/j.jobe.2020.101816

Tehrani, M., Sultanow, E., Buchanan, W. J., Amir, M., Jeschke, A., Chow, R., & Lemoudden, M. (2023). *Enabling Quantum Cybersecurity Analytics in Botnet Detection: Stable Architecture and Speed-up through Tree Algorithms* (arXiv:2306.13727). arXiv. https://doi.org/10.48550/arXiv.2306.13727

Tilly, J., Chen, H., Cao, S., Picozzi, D., Setia, K., Li, Y., Grant, E., Wossnig, L., Rungger, I., Booth, G. H., & Tennyson, J. (2022). *The Variational Quantum Eigensolver: A review of methods and best practices*. https://doi.org/10.1016/j.physrep.2022.08.003

Vlasov, A. Y. (1997). *Quantum Computations and Images Recognition* (arXiv:quant-ph/9703010). arXiv. https://doi.org/10.48550/arXiv.quant-ph/9703010

Watabe, M., Shiba, K., Chen, C.-C., Sogabe, M., Sakamoto, K., & Sogabe, T. (2021). Quantum Circuit Learning with Error Backpropagation Algorithm and Experimental Implementation. *Quantum Reports*, *3*(2), 333–349. https://doi.org/10.3390/quantum3020021

Watabe, M., Shiba, K., Sogabe, M., Sakamoto, K., & Sogabe, T. (2021). *Quantum Circuit Parameters Learning with Gradient Descent Using Backpropagation*. 11.

Wiebe, N. (2020). Key questions for the quantum machine learner to ask themselves. *New Journal of Physics*, *22*(9), 091001. https://doi.org/10.1088/1367-2630/abac39

Wold, H. (1975). Soft Modelling by Latent Variables: The Non-Linear Iterative Partial Least Squares (NIPALS) Approach. *Journal of Applied Probability*, *12*(S1), 117–142. https://doi.org/10.1017/S0021900200047604

Wold, S., Ruhe, A., Wold, H., & Dunn, III, W. J. (1984). The Collinearity Problem in Linear Regression. The Partial Least Squares (PLS) Approach to Generalized Inverses. *SIAM*

Journal on Scientific and Statistical Computing, *5*(3), 735–743. https://doi.org/10.1137/0905052

Wolf, S. A., Joneckis, L. G., Waruhiu, S., Biddle, J. C., Sun, O. S., & Buckley, L. J. (2019). *Overview of the Status of Quantum Science and Technology and Recommendations for the DoD* (Overview of the Status of Quantum Science and Technology and Recommendations for the DoD, pp. 51–54). Institute for Defense Analyses; JSTOR. http://www.jstor.org/stable/resrep22809.8

Wong, T. G. (2022). *Introduction to Classical and Quantum Computing*. Rooted Grove.

Xu, D., Shi, Y., Tsang, I. W., Ong, Y.-S., Gong, C., & Shen, X. (2019). A Survey on Multi-output Learning. *arXiv:1901.00248 [Cs, Stat]*. http://arxiv.org/abs/1901.00248

Yoon, K. S., Park, J. H., Kim, I. G., & Ryu, K. S. (2010). New modeling algorithm for improving accuracy of weapon launch acceptability region. *29th Digital Avionics Systems Conference*, 6.D.4-1-6.D.4-6. https://doi.org/10.1109/DASC.2010.5655454

Zhao, C., & Gao, X.-S. (2021). QDNN: Deep neural networks with quantum layers. *Quantum Machine Intelligence*, *3*(1), 15. https://doi.org/10.1007/s42484-021-00046-w

Zhao, R., & Wang, S. (2021). *A review of Quantum Neural Networks: Methods, Models, Dilemma* (arXiv:2109.01840). arXiv. http://arxiv.org/abs/2109.01840

Zhong, Y., Chang, H.-S., Bienfait, A., Dumur, É., Chou, M.-H., Conner, C. R., Grebel, J., Povey, R. G., Yan, H., Schuster, D. I., & Cleland, A. N. (2021). Deterministic multi-qubit entanglement in a quantum network. *Nature*, *590*(7847), Article 7847. https://doi.org/10.1038/s41586-021-03288-7

Zhu, C., Byrd, R. H., Lu, P., & Nocedal, J. (1997). Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software*, *23*(4), 550–560. https://doi.org/10.1145/279232.279236

Appendix A

Training Data

Table A1

Training Data with 100 Samples

	x_1	x_2	x_3	y_1	y_2
0	0.299688	0.537735	0.645746	0.399578	0.600950
1	0.331838	0.644123	0.293120	0.253450	0.325425
2	0.295416	0.358360	0.811418	0.457989	0.717744
3	0.536015	0.356220	0.477311	0.476763	0.551227
4	0.233822	0.359595	0.549973	0.268055	0.458148
5	0.424951	0.127427	0.504187	0.366132	0.484670
6	0.825019	0.831444	0.535754	0.815410	0.819303
7	0.431724	0.575225	0.922880	0.661423	0.916254
8	0.405542	0.548875	0.630376	0.479890	0.644081
9	0.415536	0.525178	0.475357	0.401833	0.510853
10	0.178638	0.396585	0.512814	0.208401	0.402351
11	0.734306	0.863788	0.360597	0.652244	0.624169
12	0.176319	0.448346	0.529194	0.222897	0.422731
13	0.450636	0.154206	0.677332	0.483602	0.652648
14	0.332218	0.862127	0.721518	0.514694	0.729683
15	0.522013	0.635610	0.526645	0.532923	0.626411
16	0.354993	0.854979	0.442413	0.383331	0.497311
17	0.358555	0.677238	0.599057	0.443593	0.610535
18	0.433278	0.795704	0.294062	0.359615	0.400315
19	0.339762	0.419308	0.328639	0.245669	0.328727
20	0.453303	0.339304	0.401092	0.365695	0.439492
21	0.526356	0.636870	0.407958	0.473315	0.525433
22	0.332796	0.609040	0.499980	0.359478	0.501204
23	0.431508	0.528849	0.359171	0.353459	0.418429
24	1.000000	0.602015	0.676092	1.000000	1.000000
25	0.536453	0.522101	0.464276	0.494694	0.563530
26	0.273922	0.592116	0.570281	0.346188	0.529505
27	0.555474	0.401090	0.643013	0.587825	0.712043
28	0.614291	0.184540	0.729634	0.650313	0.787457
29	0.504820	0.597847	0.558503	0.530237	0.639911
30	0.621484	0.669702	0.419710	0.562511	0.589690
31	0.428502	0.570651	0.424305	0.391945	0.479526
32	0.184627	0.421459	0.595543	0.261158	0.481056
33	0.091866	1.000000	0.697487	0.325007	0.603457
34	0.261540	0.584934	0.722814	0.416394	0.654966
35	0.697807	0.577356	0.731350	0.777855	0.887797
36	0.368577	0.848411	0.541244	0.446268	0.589547
37	0.370812	0.498983	0.225651	0.227954	0.266366

38	0.502029	0.551604	0.729695	0.612497	0.781089
39	0.520926	0.191261	0.789084	0.606422	0.791739
40	0.170008	0.994681	0.769221	0.426643	0.705770
41	0.631195	0.254896	0.329192	0.460818	0.457264
42	0.785323	0.470107	0.594230	0.760617	0.798602
43	0.246483	0.462048	0.000000	0.000000	0.000000
44	0.415345	0.353662	0.304288	0.284990	0.337471
45	0.359685	0.568303	0.475281	0.362335	0.487887
46	0.291910	0.301947	0.705709	0.390338	0.615850
47	0.304079	0.529426	0.591592	0.373044	0.554872
48	0.439894	0.675191	0.361848	0.383411	0.445788
49	0.499256	0.382567	0.461551	0.442079	0.522136
50	0.402666	0.329680	0.916777	0.598008	0.861165
51	0.420148	0.234752	0.481462	0.365930	0.477538
52	0.546182	0.279196	0.493079	0.482134	0.559362
53	0.460030	0.490791	0.377668	0.381112	0.443975
54	0.379833	0.392910	0.438989	0.333559	0.441947
55	0.138251	0.535653	0.239022	0.049661	0.162476
56	0.296284	0.500281	0.757734	0.451029	0.691468
57	0.431228	0.524642	0.716304	0.543258	0.728861
58	0.537066	0.106747	0.503022	0.454459	0.538932
59	0.457460	0.277409	0.569512	0.449862	0.579651
60	0.743178	0.134334	0.689016	0.726977	0.811878
61	0.567564	0.613489	0.778325	0.701391	0.866223
62	0.262700	0.217231	0.668630	0.334045	0.556413
63	0.399389	0.429617	0.637664	0.461096	0.630390
64	0.267321	0.416671	0.447726	0.249403	0.394512
65	0.329763	0.287904	0.540765	0.331272	0.489799
66	0.584461	0.828451	0.324951	0.505017	0.510334
67	0.220695	0.932088	0.668673	0.405307	0.635632
68	0.116258	0.496569	1.000000	0.432072	0.808581
69	0.000000	0.467834	0.865006	0.260349	0.626552
70	0.482772	0.283095	0.562738	0.467860	0.587692
71	0.484325	0.140238	0.414592	0.368923	0.439236
72	0.568077	0.300838	0.500480	0.507254	0.580234
73	0.294384	0.533282	0.636139	0.389438	0.589198
74	0.324679	0.563765	0.849392	0.532650	0.795038
75	0.552145	0.615957	0.787777	0.694148	0.866803
76	0.226923	0.343675	0.603354	0.288535	0.498828
77	0.238240	0.742435	0.594959	0.352309	0.553720
78	0.677515	0.360628	0.465798	0.587392	0.615270
79	0.239298	0.629330	0.429111	0.247916	0.393788
80	0.507051	0.382510	0.603821	0.524417	0.650134
81	0.213642	0.540028	0.724108	0.371136	0.624886
82	0.275092	0.728287	0.751008	0.463765	0.706818
83	0.406760	0.544131	0.732272	0.534585	0.732826
84	0.476147	0.479079	0.406729	0.408121	0.476008

85	0.546562	0.575452	0.356445	0.453315	0.482361
86	0.543740	0.563742	0.458674	0.503841	0.568314
87	0.454823	0.560471	0.266063	0.327564	0.353873
88	0.484052	0.556841	0.488039	0.469513	0.561941
89	0.656960	0.414919	0.358803	0.521434	0.519045
90	0.372597	0.545181	0.471311	0.367392	0.487863
91	0.199546	0.527572	0.230672	0.094312	0.185877
92	0.553594	0.759438	0.435676	0.528590	0.581037
93	0.216131	0.218352	0.731437	0.329522	0.587122
94	0.108301	0.539458	0.569990	0.202328	0.435840
95	0.515065	0.594370	0.543169	0.529944	0.631377
96	0.460624	0.459000	0.764013	0.583144	0.776416
97	0.642379	0.319082	0.648436	0.649912	0.750295
98	0.495647	0.000000	0.556070	0.433002	0.548572
99	0.353071	0.687945	0.367350	0.316982	0.407314

Table A2

Training Data with 500 Samples

	x_1	x_2	x_3	y_1	y_2
0	0.466708	0.406284	0.756761	0.506264	0.692422
1	0.599563	0.681620	0.621540	0.719438	0.656702
2	0.403828	0.343923	0.642646	0.390797	0.555567
3	0.552279	0.408136	0.562953	0.433086	0.556311
4	0.442120	0.687416	0.522031	0.662972	0.497405
5	0.708124	0.642916	0.604284	0.688388	0.685865
6	0.670105	0.303502	0.412218	0.283148	0.463040
7	0.370560	0.534450	0.896580	0.675901	0.787825
8	0.377079	0.457586	0.465234	0.416074	0.394936
9	0.663015	0.284996	0.328706	0.228490	0.382893
10	0.138745	0.501068	0.575936	0.477408	0.392445
11	0.341669	0.271842	0.747138	0.362057	0.614903
12	0.381547	0.417811	0.597512	0.437394	0.512159
13	0.516352	0.405741	0.458581	0.380976	0.446077
14	0.338986	0.663227	0.871522	0.781428	0.763671
15	0.542528	0.358629	0.648801	0.423341	0.624442
16	0.456564	0.781267	0.148709	0.588776	0.176860
17	0.602574	0.665389	0.484316	0.644560	0.532929
18	0.471968	0.352592	0.426220	0.312051	0.391975
19	0.452615	0.423569	0.996072	0.625534	0.903260
20	0.421340	0.587955	0.727205	0.657576	0.663195
21	0.310500	0.353967	0.361142	0.266067	0.261451
22	0.679088	0.561712	0.532788	0.577932	0.600656
23	0.365203	0.622044	0.602262	0.628155	0.528962
24	0.689323	0.507311	0.400492	0.470434	0.480839
25	0.479781	0.618805	0.290686	0.502035	0.299277
26	0.379939	0.415792	0.467346	0.378331	0.394058
27	0.686935	0.310252	0.639570	0.390947	0.675891
28	0.529267	0.410486	0.539793	0.422462	0.525417
29	0.479536	0.536647	0.619423	0.569273	0.587154
30	0.468317	0.446237	0.600937	0.475508	0.556730
31	0.429087	0.477885	0.689124	0.539091	0.621686
32	0.456067	0.580823	0.225585	0.435331	0.226396
33	0.455084	0.503730	0.967806	0.688248	0.886694
34	0.435491	0.417878	0.565255	0.429605	0.507202
35	0.371459	0.605238	0.866142	0.728736	0.767693
36	0.635385	0.430999	0.723477	0.534366	0.740142
37	0.458223	0.463928	0.449557	0.424564	0.417655
38	0.436387	0.370879	0.526667	0.368957	0.468300
39	0.358595	0.455437	0.948719	0.623607	0.821757
40	0.561676	0.433457	0.381259	0.378254	0.399384
41	0.554911	0.547250	0.494082	0.533052	0.508981
42	0.530689	0.799620	0.414409	0.730853	0.450934
43	0.486824	0.613126	0.585482	0.626626	0.567272
44	0.416973	0.086134	0.479099	0.080162	0.389176
45	0.208994	0.810491	0.427515	0.709332	0.320205
46	0.481378	0.442549	0.350165	0.363788	0.336433

47	0.482387	0.504058	0.217838	0.363366	0.223719
48	0.552426	0.606951	0.631882	0.648808	0.637728
49	0.427400	0.485463	0.551836	0.485856	0.498068
50	0.394695	0.452860	0.826275	0.571789	0.727384
51	0.705140	0.578262	0.776465	0.703096	0.833272
52	0.405076	0.416364	0.747428	0.504418	0.657489
53	0.452945	0.398017	0.623436	0.438575	0.565445
54	0.654589	0.570028	0.411094	0.529562	0.480968
55	0.489248	0.573278	0.908151	0.731000	0.854985
56	0.204432	0.634816	0.571930	0.608100	0.431137
57	0.504135	0.334100	0.681514	0.410311	0.634377
58	0.511112	0.473443	0.829301	0.605857	0.784066
59	0.103734	0.355985	0.374234	0.249640	0.181147
60	0.268849	0.408910	0.748118	0.481926	0.596585
61	0.115672	0.710007	0.728230	0.736379	0.539534
62	0.000000	0.634948	0.461223	0.535982	0.240236
63	0.515974	0.685518	0.754634	0.771630	0.739596
64	0.743377	0.316994	0.818536	0.482158	0.862859
65	0.176080	0.472124	0.342125	0.352369	0.195800
66	0.710697	0.414151	0.336532	0.357983	0.423756
67	0.320599	0.533886	0.341445	0.426508	0.265685
68	0.397620	0.539882	0.664635	0.582563	0.591611
69	0.296679	0.520346	0.409867	0.441051	0.315296
70	0.427153	0.522217	0.998840	0.715839	0.903962
71	0.038426	0.421629	0.667537	0.431722	0.422428
72	0.512867	0.488831	0.172420	0.332815	0.194956
73	0.423061	0.805922	0.419022	0.726239	0.407660
74	0.357914	0.520280	0.529861	0.500646	0.450650
75	0.150370	0.673345	0.833046	0.752093	0.645827
76	0.387923	0.790584	0.905229	0.920720	0.828219
77	0.439651	0.514839	0.569826	0.522570	0.522584
78	0.531622	0.477490	0.617011	0.519070	0.602489
79	0.301076	0.520524	0.371124	0.424765	0.282395
80	0.203392	0.676109	0.564797	0.643389	0.428258
81	0.389418	0.504898	0.578293	0.511160	0.506822
82	0.417955	0.359848	0.606599	0.391517	0.530965
83	0.315315	0.369827	0.508592	0.345984	0.397887
84	0.609858	0.605624	0.432369	0.566893	0.483612
85	0.452922	0.609031	0.579857	0.616401	0.546679
86	0.473861	0.778100	0.532267	0.755766	0.529588
87	0.435150	0.699440	0.550027	0.685640	0.520666
88	0.491370	0.625338	0.460455	0.583810	0.457926
89	0.506716	0.419246	0.762226	0.525404	0.716457
90	0.360526	0.435696	0.453688	0.388668	0.375029
91	0.619885	0.641564	0.679638	0.709859	0.714190
92	0.498561	0.539183	0.598737	0.564795	0.577269
93	0.318475	0.559218	0.571317	0.550546	0.474149
94	0.277401	0.431960	0.819845	0.535833	0.667214
95	0.552871	0.442197	0.252268	0.328910	0.280172
96	0.388404	0.275013	0.481498	0.254144	0.396914
97	0.408313	0.546704	0.401122	0.474796	0.359806
98	0.516627	0.531839	0.535843	0.532505	0.527997
99	0.526772	0.506588	0.721226	0.591289	0.696973

100	0.397174	0.416407	0.473164	0.383457	0.407048
101	0.488371	0.462160	0.440538	0.422470	0.422820
102	0.247302	0.587819	0.869624	0.699569	0.713722
103	0.700640	0.497155	0.647641	0.570483	0.707412
104	0.452214	0.872125	0.644459	0.890112	0.630058
105	0.384030	0.421738	0.432025	0.368892	0.364662
106	0.303167	0.703746	0.555696	0.676794	0.467279
107	0.639977	0.498648	0.658379	0.569525	0.690149
108	0.223891	0.242257	0.585113	0.249815	0.413593
109	0.631643	0.898874	0.715641	0.967101	0.776824
110	0.567436	0.692521	0.633157	0.730961	0.653879
111	0.287873	0.587780	0.334909	0.470134	0.250425
112	0.355062	0.699463	0.863357	0.813536	0.767012
113	0.327867	0.544161	0.845918	0.657818	0.724102
114	0.352646	0.475376	0.387773	0.395920	0.316020
115	0.492754	0.768175	0.605953	0.780964	0.603396
116	0.329908	0.901334	0.581710	0.875674	0.521811
117	0.355017	0.634273	0.817293	0.732531	0.719194
118	0.521815	0.388657	0.531586	0.397633	0.512583
119	0.817893	0.362629	0.721315	0.490840	0.813019
120	0.688525	0.409474	0.442171	0.397293	0.508512
121	0.625846	0.810628	0.615001	0.840016	0.675066
122	0.452627	0.258504	0.767609	0.371475	0.681562
123	0.206553	0.318901	0.816339	0.420559	0.621464
124	0.606945	0.755048	0.638460	0.796225	0.682356
125	0.470948	0.469048	0.488499	0.447871	0.458890
126	0.422624	0.548255	0.434379	0.492470	0.396285
127	0.450647	0.629690	0.564853	0.628845	0.534162
128	0.272210	0.000000	0.518033	0.000000	0.351256
129	0.725048	0.617044	0.589980	0.659951	0.678030
130	0.043965	0.277900	0.702307	0.313468	0.442254
131	0.529708	0.607370	0.304722	0.503315	0.333087
132	0.469737	0.630264	0.418290	0.567430	0.410787
133	0.244558	0.490049	0.357365	0.383731	0.241824
134	0.508314	0.732811	0.678241	0.781423	0.671990
135	0.437354	0.653608	0.577671	0.655231	0.542089
136	0.162317	0.288427	0.699209	0.335696	0.493312
137	0.389076	0.551349	0.520578	0.529197	0.459217
138	0.277979	0.663973	0.517457	0.620009	0.417751
139	0.277552	0.497287	0.219742	0.334065	0.133351
140	0.419082	0.797799	0.405383	0.712225	0.392817
141	0.411901	0.296111	0.427096	0.252744	0.360472
142	0.178128	0.352707	0.450948	0.288820	0.283098
143	0.354784	0.432636	0.573638	0.437664	0.480160
144	0.316086	0.620376	0.398277	0.531575	0.323231
145	0.551949	0.750303	0.450722	0.703204	0.488330
146	0.219245	0.386537	0.496590	0.345152	0.345824
147	0.419895	0.560559	0.493775	0.529639	0.449735
148	0.649265	0.626379	0.479705	0.611568	0.545831
149	0.473029	0.564865	0.517191	0.550087	0.494950
150	0.414555	0.596657	0.539248	0.582613	0.491794
151	0.419328	0.683520	0.445104	0.623005	0.417597
152	0.603634	0.358884	0.550741	0.387749	0.563457

153	0.296986	0.542092	0.618991	0.552940	0.505816
154	0.457778	0.464149	0.923949	0.632423	0.844570
155	0.792653	0.522377	0.524061	0.550610	0.639670
156	0.396671	0.348729	0.627317	0.387738	0.539039
157	0.342456	0.349046	0.509358	0.330084	0.408674
158	0.581614	0.430139	0.495137	0.427337	0.510485
159	0.478466	0.332001	0.539533	0.343204	0.494892
160	0.440799	0.359631	0.629352	0.403933	0.561625
161	0.370989	0.237582	0.738686	0.329796	0.617055
162	0.521726	0.490970	0.619611	0.531637	0.601722
163	0.747000	0.435647	0.652703	0.520694	0.726693
164	0.420187	0.426320	0.424143	0.373920	0.374149
165	0.641001	0.443271	0.494777	0.446338	0.537942
166	0.474451	0.266666	0.571596	0.295808	0.515625
167	0.692868	0.507605	0.395424	0.468902	0.477887
168	0.637050	0.700978	0.412506	0.650338	0.487120
169	0.350156	0.582468	0.127366	0.381548	0.090858
170	0.454401	0.723151	0.646275	0.752144	0.618211
171	0.425838	0.554152	0.582980	0.563408	0.532076
172	0.533378	0.530156	0.552501	0.540175	0.550306
173	0.598648	0.285659	0.792103	0.424516	0.771423
174	0.204442	0.465205	0.507784	0.421742	0.356930
175	0.523881	0.530808	0.458539	0.498540	0.461537
176	0.307116	0.548271	0.616743	0.558899	0.508914
177	0.404244	0.843278	0.000000	0.575448	0.025644
178	0.495422	0.381148	0.424676	0.340748	0.403824
179	0.563054	0.734259	0.633466	0.769535	0.656252
180	0.267013	0.431541	0.357266	0.331701	0.246079
181	0.389212	0.625660	0.305723	0.504486	0.273056
182	0.570205	0.612909	0.426188	0.566374	0.461055
183	0.479040	0.671630	0.515753	0.649785	0.506700
184	0.475454	0.553102	0.742159	0.637892	0.697428
185	0.697562	0.417024	0.718807	0.526509	0.762332
186	0.416753	0.401297	0.605423	0.429541	0.533393
187	0.233620	0.525002	0.616019	0.528323	0.473199
188	0.179540	0.292983	0.947267	0.450557	0.724765
189	0.130285	0.523048	0.617746	0.515242	0.428443
190	0.634407	0.368960	0.673358	0.454416	0.688563
191	0.362304	0.288901	0.486064	0.266068	0.390724
192	0.338988	0.672067	0.458677	0.608919	0.392847
193	0.391554	0.357394	0.700703	0.427359	0.603665
194	0.602749	0.581981	0.529459	0.586513	0.565555
195	0.372620	0.216842	0.344810	0.138179	0.261165
196	0.562963	0.314890	0.588460	0.358483	0.574994
197	0.509981	0.425256	0.835279	0.563377	0.784267
198	0.513495	0.674878	0.749570	0.759196	0.732898
199	0.640521	0.314823	0.593194	0.369510	0.613866
200	0.215038	0.454076	0.466323	0.394436	0.323252
201	0.554842	0.366938	0.667142	0.440557	0.647257
202	0.416780	0.616383	0.802105	0.716367	0.731351
203	0.737312	0.383384	0.637589	0.464180	0.703689
204	0.311177	0.603129	0.407897	0.519123	0.328027
205	0.363103	0.566405	0.737771	0.635321	0.644623

206	0.338065	0.648994	0.677870	0.683252	0.587534
207	0.408845	0.755363	0.623744	0.767042	0.580720
208	0.537798	0.421588	0.363167	0.356481	0.371287
209	0.830993	0.470972	0.794342	0.625437	0.895126
210	0.401496	0.754859	0.219832	0.588870	0.213750
211	0.335633	0.260570	0.754949	0.354256	0.618146
212	0.257536	0.619619	0.519870	0.577300	0.406495
213	0.400137	0.615488	0.781308	0.704491	0.705112
214	0.253317	0.530397	0.429490	0.453980	0.314584
215	0.120935	0.448279	0.748264	0.501529	0.534519
216	0.407983	0.727062	0.216057	0.562033	0.210549
217	0.214113	0.539644	0.360773	0.427963	0.236117
218	0.585288	0.543556	0.447377	0.512689	0.480134
219	0.192786	0.289994	0.400279	0.209818	0.237937
220	0.574003	0.567670	0.696240	0.642839	0.701487
221	0.686558	0.337127	0.386861	0.305336	0.450818
222	0.676359	0.678523	0.569892	0.702864	0.644179
223	0.693065	0.757699	0.252104	0.639551	0.373217
224	0.266133	0.305283	0.530794	0.289757	0.389660
225	0.034142	0.760411	0.522361	0.683797	0.322693
226	0.418073	0.567672	0.443493	0.514050	0.404344
227	0.392294	0.198365	0.545660	0.211164	0.448978
228	0.346619	0.176795	0.419444	0.130463	0.312866
229	0.423714	0.517306	0.364828	0.433264	0.331152
230	0.029483	0.750815	0.482340	0.656778	0.283651
231	0.442583	0.509540	0.640367	0.548851	0.586886
232	0.538114	0.610023	0.488551	0.587254	0.502597
233	0.596575	0.497648	0.996416	0.711550	0.975014
234	0.212838	0.393174	0.566059	0.381017	0.406152
235	0.226876	0.202780	0.567045	0.205413	0.394827
236	0.629047	0.273107	0.108535	0.117048	0.168360
237	0.387964	0.557558	0.498384	0.525144	0.439342
238	0.279077	0.543834	0.386772	0.450810	0.288926
239	0.597894	0.320733	0.345188	0.261483	0.372135
240	0.298712	0.501323	0.615324	0.513492	0.499329
241	0.445002	0.463937	0.563975	0.473131	0.514765
242	0.582824	0.439963	0.397448	0.393872	0.424029
243	0.422312	0.354222	0.525431	0.351236	0.459289
244	0.377015	0.388917	0.811946	0.503792	0.700387
245	0.656546	0.394418	0.589589	0.444070	0.625497
246	0.242894	0.642993	0.335748	0.516794	0.236463
247	0.184291	0.295779	0.841489	0.407406	0.631926
248	0.415263	0.760550	0.597788	0.761264	0.560720
249	0.292638	0.579227	0.506816	0.537973	0.406489
250	0.571717	0.553357	0.615207	0.593738	0.626124
251	0.518840	0.611598	0.673474	0.667449	0.660633
252	0.383127	0.354366	0.404362	0.293806	0.332816
253	0.440957	0.400057	0.429665	0.354245	0.385841
254	0.355576	0.439523	0.627983	0.467976	0.530109
255	0.263754	0.310834	0.578498	0.315547	0.432084
256	0.402683	0.730517	0.474563	0.677823	0.441251
257	0.432250	0.255330	0.199302	0.117319	0.160515
258	0.536590	0.424213	0.377515	0.365072	0.383919

259	0.347051	0.762773	0.794315	0.841454	0.707422
260	0.480894	0.197390	0.681119	0.279864	0.610381
261	0.277397	0.264356	0.696603	0.325472	0.539993
262	0.537282	0.693669	0.315484	0.589438	0.354532
263	0.461439	0.382752	0.425958	0.338855	0.389966
264	0.505875	0.227730	0.315222	0.150878	0.295059
265	0.414604	0.585500	0.681191	0.634355	0.618524
266	0.455356	0.612992	0.428058	0.553917	0.411487
267	0.588356	0.521664	0.505932	0.518254	0.532095
268	0.589040	0.478404	0.491980	0.471856	0.515641
269	0.484636	0.426627	0.511242	0.419835	0.481359
270	0.429851	0.759915	0.766622	0.836290	0.719169
271	0.568835	0.382995	0.740482	0.489278	0.721088
272	0.538332	0.462964	0.647858	0.519801	0.631846
273	0.198346	0.676208	0.916443	0.796857	0.742601
274	0.645920	0.511585	0.689905	0.596091	0.722440
275	0.212466	0.523531	0.487673	0.468297	0.348066
276	0.151066	0.302214	0.385369	0.209842	0.207078
277	0.356092	0.528055	0.434665	0.466009	0.364887
278	0.565091	0.517555	0.534075	0.524036	0.546649
279	0.435754	0.377394	0.680536	0.442332	0.607178
280	0.573113	0.407139	0.587650	0.445391	0.587748
281	0.668672	0.315417	0.482780	0.324995	0.527083
282	0.656449	0.549905	0.522569	0.559809	0.580206
283	0.464976	0.466918	0.696741	0.536365	0.643499
284	0.586355	0.601419	0.640406	0.651322	0.660008
285	0.532767	0.714037	0.699773	0.776175	0.700468
286	0.751946	0.605702	0.272902	0.513666	0.403471
287	0.394018	0.414321	0.580217	0.428015	0.501816
288	0.761195	0.403965	0.713312	0.519316	0.784520
289	0.449036	0.539811	0.458016	0.498009	0.428535
290	0.516516	0.348309	0.731275	0.446796	0.686082
291	0.323445	0.533116	0.267620	0.393798	0.200417
292	0.380455	0.526620	0.678996	0.574479	0.595592
293	0.342184	0.414251	0.431935	0.356999	0.345176
294	0.475983	0.500647	0.379923	0.430405	0.366454
295	0.509084	0.291560	0.877594	0.457040	0.808987
296	0.632797	0.541310	0.327370	0.463573	0.393078
297	0.494032	0.357548	0.530772	0.365017	0.496431
298	0.327876	0.576087	0.292081	0.445121	0.228586
299	0.506194	0.698401	0.524237	0.681638	0.529055
300	0.122872	0.613148	0.920422	0.730979	0.706377
301	0.431132	0.520578	0.628118	0.552457	0.571819
302	0.378883	0.606690	0.641383	0.632546	0.568798
303	0.126893	0.525289	0.514163	0.471586	0.333891
304	0.674135	0.459717	0.292401	0.376930	0.372128
305	0.582329	0.684212	0.631312	0.724131	0.658059
306	0.493105	0.600856	0.771453	0.697332	0.736314
307	0.570651	0.565261	0.472436	0.542212	0.498267
308	0.224449	0.468262	0.452170	0.402571	0.316087
309	0.206681	0.450806	0.469067	0.391614	0.321675
310	0.424169	0.907299	0.599864	0.900148	0.580805
311	0.344529	0.567843	0.532270	0.544527	0.451460

312	0.567839	0.624887	0.501060	0.610058	0.528569
313	0.559318	0.403212	0.594100	0.442946	0.587016
314	0.204384	0.574840	0.429150	0.489613	0.296750
315	0.427015	0.643727	0.579431	0.645579	0.538101
316	0.345856	0.912668	0.745923	0.960003	0.677869
317	0.264248	0.523017	0.727763	0.578958	0.587280
318	0.383191	0.393551	0.517473	0.379903	0.438480
319	0.563284	0.294054	0.369001	0.242990	0.375537
320	0.746384	0.664627	0.793906	0.796119	0.875764
321	0.120770	0.325627	0.207729	0.150390	0.035900
322	0.388190	0.189795	0.566957	0.212014	0.465488
323	0.448014	0.372999	0.599268	0.404075	0.539059
324	0.247855	0.852261	0.376355	0.730429	0.295545
325	0.403130	0.667134	0.405243	0.588379	0.372890
326	0.627460	0.641408	0.310456	0.548954	0.385184
327	0.390136	0.475798	0.807033	0.584239	0.710252
328	0.167673	0.210120	0.573089	0.208026	0.374557
329	0.450918	0.315741	0.568000	0.337291	0.506647
330	0.339840	0.614244	0.857190	0.729543	0.746395
331	0.447260	0.748716	0.489925	0.706715	0.476743
332	0.551857	0.419595	0.438873	0.389403	0.445526
333	0.473436	0.233891	0.591653	0.273887	0.530049
334	0.520369	0.369762	0.576333	0.399425	0.550390
335	0.324852	0.425775	0.576506	0.429037	0.468716
336	0.336675	0.435772	0.410085	0.366875	0.325135
337	0.439945	0.423245	0.348015	0.340016	0.314131
338	0.347736	0.746685	0.594118	0.738868	0.525930
339	0.498797	0.575472	0.414665	0.518093	0.415177
340	0.511520	0.510424	0.562388	0.523550	0.547537
341	0.321676	0.316038	0.848404	0.445312	0.701438
342	0.500432	0.494693	0.384176	0.429553	0.380618
343	0.467733	0.451503	0.416998	0.399819	0.391381
344	0.319473	0.713303	0.440474	0.637160	0.371751
345	0.542336	0.633163	0.239657	0.500364	0.282649
346	0.668824	0.331996	0.645546	0.411747	0.675297
347	0.342724	0.712618	0.250294	0.555957	0.210843
348	0.602210	0.687768	0.720770	0.768929	0.747816
349	0.563532	0.411027	0.924268	0.595288	0.886904
350	0.421440	0.422074	0.624381	0.457774	0.554570
351	0.352971	0.615249	0.601808	0.620193	0.522434
352	0.410854	0.762438	0.574445	0.752293	0.537919
353	0.259953	0.470302	0.734632	0.532275	0.586431
354	0.286753	0.635413	0.609952	0.634875	0.502168
355	0.228063	0.590680	0.364838	0.478990	0.250956
356	0.505136	0.393270	0.496953	0.384834	0.474406
357	0.375159	0.322931	0.687004	0.387296	0.580670
358	0.312518	0.560731	0.319562	0.441038	0.244983
359	0.732703	0.830259	0.810518	0.956362	0.900688
360	0.652364	0.376571	0.543032	0.406545	0.579983
361	0.310175	0.518240	0.305392	0.394911	0.227056
362	0.629309	0.614961	0.706328	0.697816	0.739844
363	0.430337	0.823714	0.310151	0.696021	0.314619
364	0.384803	0.432150	0.718375	0.504071	0.623817

365	0.583019	0.621397	0.352693	0.543606	0.401431
366	0.532361	0.700454	0.626905	0.731548	0.633365
367	0.159645	0.315912	0.387597	0.224598	0.214243
368	0.428968	0.413985	0.295232	0.306989	0.260812
369	0.592144	1.000000	0.585743	1.000000	0.652061
370	0.299767	0.384954	0.928797	0.542273	0.770724
371	0.052100	0.461451	0.708190	0.488270	0.468995
372	0.643787	0.289386	0.363083	0.245403	0.405687
373	0.244360	0.502391	0.225483	0.337483	0.124201
374	0.298256	0.440165	0.574232	0.438377	0.456195
375	0.259982	0.483471	0.441398	0.416179	0.323725
376	0.386419	0.550629	0.717050	0.614238	0.634843
377	0.215165	0.416090	0.390958	0.326006	0.251771
378	0.322230	0.841203	0.564988	0.811348	0.497493
379	0.103809	0.338268	0.621671	0.341453	0.402227
380	0.391534	0.296574	0.699525	0.370087	0.596693
381	0.401844	0.526547	0.418606	0.462890	0.370703
382	0.491421	0.323629	0.600977	0.363800	0.555179
383	0.653161	0.555345	0.539315	0.571834	0.594343
384	0.649420	0.329565	0.528245	0.355865	0.560795
385	0.435306	0.267494	0.436960	0.233081	0.377022
386	0.606100	0.462783	0.329429	0.388093	0.375395
387	0.790043	0.334767	0.580030	0.399743	0.670686
388	0.700986	0.623003	0.486731	0.617508	0.574913
389	0.285808	0.508526	0.499953	0.468199	0.390400
390	0.669673	0.385686	1.000000	0.617140	1.000000
391	0.519859	0.654488	0.600760	0.675754	0.599786
392	0.301631	0.563909	0.476189	0.511315	0.381443
393	0.291577	0.711538	0.441476	0.632708	0.360030
394	0.472794	0.417696	0.285289	0.311193	0.271782
395	0.676839	0.341720	0.410829	0.318986	0.468503
396	0.440624	0.557338	0.667496	0.605105	0.615075
397	0.403533	0.544189	0.524498	0.525912	0.468503
398	0.490185	0.786186	0.920117	0.935024	0.886838
399	0.378775	0.620748	0.532739	0.598084	0.472302
400	0.431613	0.492306	0.495067	0.467876	0.449504
401	0.232580	0.524447	0.679228	0.555360	0.529588
402	0.451126	0.811287	0.602689	0.814925	0.586063
403	0.306758	0.428947	0.620950	0.449352	0.500961
404	0.562043	0.313344	0.602859	0.363238	0.587397
405	0.320311	0.666110	0.546631	0.639698	0.463117
406	0.292337	0.393587	0.484091	0.354758	0.367879
407	0.376424	0.650680	0.907905	0.790002	0.811918
408	0.419520	0.501518	0.493047	0.474182	0.443182
409	0.321825	0.271472	0.333488	0.178294	0.233602
410	0.506158	0.248707	0.666119	0.324120	0.613132
411	0.306158	0.121773	0.792157	0.237601	0.625019
412	0.547512	0.688270	0.636766	0.726258	0.647823
413	0.063371	0.441281	0.183842	0.241182	0.000000
414	0.519031	0.641461	0.585021	0.656611	0.583983
415	0.269163	0.380064	0.462257	0.329883	0.336566
416	0.393142	0.668590	0.346588	0.562895	0.315767
417	0.476073	0.536643	0.390273	0.468537	0.379306

418	0.674665	0.321452	0.753095	0.449677	0.773707
419	0.526830	0.678058	0.439524	0.627964	0.460025
420	0.263753	0.696281	0.462985	0.624653	0.365495
421	0.452434	0.650869	0.388773	0.571722	0.378490
422	0.212905	0.393103	0.710798	0.444331	0.536482
423	0.399267	0.372850	0.447953	0.332016	0.381058
424	1.000000	0.594340	0.622729	0.685071	0.828030
425	0.458603	0.449615	0.844275	0.584073	0.771799
426	0.481810	0.301563	0.561354	0.324743	0.513076
427	0.369439	0.627145	0.449016	0.566311	0.393381
428	0.146431	0.879082	0.543855	0.817002	0.403679
429	0.381061	0.654445	0.397710	0.570674	0.355027
430	0.240490	0.636550	0.576362	0.615851	0.451389
431	0.505471	0.524873	0.630863	0.566310	0.607887
432	0.491884	0.602998	0.612314	0.629512	0.592705
433	0.522418	0.652663	0.497379	0.629085	0.507677
434	0.501931	0.364460	0.277318	0.261414	0.272444
435	0.754941	0.665356	0.502058	0.670013	0.616905
436	0.432004	0.543083	0.762369	0.632339	0.695258
437	0.401007	0.385907	0.466318	0.352444	0.399636
438	0.483768	0.879821	0.554783	0.861698	0.564153
439	0.215771	0.407960	0.400708	0.322759	0.260031
440	0.578651	0.243457	0.603457	0.300215	0.588565
441	0.323598	0.567663	0.574647	0.560481	0.480253
442	0.534403	0.521066	0.661780	0.579658	0.648266
443	0.621663	0.428160	0.720569	0.528848	0.731123
444	0.323942	0.585625	0.391317	0.497014	0.317099
445	0.585120	0.598018	0.621642	0.639790	0.642234
446	0.443282	0.421767	0.656791	0.474218	0.593467
447	0.674267	0.605974	0.422890	0.570558	0.503860
448	0.509463	0.924496	0.572869	0.914293	0.596240
449	0.439549	0.492255	0.588101	0.509485	0.536800
450	0.316567	0.264399	0.642456	0.306360	0.508731
451	0.469392	0.595430	0.287766	0.477737	0.289743
452	0.615737	0.270113	0.323525	0.206837	0.355683
453	0.265270	0.377224	0.643160	0.405987	0.497420
454	0.358493	0.409586	0.682418	0.464213	0.577513
455	0.628639	0.548275	0.629797	0.602002	0.664172
456	0.444585	0.443221	0.267736	0.324048	0.245866
457	0.531266	0.594243	0.550871	0.599020	0.554116
458	0.192793	0.434723	0.710838	0.480847	0.531581
459	0.354651	0.257676	0.654225	0.309667	0.535673
460	0.499389	0.516725	0.387739	0.451552	0.385498
461	0.327839	0.520057	0.568432	0.513829	0.471931
462	0.280085	0.299699	0.562629	0.300107	0.424006
463	0.387644	0.461791	0.803071	0.569144	0.704213
464	0.422144	0.582540	0.237997	0.438424	0.222596
465	0.459018	0.318094	0.677465	0.388356	0.609041
466	0.416577	0.455315	0.556351	0.458442	0.494377
467	0.411776	0.185727	0.522933	0.191685	0.435985
468	0.517695	0.501070	0.483702	0.481087	0.478545
469	0.474866	0.588633	0.441675	0.539417	0.430090
470	0.443325	0.444904	0.551771	0.449832	0.501182

471	0.185352	0.361657	0.505543	0.321915	0.336343
472	0.429364	0.450611	0.596229	0.473000	0.535530
473	0.401208	0.648549	0.798327	0.742919	0.724121
474	0.461456	0.624926	0.342410	0.528263	0.338258
475	0.325232	0.479727	0.797563	0.576213	0.673138
476	0.521397	0.630922	0.280571	0.513752	0.309921
477	0.534737	0.590531	0.822431	0.714857	0.799789
478	0.260448	0.838188	0.169815	0.628331	0.113852
479	0.279450	0.298978	0.360306	0.210776	0.241502
480	0.404414	0.619360	0.764433	0.701213	0.692204
481	0.215046	0.283725	0.571279	0.281426	0.401215
482	0.476218	0.689630	0.513901	0.665443	0.505520
483	0.219129	0.770533	0.542434	0.723539	0.424312
484	0.452088	0.698519	0.649956	0.730502	0.618102
485	0.298156	0.479774	0.305327	0.357590	0.217899
486	0.486695	0.461996	0.378154	0.394808	0.365891
487	0.480459	0.480720	0.634286	0.523699	0.595520
488	0.631019	0.746471	0.241028	0.617010	0.334463
489	0.498102	0.429855	0.322720	0.341871	0.317957
490	0.439898	0.358174	0.620379	0.398540	0.553003
491	0.410318	0.525794	0.682025	0.578507	0.611568
492	0.299842	0.660198	0.717180	0.706474	0.606953
493	0.095117	0.644079	0.488892	0.567676	0.308486
494	0.598300	0.307997	0.324433	0.240559	0.352394
495	0.328186	0.352288	0.422679	0.293498	0.324582
496	0.471470	0.479074	0.397704	0.417534	0.378354
497	0.481953	0.632044	0.207359	0.478158	0.226513
498	0.555268	0.310272	0.425893	0.282101	0.424753
499	0.501667	0.429026	0.588939	0.458072	0.559143

Table A3

Training Data with 1000 Samples

	x_1	x_2	x_3	y_1	y_2
0	0.067749	0.272967	0.566500	0.326474	0.318411
1	0.358815	0.342850	0.412627	0.345126	0.327931
2	0.305155	0.442857	0.374793	0.311727	0.352814
3	0.412182	0.038874	0.440624	0.313734	0.164327
4	0.530997	0.630074	0.472967	0.562104	0.598256
5	0.533711	0.381757	0.430354	0.456598	0.408709
6	0.466233	0.416050	0.800770	0.773202	0.673199
7	0.494702	0.296210	0.454093	0.436184	0.360397
8	0.602502	0.375926	0.450402	0.506678	0.435971
9	0.477276	0.594198	0.326742	0.392161	0.459545
10	0.492723	0.607820	0.238242	0.322140	0.410279
11	0.536941	0.491967	0.579542	0.625110	0.585030
12	0.509525	0.630274	0.695093	0.755594	0.748403
13	0.475785	0.444067	0.282865	0.310237	0.331505
14	0.592022	0.422503	0.401287	0.469243	0.429125
15	0.534299	0.641077	0.403074	0.502569	0.557307
16	0.359075	0.699967	0.206030	0.253067	0.414237
17	0.512860	0.485911	0.312798	0.367047	0.388636
18	0.417429	0.431349	0.657296	0.622118	0.570688
19	0.508060	0.581401	0.463079	0.528663	0.554226
20	0.513764	0.219234	0.332960	0.313262	0.230678
21	0.466057	0.639314	0.316464	0.389606	0.478730
22	0.295942	0.652186	0.419086	0.404998	0.516742
23	0.318158	0.491216	0.169536	0.142862	0.243740
24	0.053628	0.737486	0.391082	0.285357	0.492347
25	0.591452	0.542489	0.577299	0.663195	0.629579
26	0.333014	0.559452	0.350253	0.334480	0.417866
27	0.507249	0.292909	0.470966	0.456835	0.373166
28	0.375823	0.632123	0.409068	0.428972	0.516527
29	0.309093	0.543433	0.293443	0.266413	0.361877
30	0.355911	0.448719	0.514401	0.465980	0.466776
31	0.474838	0.375957	0.346118	0.349246	0.331501
32	0.612461	0.653576	0.392657	0.534225	0.577425
33	0.443561	0.442612	0.481905	0.476891	0.461779
34	0.300567	0.505961	0.534637	0.473388	0.504221
35	0.409093	0.406965	0.469136	0.438782	0.421299
36	0.543274	0.469013	0.498478	0.547529	0.515081
37	0.312831	0.363550	0.483032	0.393131	0.379165
38	0.382268	0.248942	0.394785	0.314489	0.260586
39	0.484110	0.460721	0.395993	0.422611	0.423430
40	0.341648	0.418217	0.466177	0.406512	0.409821
41	0.733941	0.575571	0.195477	0.390781	0.419187
42	0.317854	0.527328	0.381634	0.347182	0.415309
43	0.419087	0.690844	0.517890	0.565767	0.641257
44	0.675790	0.522807	0.289379	0.434429	0.436384
45	0.441651	0.285296	0.374257	0.334289	0.284401
46	0.417874	0.424770	0.250440	0.247210	0.282044

47	0.795266	0.328824	0.468987	0.604126	0.466203
48	0.151922	0.513763	0.499065	0.370976	0.447630
49	0.608633	0.280588	0.638110	0.655884	0.507154
50	0.299767	0.468543	0.512986	0.442931	0.464710
51	0.309942	0.386600	0.392478	0.314929	0.330021
52	0.518444	0.244291	0.537640	0.510172	0.391149
53	0.431899	0.678748	0.423953	0.482469	0.570923
54	0.658142	0.323715	0.427690	0.498512	0.400117
55	0.587461	0.361148	0.264736	0.325007	0.292874
56	0.465719	0.686118	0.524751	0.593329	0.654532
57	0.547177	0.347987	0.434174	0.457408	0.392894
58	0.283549	0.370527	0.519332	0.414179	0.401816
59	0.485891	0.227323	0.601882	0.548746	0.417060
60	0.327820	0.813410	0.678611	0.702521	0.810257
61	0.639376	0.411168	0.333005	0.426402	0.385764
62	0.295374	0.259663	0.561951	0.428773	0.362920
63	0.643503	0.362410	0.543414	0.608171	0.502418
64	0.319005	0.553898	0.386177	0.359153	0.435934
65	0.665091	0.387416	0.476610	0.564133	0.477197
66	0.521297	0.286384	0.706286	0.677779	0.536977
67	0.511036	0.529612	0.483906	0.535089	0.536068
68	0.562836	0.512760	0.421978	0.498725	0.494686
69	0.822423	0.356191	0.581658	0.728111	0.569386
70	0.396916	0.464523	0.475265	0.454212	0.459758
71	0.390664	0.663814	0.334493	0.376364	0.488522
72	0.273663	0.600443	0.471973	0.428640	0.514787
73	0.554424	0.397340	0.574941	0.603538	0.525004
74	0.304856	0.238935	0.374839	0.256015	0.221032
75	0.457768	0.294557	0.555103	0.510554	0.420830
76	0.548794	0.681354	0.258018	0.387460	0.485478
77	0.223210	0.664107	0.456839	0.407711	0.532857
78	0.160172	0.492172	0.465722	0.338483	0.412407
79	0.259386	0.625250	0.466062	0.423075	0.523149
80	0.338551	0.310427	0.683009	0.574583	0.491044
81	0.529990	0.328028	0.317784	0.336854	0.294362
82	0.642000	0.399518	0.573931	0.645566	0.547359
83	0.482850	1.000000	0.401931	0.574513	0.775661
84	0.739353	0.815520	0.652795	0.878465	0.895329
85	0.563096	0.736983	0.365867	0.508510	0.600368
86	0.375878	0.504741	0.507771	0.484830	0.503268
87	0.705086	0.409620	0.579658	0.684090	0.573490
88	0.583733	0.399911	0.475064	0.526770	0.464070
89	0.629342	0.630174	0.548421	0.678936	0.675402
90	0.593392	0.610365	0.287691	0.416901	0.471394
91	0.165545	0.483766	0.397906	0.276562	0.360883
92	0.644456	0.423348	0.583382	0.661925	0.569971
93	0.493384	0.472186	0.512255	0.536904	0.514428
94	0.454541	0.547444	0.538739	0.562940	0.571961
95	0.537246	0.466247	0.487897	0.534150	0.504405
96	0.075682	0.453259	0.571191	0.383782	0.440127
97	0.243743	0.459949	0.367022	0.279552	0.343236
98	0.649464	0.309234	0.479485	0.537893	0.424836
99	0.464314	0.618511	0.456884	0.511939	0.563051

100	0.562798	0.291268	0.495672	0.505927	0.403118
101	0.400108	0.424478	0.579313	0.540309	0.507435
102	0.376572	0.658895	0.649767	0.657499	0.702321
103	0.498836	0.609636	0.267686	0.352609	0.433553
104	0.024878	0.623676	0.374241	0.224962	0.399938
105	0.445285	0.511336	0.507336	0.519802	0.524387
106	0.694356	0.331142	0.314940	0.414595	0.335027
107	0.348334	0.674150	0.606405	0.608212	0.674873
108	0.537288	0.506791	0.492071	0.549056	0.533526
109	0.641932	0.531701	0.267497	0.400398	0.418456
110	0.491124	0.734397	0.497012	0.593330	0.672606
111	0.117629	0.632645	0.396306	0.292518	0.444096
112	0.697263	0.499466	0.009921	0.182022	0.231198
113	0.460738	0.428667	0.558561	0.551736	0.510622
114	0.427285	0.285179	0.327376	0.284292	0.247991
115	0.422711	0.714155	0.178727	0.262663	0.420046
116	0.390704	0.317198	0.554296	0.483549	0.418309
117	0.469599	0.570512	0.186326	0.253143	0.344155
118	0.181955	0.499768	0.354093	0.248661	0.344641
119	0.251672	0.515137	0.548094	0.464588	0.507469
120	0.460243	0.446351	0.466261	0.471625	0.457381
121	0.364231	0.574416	0.423319	0.420705	0.486346
122	0.207328	0.431313	0.406719	0.290555	0.343491
123	0.535726	0.521385	0.372100	0.442197	0.458676
124	0.652435	0.330926	0.809886	0.848418	0.670623
125	0.557447	0.632695	0.238954	0.360884	0.442851
126	0.663182	0.559909	0.334264	0.479634	0.488622
127	0.563418	0.400891	0.455368	0.499138	0.445907
128	0.416725	0.517200	0.340255	0.354274	0.404278
129	0.492876	0.286269	0.501011	0.475641	0.386332
130	0.640807	0.882917	0.577133	0.779752	0.861594
131	0.624889	0.643160	0.107014	0.275291	0.374026
132	0.480975	0.443488	0.338973	0.364074	0.371649
133	0.715386	0.631503	0.488329	0.665777	0.655516
134	0.487930	0.578183	0.358652	0.422227	0.474147
135	0.384782	0.407950	0.244694	0.221345	0.258978
136	0.441270	0.632299	0.468509	0.515220	0.574390
137	0.556825	0.427335	0.458748	0.506261	0.463724
138	0.595605	0.533907	0.363199	0.466407	0.475348
139	0.505204	0.491365	0.633879	0.659453	0.614789
140	0.663167	0.630993	0.260991	0.431776	0.483303
141	0.401611	0.583487	0.364794	0.387557	0.460523
142	0.496480	0.932636	0.582615	0.728529	0.861859
143	0.492001	0.659707	0.417727	0.500635	0.569130
144	0.308593	0.519961	0.663732	0.599545	0.605522
145	0.413789	0.248366	0.413177	0.346455	0.280870
146	0.402878	0.453593	0.646857	0.611566	0.574160
147	0.344597	0.565860	0.239354	0.240070	0.347322
148	0.282070	0.305316	0.467765	0.348363	0.323263
149	0.311054	0.499621	0.584062	0.522084	0.537280
150	0.316222	0.579077	0.552406	0.517203	0.567752
151	0.723733	0.522944	0.341949	0.505893	0.485088
152	0.531169	0.738956	0.410484	0.534546	0.624947

153	0.449571	0.477505	0.413003	0.426090	0.437627
154	0.538545	0.497595	0.581587	0.629296	0.590492
155	0.409754	0.444820	0.665361	0.629479	0.583132
156	0.197464	0.706457	0.173572	0.146883	0.355770
157	0.738632	0.598056	0.489394	0.668877	0.640401
158	0.531839	0.192312	0.389693	0.366720	0.257426
159	0.453648	0.607662	0.402647	0.454053	0.515478
160	0.489249	0.677387	0.416251	0.502770	0.578839
161	0.412606	0.530297	0.536457	0.535888	0.548920
162	0.490399	0.531652	0.503056	0.543236	0.545674
163	0.690344	0.179839	1.000000	1.000000	0.715332
164	0.233664	0.530064	0.294135	0.226917	0.335072
165	0.341131	0.387973	0.404467	0.341389	0.347003
166	0.527664	0.987071	0.718847	0.883464	1.000000
167	0.084499	0.413173	0.466860	0.281381	0.343454
168	0.446375	0.250797	0.165366	0.135490	0.117211
169	0.313199	0.631975	0.450815	0.436948	0.530139
170	0.339092	0.826270	0.456991	0.508124	0.666380
171	0.387047	0.226191	0.511680	0.417859	0.328812
172	0.572939	0.307782	0.487193	0.507554	0.410365
173	0.596539	0.794881	0.330911	0.508400	0.621598
174	0.708087	0.488329	0.522904	0.654929	0.585394
175	0.410058	0.326929	0.523097	0.466936	0.407565
176	0.491441	0.326107	0.436691	0.426792	0.366737
177	0.480930	0.495211	0.460725	0.489876	0.490190
178	0.544391	0.701361	0.564479	0.671993	0.711614
179	0.126105	0.332262	0.502194	0.311864	0.326180
180	0.797811	0.513103	0.424353	0.614655	0.554671
181	0.404956	0.648619	0.682686	0.698633	0.725721
182	0.398536	0.386565	0.419099	0.382198	0.370520
183	0.483587	0.289107	0.313750	0.300093	0.254922
184	0.432550	0.557594	0.360102	0.391156	0.448166
185	0.348182	0.289329	0.286710	0.209847	0.202676
186	0.506620	0.351201	0.429705	0.434566	0.381816
187	0.396933	0.410394	0.482928	0.446490	0.430152
188	0.572316	0.560837	0.395232	0.491879	0.509386
189	0.287039	0.401651	0.603033	0.501158	0.481316
190	0.534127	0.348977	0.368782	0.391362	0.344580
191	0.479975	0.399072	0.609926	0.600102	0.532177
192	0.399354	0.414251	0.350957	0.327618	0.340958
193	0.484677	0.534405	0.495669	0.534441	0.540872
194	0.573490	0.474350	0.234478	0.321356	0.341393
195	0.558769	0.676052	0.514233	0.625940	0.663682
196	0.167667	0.863476	0.440138	0.419888	0.636243
197	0.387426	0.267842	0.417787	0.343245	0.290157
198	0.266028	0.577383	0.699931	0.627831	0.657406
199	0.521284	0.765619	0.607469	0.717787	0.777474
200	0.351217	0.611433	0.233271	0.250118	0.374146
201	0.340762	0.575229	0.318493	0.313387	0.407765
202	0.586559	0.504317	0.714030	0.775882	0.699321
203	0.360929	0.409269	0.306282	0.266680	0.296999
204	0.509615	0.647004	0.382555	0.473417	0.540684
205	0.615899	0.454568	0.269139	0.368278	0.363337

206	0.439553	0.536891	0.152444	0.198352	0.291313
207	0.532301	0.363198	0.466038	0.483599	0.421324
208	0.436684	0.347934	0.425445	0.395939	0.359434
209	0.314177	0.511109	0.333286	0.296617	0.370113
210	0.565851	0.656896	0.658039	0.756097	0.753618
211	0.383045	0.637335	0.348181	0.378018	0.479104
212	0.442404	0.823509	0.518383	0.613675	0.733072
213	0.613494	0.352513	0.445629	0.501231	0.420226
214	0.275419	0.259604	0.493416	0.356218	0.310024
215	0.579562	0.376196	0.596943	0.630120	0.532945
216	0.218290	0.664203	0.737268	0.662677	0.727799
217	0.574154	0.724859	0.408659	0.549818	0.625193
218	0.592450	0.270141	0.522941	0.539529	0.415869
219	0.318464	0.511283	0.302390	0.270388	0.349681
220	0.352161	0.255944	0.608480	0.497921	0.407096
221	0.317188	0.378120	0.747120	0.641537	0.574325
222	0.234691	0.408703	0.318207	0.216406	0.273756
223	0.406899	0.541656	0.453459	0.460066	0.496808
224	0.467663	0.554376	0.512628	0.547219	0.561426
225	0.431477	0.084605	0.388497	0.287707	0.162191
226	0.511266	0.565256	0.337113	0.410225	0.456504
227	0.507433	0.543177	0.209334	0.285102	0.351941
228	0.300100	0.353960	0.466120	0.368840	0.357996
229	0.315715	0.432351	0.464363	0.396158	0.411271
230	0.366323	0.572127	0.107998	0.131757	0.264889
231	0.393385	0.407750	0.456950	0.420216	0.409400
232	0.592720	0.672065	0.509883	0.637283	0.666460
233	0.695280	0.547358	0.573389	0.711156	0.655664
234	0.617054	0.791897	0.279060	0.469931	0.588485
235	0.649932	0.435401	0.401000	0.500508	0.451577
236	0.492851	0.967697	0.453731	0.618073	0.793487
237	0.731889	0.606090	0.476922	0.656363	0.635203
238	0.333570	0.259706	0.518770	0.407637	0.342198
239	0.376435	0.427961	0.368232	0.336122	0.356228
240	0.552501	0.691172	0.190765	0.330221	0.445709
241	0.432755	0.548398	0.386353	0.412835	0.460632
242	0.661790	0.545486	0.436517	0.568854	0.550463
243	0.411174	0.632655	0.291335	0.338185	0.443285
244	0.687072	0.551740	0.431312	0.578012	0.557115
245	0.631645	0.402820	0.401822	0.483532	0.426581
246	0.371063	0.623013	0.658198	0.652786	0.683675
247	0.428326	0.657189	0.496131	0.541092	0.606584
248	0.400303	0.643125	0.663005	0.676824	0.707259
249	0.549297	0.414103	0.296044	0.349717	0.339541
250	0.529160	0.668473	0.371125	0.478237	0.551394
251	0.282069	0.683920	0.375646	0.367082	0.503436
252	0.294687	0.468214	0.353474	0.294017	0.351699
253	0.405308	0.467351	0.650016	0.619393	0.585859
254	0.224234	0.456946	0.411372	0.309993	0.367485
255	0.287345	0.373316	0.375066	0.284398	0.303675
256	0.490518	0.613824	0.601358	0.655904	0.667529
257	0.468640	0.494788	0.778228	0.775155	0.708899
258	0.489969	0.491073	0.354790	0.395914	0.415674

259	0.501955	0.563101	0.730655	0.766245	0.728002
260	0.373427	0.477293	0.427723	0.402709	0.428955
261	0.509788	0.421184	0.456483	0.479754	0.446535
262	0.423033	0.536998	0.326312	0.349930	0.408878
263	0.505373	0.754538	0.489660	0.598968	0.684000
264	0.286054	0.252543	0.544712	0.406504	0.343961
265	0.404407	0.614553	0.250234	0.292262	0.401176
266	0.440122	0.544314	0.471339	0.493267	0.519244
267	0.404237	0.463928	0.386727	0.376349	0.399272
268	0.538602	0.666021	0.356936	0.469116	0.542223
269	0.457132	0.414279	0.530202	0.520046	0.480605
270	0.276459	0.427768	0.652960	0.548976	0.530484
271	0.548119	0.416682	0.307486	0.360349	0.348916
272	0.694822	0.471964	0.484475	0.608788	0.544670
273	0.395365	0.408410	0.379124	0.349941	0.355895
274	0.324005	0.421332	0.501620	0.431350	0.432255
275	0.520547	0.719802	0.547307	0.649732	0.705623
276	0.568191	0.600956	0.122384	0.250461	0.343489
277	1.000000	0.583789	0.503038	0.803929	0.705354
278	0.639145	0.475707	0.376159	0.483488	0.457577
279	0.412146	0.570308	0.273496	0.305285	0.390773
280	0.824164	0.487276	0.360450	0.561721	0.499817
281	0.542365	0.785425	0.336908	0.485120	0.606286
282	0.482533	0.429222	0.363693	0.383620	0.380104
283	0.606612	0.488813	0.801044	0.861197	0.755110
284	0.394490	0.470015	0.244936	0.243188	0.301643
285	0.619204	0.330217	0.179569	0.253777	0.221186
286	0.528385	0.398538	0.372134	0.405176	0.377522
287	0.454865	0.392955	0.349074	0.346933	0.339611
288	0.448353	0.345868	0.595493	0.557054	0.479894
289	0.313338	0.700453	0.278552	0.297622	0.453953
290	0.461747	0.366377	0.484329	0.467123	0.418722
291	0.180551	0.310304	0.315526	0.160926	0.194926
292	0.538018	0.083619	0.573999	0.509186	0.317614
293	0.638276	0.268259	0.092981	0.166654	0.125327
294	0.209294	0.582806	0.467117	0.388239	0.484081
295	0.433534	0.783636	0.329710	0.425387	0.573187
296	0.263352	0.481349	0.291417	0.225499	0.309042
297	0.395261	0.500724	0.377343	0.373431	0.414262
298	0.660453	0.362663	0.559615	0.631305	0.518102
299	0.315921	0.498856	0.293258	0.257389	0.334638
300	0.337220	0.654285	0.442352	0.446885	0.544573
301	0.283280	0.619979	0.139965	0.133971	0.297623
302	0.483293	0.348578	0.496483	0.483843	0.421050
303	0.369068	0.604327	0.486354	0.489042	0.550943
304	0.315616	0.471257	0.249918	0.209947	0.286426
305	0.612201	0.320502	0.475452	0.519240	0.420080
306	0.566613	0.429730	0.308904	0.374154	0.362911
307	0.226418	0.442803	0.380496	0.278860	0.337297
308	0.438564	0.763984	0.494881	0.574020	0.677234
309	0.711627	0.211573	0.197245	0.282347	0.179753
310	0.606492	0.437480	0.318730	0.404572	0.384649
311	0.353167	0.311906	0.663495	0.564150	0.481968

312	0.700509	0.498594	0.321768	0.469502	0.449503
313	0.366661	0.076738	0.494868	0.351815	0.215464
314	0.574391	0.403730	0.743508	0.769615	0.651943
315	0.634432	0.763822	0.388653	0.571242	0.651280
316	0.484203	0.518970	0.586340	0.613201	0.594187
317	0.263159	0.493473	0.186244	0.132206	0.243282
318	0.790445	0.474054	0.055057	0.261579	0.269384
319	0.408933	0.416373	0.631471	0.590227	0.540853
320	0.490672	0.463512	0.606334	0.619554	0.573941
321	0.682371	0.595216	0.343672	0.507176	0.522755
322	0.371107	0.687045	0.694933	0.703976	0.750740
323	0.579776	0.378498	0.183052	0.251069	0.245063
324	0.492291	0.309101	0.426454	0.413172	0.348802
325	0.694723	0.304744	0.516469	0.592496	0.458988
326	0.533625	0.482254	0.500552	0.548376	0.522699
327	0.452965	0.484672	0.288401	0.315352	0.355965
328	0.337501	0.524408	0.278721	0.261457	0.346316
329	0.493808	0.193885	0.549603	0.495486	0.360859
330	0.783352	0.670500	0.397732	0.626155	0.634163
331	0.370683	0.477610	0.337799	0.318954	0.365600
332	0.443290	0.469401	0.555986	0.552042	0.530822
333	0.353177	0.428036	0.333460	0.292986	0.326210
334	0.598597	0.512403	0.409896	0.504839	0.494849
335	0.611043	0.467915	0.323663	0.419600	0.408886
336	0.235281	0.400717	0.325982	0.221649	0.274180
337	0.684411	0.666477	0.460902	0.635166	0.651272
338	0.304498	0.624132	0.492849	0.469170	0.552314
339	0.504525	0.592293	0.494732	0.558968	0.582523
340	0.297012	0.534180	0.314868	0.277706	0.367894
341	0.247298	0.388351	0.456556	0.343902	0.360469
342	0.359268	0.146881	0.217943	0.113263	0.065304
343	0.487080	0.504267	0.044365	0.113272	0.206413
344	0.566631	0.578255	0.419220	0.515891	0.536007
345	0.509551	0.619167	0.439276	0.517841	0.562348
346	0.158877	0.564944	0.481914	0.372559	0.470422
347	0.319056	0.367290	0.398553	0.319646	0.324048
348	0.144694	0.602263	0.740441	0.613099	0.671805
349	0.428052	0.640991	0.200564	0.265332	0.389370
350	0.509496	0.298212	0.231455	0.239594	0.209664
351	0.497457	0.419054	0.414130	0.434346	0.412494
352	0.600854	0.602388	0.499456	0.612649	0.616167
353	0.451351	0.642005	0.467105	0.521455	0.582171
354	0.717894	0.406799	0.274804	0.409784	0.361658
355	0.507975	0.374382	0.345096	0.363906	0.337962
356	0.415909	0.379056	0.378304	0.351121	0.341438
357	0.643329	0.568768	0.504309	0.628479	0.608344
358	0.170734	0.664636	0.557856	0.475165	0.590862
359	0.501017	0.355302	0.369430	0.377667	0.340932
360	0.518479	0.686002	0.424493	0.526821	0.597396
361	0.391665	0.502297	0.428245	0.418827	0.449982
362	0.541408	0.596587	0.249436	0.352897	0.422888
363	0.478287	0.458629	0.562061	0.571607	0.536765
364	0.660033	0.878554	0.263621	0.500184	0.644299

365	0.212868	0.284846	0.325640	0.178895	0.193542
366	0.363963	0.733430	0.480221	0.516152	0.628797
367	0.664996	0.540143	0.423163	0.556694	0.538466
368	0.403619	0.270131	0.390410	0.326580	0.276495
369	0.554841	0.629754	0.408655	0.514537	0.558974
370	0.487525	0.261932	0.462261	0.430860	0.342190
371	0.654979	0.339172	0.354562	0.434095	0.358185
372	0.359829	0.531223	0.323106	0.314843	0.387277
373	0.652649	0.284249	0.295977	0.364234	0.281160
374	0.425511	0.748903	0.464533	0.535746	0.643042
375	0.513338	0.537218	0.540090	0.589831	0.580839
376	0.303575	0.499499	0.405127	0.354244	0.410228
377	0.573025	0.655399	0.385240	0.508841	0.563664
378	0.479740	0.513057	0.247739	0.298732	0.352490
379	0.568543	0.586813	0.401535	0.502922	0.529642
380	0.416105	0.637036	0.643073	0.664518	0.693295
381	0.442479	0.670970	0.312288	0.383002	0.490430
382	0.354973	0.734883	0.527543	0.555622	0.660604
383	0.278606	0.633255	0.348369	0.326561	0.450774
384	0.486140	0.554837	0.418874	0.470254	0.500733
385	0.470621	0.485679	0.464949	0.486166	0.484436
386	0.259072	0.744484	0.443048	0.434321	0.584007
387	0.358048	0.267016	0.602251	0.498072	0.411349
388	0.521667	0.908084	0.463276	0.624512	0.768775
389	0.484619	0.470569	0.323586	0.359103	0.379286
390	0.340426	0.557586	0.464694	0.442566	0.498519
391	0.449203	0.687023	0.445060	0.512464	0.595307
392	0.412119	0.549154	0.330964	0.352235	0.417286
393	0.252686	0.515682	0.497687	0.418974	0.472823
394	0.454766	0.645382	0.717625	0.753902	0.760379
395	0.369133	0.379536	0.334141	0.288103	0.299300
396	0.340276	0.523652	0.219848	0.208571	0.305345
397	0.425487	0.347697	0.479157	0.439743	0.394071
398	0.212875	0.427005	0.573303	0.444920	0.458567
399	0.376909	0.497986	0.339121	0.328736	0.381228
400	0.385607	0.538759	0.402823	0.402514	0.454263
401	0.318821	0.254937	0.645186	0.515201	0.423870
402	0.622309	0.433804	0.263640	0.360671	0.347662
403	0.310354	0.373413	0.383388	0.303191	0.315247
404	0.432435	0.353460	0.490175	0.454786	0.407216
405	0.394335	0.595919	0.517914	0.527930	0.573829
406	0.086620	0.387614	0.425103	0.237121	0.298268
407	0.419807	0.497633	0.392669	0.398523	0.429051
408	0.418372	0.534528	0.424701	0.437283	0.474930
409	0.528544	0.342125	0.589601	0.589416	0.493185
410	0.451847	0.346167	0.562818	0.528843	0.458102
411	0.367909	0.263063	0.402470	0.318447	0.271533
412	0.831249	0.532313	0.472065	0.679847	0.608713
413	0.451374	0.621867	0.209079	0.279211	0.388736
414	0.563006	0.412146	0.356418	0.411212	0.383884
415	0.476884	0.515784	0.289585	0.336492	0.382807
416	0.437414	0.791615	0.340587	0.439420	0.586906
417	0.426532	0.290854	0.347026	0.303506	0.265212

418	0.758117	0.594949	0.224060	0.433987	0.457672
419	0.444509	0.809091	0.253765	0.367950	0.539239
420	0.575910	0.508367	0.432339	0.513358	0.502325
421	0.324354	0.538537	0.498265	0.460402	0.505714
422	0.372111	0.518681	0.353062	0.344850	0.403159
423	0.542612	0.533349	0.476561	0.544643	0.541155
424	0.232486	0.386125	0.573271	0.443228	0.436984
425	0.624522	0.544775	0.457313	0.569716	0.555330
426	0.494841	0.620823	0.171791	0.265736	0.372736
427	0.542927	0.583694	0.445733	0.530237	0.552200
428	0.677891	0.378839	0.803902	0.868307	0.703687
429	0.333100	0.609362	0.324043	0.324083	0.431803
430	0.415060	0.604562	0.629501	0.642703	0.662566
431	0.432263	0.635093	0.387600	0.437383	0.517390
432	0.392633	0.773986	0.335657	0.408428	0.560997
433	0.370836	0.424956	0.463889	0.420369	0.419793
434	0.431092	0.447231	0.450103	0.442939	0.439443
435	0.226293	0.313240	0.657437	0.497586	0.447222
436	0.537067	0.478855	0.444966	0.498108	0.482485
437	0.494280	0.434692	0.285675	0.319205	0.331987
438	0.416386	0.741458	0.181738	0.269812	0.438225
439	0.480600	0.535006	0.360551	0.408650	0.445768
440	0.502869	0.772229	0.740197	0.832473	0.870004
441	0.545986	0.511816	0.534181	0.593274	0.568369
442	0.573138	0.280212	0.444710	0.461150	0.362896
443	0.403190	0.547660	0.403876	0.414412	0.465098
444	0.335431	0.549290	0.462038	0.435451	0.490067
445	0.336403	0.700640	0.357695	0.381450	0.515119
446	0.389439	0.545569	0.371178	0.377187	0.437482
447	0.297996	0.294378	0.455110	0.341471	0.311285
448	0.454711	0.353247	0.508320	0.482152	0.425275
449	0.291669	0.749383	0.594257	0.590172	0.700968
450	0.368760	0.388349	0.623576	0.555909	0.507295
451	0.378646	0.360964	0.509115	0.448194	0.412008
452	0.456451	0.500494	0.517104	0.531209	0.526974
453	0.374552	0.447352	0.763007	0.702742	0.644345
454	0.360836	0.557762	0.430899	0.421477	0.480047
455	0.529694	0.492174	0.392168	0.449727	0.452346
456	0.574149	0.288834	0.300696	0.331845	0.268012
457	0.406861	0.789424	0.481703	0.553531	0.676615
458	0.363295	0.749232	0.639874	0.666634	0.750481
459	0.462639	0.518172	0.348303	0.384132	0.421887
460	0.525598	0.662897	0.604191	0.688854	0.709887
461	0.431290	0.605441	0.646086	0.666011	0.678745
462	0.403433	0.830071	0.418583	0.505039	0.657888
463	0.517173	0.368796	0.484955	0.495166	0.434428
464	0.602951	0.443361	0.425490	0.502426	0.462227
465	0.391937	0.639128	0.730459	0.733583	0.749777
466	0.247267	0.580194	0.297377	0.250142	0.373089
467	0.205619	0.355237	0.409586	0.271613	0.295926
468	0.577826	0.494106	0.609802	0.673234	0.617681
469	0.417979	0.655291	0.329598	0.382760	0.486347
470	0.572581	0.425762	0.310472	0.377397	0.362919

471	0.452729	0.493767	0.399860	0.419992	0.439723
472	0.269140	0.420953	0.503011	0.405985	0.419417
473	0.307878	0.499982	0.487010	0.431592	0.468862
474	0.375830	0.545064	0.359761	0.359991	0.425807
475	0.340757	0.348276	0.710584	0.611274	0.535324
476	0.129353	0.525696	0.483990	0.349481	0.439218
477	0.320319	0.418080	0.541569	0.465337	0.457178
478	0.478675	0.220793	0.345153	0.307902	0.231534
479	0.435736	0.577518	0.586356	0.605739	0.620037
480	0.489020	0.491760	0.309406	0.353999	0.384148
481	0.671399	0.279937	0.268562	0.346971	0.263840
482	0.591776	0.297588	0.485112	0.511977	0.406982
483	0.518168	0.223687	0.257809	0.247648	0.182092
484	0.392514	0.382013	0.653934	0.593527	0.530304
485	0.581332	0.555226	0.383183	0.483654	0.499565
486	0.319400	0.331807	0.593597	0.489107	0.437598
487	0.423764	0.561190	0.599059	0.607152	0.615412
488	0.278364	0.461953	0.592278	0.503538	0.510607
489	0.500724	0.607237	0.500702	0.566682	0.595411
490	0.522172	0.417706	0.672541	0.683050	0.598435
491	0.506691	0.607298	0.260913	0.349557	0.429249
492	0.196577	0.463742	0.279263	0.177246	0.272657
493	0.211761	0.810911	0.513324	0.494036	0.664365
494	0.520999	0.752197	0.390525	0.514923	0.617030
495	0.370706	0.253102	0.528156	0.432411	0.353677
496	0.442902	0.337831	0.728584	0.674349	0.566421
497	0.528653	0.522397	0.433749	0.495620	0.500690
498	0.578590	0.613793	0.406010	0.519245	0.552686
499	0.453252	0.260548	0.479135	0.429387	0.344620
500	0.321570	0.547284	0.424369	0.393635	0.459003
501	0.416786	0.555678	0.247374	0.279570	0.364203
502	0.557710	0.308462	0.638453	0.639169	0.512811
503	0.676915	0.309818	0.391432	0.470533	0.370428
504	0.441405	0.512946	0.802978	0.789643	0.731201
505	0.374452	0.402315	0.550639	0.495545	0.466722
506	0.530415	0.589578	0.248720	0.345011	0.415141
507	0.406672	0.291309	0.564281	0.493375	0.412515
508	0.514038	0.651873	0.553806	0.634023	0.664674
509	0.749762	0.311367	0.659189	0.751883	0.576676
510	0.445753	0.483566	0.401231	0.415094	0.432366
511	0.486403	0.228900	0.448732	0.408895	0.311112
512	0.086672	0.433447	0.153037	0.000000	0.137642
513	0.301381	0.596561	0.297864	0.281227	0.397383
514	0.640397	0.775654	0.194246	0.398968	0.524455
515	0.379213	0.297654	0.375821	0.308894	0.278040
516	0.325185	0.334397	0.333249	0.253718	0.258648
517	0.530810	0.600740	0.543506	0.618740	0.628585
518	0.361412	0.742107	0.336574	0.385474	0.533323
519	0.278847	0.823308	0.141850	0.189005	0.429201
520	0.318538	0.378126	0.440639	0.360968	0.360349
521	0.554196	0.613233	0.640229	0.722210	0.710074
522	0.385537	0.490009	0.412252	0.397837	0.429346
523	0.453878	0.415731	0.283961	0.292919	0.308548

524	0.619661	0.783569	0.454989	0.630352	0.706771
525	0.492484	0.527120	0.500402	0.540573	0.541406
526	0.607898	0.315039	0.281706	0.337890	0.280006
527	0.396602	0.635706	0.529507	0.550514	0.608199
528	0.429810	0.747049	0.483149	0.554402	0.655924
529	0.335226	0.493302	0.307075	0.277890	0.345485
530	0.408499	0.514938	0.515902	0.510851	0.523608
531	0.662357	0.376546	0.655897	0.724360	0.594869
532	0.638531	0.604044	0.569706	0.695786	0.675677
533	0.362115	0.812562	0.300085	0.371546	0.553498
534	0.406691	0.351049	0.565221	0.510538	0.451771
535	0.330741	0.471749	0.399603	0.354747	0.395154
536	0.599264	0.469911	0.398769	0.483364	0.459782
537	0.422387	0.418335	0.561524	0.533088	0.496536
538	0.401124	0.500749	0.542478	0.527800	0.531201
539	0.453591	0.505699	0.460849	0.479627	0.490293
540	0.546312	0.580047	0.662295	0.729595	0.702116
541	0.280628	0.678850	0.578586	0.551218	0.641714
542	0.362130	0.896466	0.601282	0.670811	0.818325
543	0.263137	0.642466	0.141304	0.131589	0.308105
544	0.575806	0.606199	0.309220	0.427014	0.479409
545	0.424358	0.564427	0.408209	0.433199	0.484194
546	0.336296	0.494315	0.705089	0.643899	0.624723
547	0.412171	0.409015	0.383764	0.362493	0.363686
548	0.608293	0.302529	0.265155	0.319482	0.260449
549	0.544986	0.520744	0.447033	0.515259	0.512950
550	0.351491	0.732618	0.555795	0.579244	0.678035
551	0.431263	0.394173	0.489237	0.464461	0.432573
552	0.464781	0.796884	0.487057	0.588494	0.699500
553	0.390765	0.678484	0.419157	0.458101	0.557228
554	0.488938	0.589450	0.467753	0.525897	0.557966
555	0.398474	0.348075	0.328896	0.288903	0.282563
556	0.505918	0.516486	0.321295	0.379825	0.412614
557	0.153791	0.346998	0.349742	0.189385	0.235940
558	0.408240	0.656721	0.282822	0.335518	0.452154
559	0.601288	0.267145	0.231222	0.275308	0.212129
560	0.501100	0.453956	0.357745	0.393888	0.396514
561	0.549605	0.599193	0.396034	0.492090	0.529111
562	0.256863	0.627183	0.351167	0.316956	0.443432
563	0.468563	0.377817	0.321234	0.323885	0.313750
564	0.452279	0.695856	0.568807	0.629909	0.688307
565	0.549137	0.628138	0.491116	0.587004	0.614182
566	0.344562	0.510825	0.459738	0.427267	0.465867
567	0.345344	0.570135	0.569398	0.544443	0.581059
568	0.499591	0.602241	0.473334	0.539660	0.572766
569	0.517447	0.386288	0.402735	0.424624	0.388301
570	0.451876	0.554700	0.409944	0.445450	0.485928
571	0.675976	0.799674	0.384548	0.597348	0.681844
572	0.434327	0.550607	0.400201	0.426904	0.472131
573	0.427347	0.213654	0.323822	0.261556	0.199314
574	0.398441	0.618259	0.556960	0.571836	0.616580
575	0.470122	0.409452	0.100996	0.131177	0.180566
576	0.368751	0.466937	0.315666	0.294801	0.342750

577	0.367622	0.396548	0.426199	0.376483	0.374290
578	0.160041	0.507688	0.421601	0.302166	0.391546
579	0.648934	0.489796	0.531468	0.634575	0.577703
580	0.525020	0.437031	0.389381	0.429872	0.413618
581	0.602291	0.982247	0.473542	0.693155	0.843802
582	0.232422	0.386194	0.457845	0.337302	0.356299
583	0.531804	0.642869	0.485518	0.577500	0.615499
584	0.292913	0.708517	0.452748	0.449782	0.575923
585	0.232174	0.742061	0.106097	0.111466	0.340171
586	0.632755	0.600422	0.237113	0.386820	0.439336
587	0.543995	0.688019	0.506368	0.614841	0.662261
588	0.488503	0.543283	0.598281	0.632869	0.619308
589	0.578386	0.365541	0.361873	0.410948	0.361393
590	0.343293	0.504840	0.435463	0.402747	0.444712
591	0.582372	0.543533	0.497040	0.585443	0.571885
592	0.517774	0.515555	0.509525	0.558023	0.546567
593	0.533304	0.619722	0.230176	0.337613	0.422362
594	0.422197	0.708442	0.594217	0.642107	0.706768
595	0.612106	0.675558	0.581223	0.713075	0.723397
596	0.392576	0.449462	0.377440	0.358242	0.380549
597	0.300204	0.302748	0.533034	0.416325	0.371729
598	0.616723	0.741644	0.515583	0.673099	0.721332
599	0.415689	0.419179	0.516058	0.488358	0.463631
600	0.687347	0.495598	0.223574	0.372216	0.375648
601	0.329929	0.455687	0.215948	0.181454	0.256153
602	0.538358	0.337284	0.608184	0.609894	0.505479
603	0.691481	0.525958	0.254341	0.410727	0.417798
604	0.323419	0.119611	0.638153	0.474068	0.332664
605	0.363116	0.440134	0.327042	0.295204	0.331996
606	0.280711	0.572427	0.452776	0.406794	0.485007
607	0.405325	0.308296	0.310128	0.264147	0.245434
608	0.547786	0.392025	0.616935	0.637411	0.549293
609	0.310236	0.568944	0.410601	0.381426	0.460566
610	0.454787	0.610407	0.368937	0.424421	0.493961
611	0.336125	0.512280	0.246798	0.228192	0.315818
612	0.387096	0.594494	0.697154	0.688509	0.696455
613	0.630045	0.624914	0.383355	0.526378	0.556751
614	0.502903	0.049720	0.272077	0.205915	0.075907
615	0.743847	0.376578	0.514888	0.634396	0.516437
616	0.767136	0.396793	0.575276	0.706585	0.577482
617	0.599582	0.459293	0.059724	0.169517	0.215917
618	0.616490	0.453649	0.663465	0.730144	0.638630
619	0.467974	0.710313	0.521815	0.598324	0.668668
620	0.433774	0.671386	0.361386	0.423958	0.522879
621	0.304890	0.467935	0.317753	0.266098	0.329063
622	0.511654	0.312652	0.713412	0.686816	0.556545
623	0.466532	0.598219	0.468849	0.518457	0.558857
624	0.761038	0.653544	0.406806	0.619065	0.624037
625	0.732189	0.673297	0.484546	0.681831	0.684026
626	0.402279	0.589047	0.507914	0.520722	0.564361
627	0.435834	0.414581	0.504356	0.486110	0.457459
628	0.489108	0.527116	0.317661	0.371257	0.412783
629	0.290316	0.000000	0.419546	0.224846	0.094340

630	0.453492	0.483516	0.475423	0.486902	0.486128
631	0.269451	0.733281	0.207343	0.220005	0.414514
632	0.743273	0.662481	0.345965	0.557082	0.582873
633	0.446135	0.533442	0.614398	0.624480	0.613744
634	0.593203	0.432152	0.323382	0.400961	0.381174
635	0.142643	0.319846	0.172086	0.013573	0.091413
636	0.215932	0.576674	0.474257	0.396330	0.486753
637	0.499004	0.722949	0.440422	0.542093	0.627588
638	0.355431	0.651453	0.701445	0.692663	0.728423
639	0.625296	0.265318	0.264431	0.316894	0.240107
640	0.167772	0.410427	0.634954	0.475153	0.479816
641	0.394566	0.380588	0.407168	0.367700	0.357334
642	0.516471	0.328169	0.549918	0.543357	0.453434
643	0.487326	0.415182	0.366432	0.384623	0.374134
644	0.344588	0.534501	0.681026	0.636789	0.635909
645	0.353690	0.333205	0.459784	0.383288	0.353408
646	0.142805	0.440321	0.603150	0.442045	0.470714
647	0.327458	0.385203	0.194102	0.140992	0.194731
648	0.351071	0.174153	0.102257	0.010583	0.000000
649	0.569664	0.378739	0.519894	0.555327	0.478263
650	0.166315	0.427293	0.602984	0.449713	0.467996
651	0.354533	0.637473	0.547012	0.546710	0.611179
652	0.358138	0.407812	0.550391	0.488925	0.466065
653	0.495264	0.495109	0.180148	0.239326	0.297469
654	0.688294	0.651435	0.272708	0.460257	0.510916
655	0.373888	0.511044	0.428978	0.413287	0.451750
656	0.420278	0.350320	0.397467	0.362982	0.337354
657	0.486923	0.729478	0.362207	0.466261	0.574125
658	0.494251	0.446299	0.518735	0.536210	0.502450
659	0.755278	0.372689	0.392380	0.526452	0.431084
660	0.320503	0.691252	0.449581	0.455513	0.569377
661	0.518461	0.372212	0.612250	0.613526	0.525967
662	0.678014	0.708100	0.663014	0.828878	0.817911
663	0.512257	0.602905	0.753163	0.802736	0.772004
664	0.424804	0.694427	0.539780	0.589595	0.660292
665	0.596789	0.726199	0.476835	0.623690	0.679329
666	0.405269	0.549469	0.466587	0.473454	0.510632
667	0.281475	0.408591	0.379965	0.295674	0.328439
668	0.271894	0.520981	0.348935	0.293216	0.376978
669	0.426410	0.182430	0.423453	0.344008	0.248580
670	0.429119	0.586065	0.436464	0.467330	0.519108
671	0.532576	0.431242	0.573347	0.600753	0.540388
672	0.257427	0.503632	0.382623	0.312400	0.385750
673	0.581814	0.773994	0.635357	0.774939	0.817353
674	0.474064	0.359618	0.597421	0.575009	0.496482
675	0.816193	0.688280	0.571768	0.806584	0.775469
676	0.584601	0.557592	0.561661	0.649651	0.626707
677	0.568920	0.334039	0.552661	0.572844	0.472114
678	0.434612	0.418752	0.344671	0.340131	0.348189
679	0.458810	0.455701	0.456183	0.464234	0.456020
680	0.378816	0.620394	0.207266	0.242049	0.368575
681	0.378169	0.283772	0.394051	0.321331	0.281561
682	0.505715	0.590189	0.621642	0.675421	0.670202

683	0.305590	0.577346	0.273509	0.255675	0.368980
684	0.599028	0.169080	0.351607	0.357936	0.232399
685	0.285635	0.421531	0.339584	0.264162	0.309590
686	0.137673	0.697403	0.587173	0.495010	0.624356
687	0.488133	0.617486	0.380236	0.452849	0.514681
688	0.510973	0.569168	0.529986	0.588129	0.593830
689	0.498609	0.383625	0.566205	0.564785	0.496233
690	0.646080	0.688534	0.335391	0.507472	0.568276
691	0.228673	0.796103	0.347360	0.345891	0.542926
692	0.407413	0.194844	0.441578	0.354836	0.264576
693	0.661012	0.369886	0.439482	0.523311	0.438901
694	0.563995	0.434347	0.207604	0.281195	0.294410
695	0.278392	0.474888	0.358431	0.292503	0.355448
696	0.580261	0.592424	0.477737	0.580042	0.589451
697	0.753294	0.427914	0.526942	0.664025	0.560366
698	0.438051	0.508122	0.297369	0.322762	0.373698
699	0.506614	0.431567	0.285325	0.323998	0.332773
700	0.510588	0.422224	0.263788	0.303610	0.312660
701	0.465842	0.856643	0.520368	0.635869	0.761661
702	0.436988	0.538522	0.352834	0.381432	0.431860
703	0.558749	0.543088	0.336237	0.426343	0.453311
704	0.511843	0.507552	0.315640	0.375064	0.404352
705	0.355433	0.660136	0.372211	0.392929	0.503809
706	0.578197	0.680227	0.511354	0.633835	0.669171
707	0.637878	0.538542	0.508675	0.621606	0.590522
708	0.304578	0.577352	0.263521	0.246022	0.361749
709	0.240270	0.493076	0.149729	0.087520	0.211832
710	0.411599	0.784166	0.499264	0.570503	0.686669
711	0.289591	0.567253	0.169538	0.149781	0.285800
712	0.437275	0.430101	0.489971	0.477840	0.457783
713	0.458402	0.435353	0.219940	0.241714	0.277576
714	0.399926	0.735729	0.311246	0.379124	0.521015
715	0.538811	0.509638	0.317056	0.389977	0.413359
716	0.386219	0.524962	0.721428	0.691397	0.668293
717	0.314476	0.532469	0.500057	0.455614	0.500604
718	0.316159	0.492424	0.497133	0.442824	0.473105
719	0.486536	0.229739	0.478256	0.436279	0.332333
720	0.628107	0.409278	0.119205	0.224254	0.232252
721	0.395426	0.609292	0.389848	0.414592	0.493185
722	0.261729	0.631578	0.274178	0.249863	0.393638
723	0.271891	0.541247	0.385019	0.331851	0.415302
724	0.391560	0.602958	0.297055	0.325849	0.423250
725	0.429168	0.575106	0.549660	0.568233	0.591195
726	0.346641	0.694745	0.506101	0.520971	0.617619
727	0.348774	0.767179	0.434564	0.476115	0.614918
728	0.346579	0.885333	0.470327	0.540089	0.715714
729	0.595442	0.587402	0.502172	0.608436	0.607046
730	0.305596	0.826086	0.463056	0.497437	0.662221
731	0.356264	0.603339	0.690015	0.669457	0.689554
732	0.558391	0.262265	0.272118	0.290754	0.226966
733	0.305235	0.278445	0.416063	0.304799	0.275478
734	0.609526	0.669520	0.502090	0.637567	0.663522
735	0.471267	0.455909	0.743253	0.733730	0.659974

736	0.354309	0.179387	0.494970	0.373927	0.278795
737	0.438495	0.891193	0.484804	0.599430	0.752351
738	0.071294	0.590486	0.434552	0.293702	0.432147
739	0.433962	0.492614	0.399620	0.410380	0.434169
740	0.596645	0.590742	0.517135	0.623660	0.619965
741	0.417366	0.473073	0.441260	0.435232	0.446559
742	0.343655	0.663910	0.502602	0.507907	0.594514
743	0.619564	0.594872	0.351231	0.483640	0.512289
744	0.364701	0.693032	0.460332	0.487243	0.588974
745	0.243200	0.237956	0.473040	0.316033	0.273823
746	0.403410	0.186426	0.458562	0.366188	0.270024
747	0.553909	0.352262	0.523830	0.544097	0.460014
748	0.423552	0.741953	0.503848	0.568979	0.665560
749	0.580897	0.602028	0.346301	0.462365	0.503903
750	0.556961	0.420445	0.586368	0.621552	0.548548
751	0.379281	0.417110	0.514870	0.469093	0.452462
752	0.511876	0.542053	0.266351	0.339262	0.392183
753	0.452242	0.652211	0.529631	0.582042	0.632707
754	0.243536	0.456539	0.332372	0.246728	0.316752
755	0.682242	0.451556	0.239610	0.372451	0.357147
756	0.272374	0.554106	0.400471	0.349771	0.434534
757	0.166516	0.712069	0.474696	0.409753	0.562311
758	0.212007	0.583927	0.437970	0.363113	0.465095
759	0.342019	0.471210	0.642464	0.582902	0.567421
760	0.570281	0.396054	0.480207	0.523931	0.461849
761	0.373593	0.640870	0.516251	0.528630	0.596577
762	0.466245	0.253915	0.618576	0.561813	0.441055
763	0.234574	0.278688	0.462008	0.312846	0.290290
764	0.198570	0.548614	0.534889	0.435915	0.506731
765	0.573924	0.721224	0.511601	0.643174	0.694772
766	0.322962	0.472196	0.236090	0.201068	0.279180
767	0.324745	0.419403	0.165923	0.123149	0.196449
768	0.677063	0.326102	0.521234	0.594151	0.471751
769	0.334732	0.169317	0.298428	0.181366	0.130013
770	0.566332	0.304764	0.346067	0.374039	0.308095
771	0.315487	0.468028	0.105013	0.076041	0.182981
772	0.301298	0.740130	0.496177	0.502309	0.628787
773	0.472948	0.506988	0.670178	0.681419	0.642289
774	0.578033	0.482381	0.628263	0.687077	0.623067
775	0.058171	0.746912	0.422997	0.319410	0.521878
776	0.472072	0.441633	0.679716	0.671925	0.606521
777	0.419066	0.828308	0.006427	0.133929	0.372406
778	0.443175	0.406636	0.241867	0.246637	0.270591
779	0.540932	0.721847	0.302813	0.435804	0.541017
780	0.276007	0.608620	0.421011	0.385241	0.485013
781	0.514065	0.421410	0.476091	0.499877	0.461451
782	0.411725	0.504799	0.722916	0.699601	0.662615
783	0.697052	0.498305	0.325810	0.471459	0.451287
784	0.472476	0.312447	0.546689	0.514826	0.430141
785	0.514322	0.357990	0.227582	0.254677	0.246768
786	0.328314	0.592422	0.331715	0.324188	0.425041
787	0.467118	0.670277	0.675783	0.728271	0.750256
788	0.696274	0.402205	0.359047	0.475375	0.412254

789	0.446783	0.529349	0.214496	0.256731	0.331619
790	0.462663	0.418530	0.405511	0.409465	0.397525
791	0.573836	0.309300	0.475710	0.497865	0.403538
792	0.297376	0.293670	0.293762	0.192927	0.197848
793	0.499465	0.453794	0.307996	0.347403	0.361217
794	0.451913	0.375122	0.374763	0.364214	0.345324
795	0.234567	0.446015	0.378308	0.281670	0.339857
796	0.309201	0.698906	0.358493	0.368553	0.507832
797	0.392951	0.386875	0.459263	0.416436	0.397425
798	0.282552	0.513732	0.586800	0.514658	0.541263
799	0.338696	0.266653	0.272376	0.185922	0.175657
800	0.295525	0.488459	0.181666	0.142293	0.244845
801	0.351748	0.432703	0.670147	0.602509	0.564307
802	0.352432	0.464649	0.542301	0.494242	0.495716
803	0.512231	0.593138	0.557883	0.620872	0.629133
804	0.538235	0.334989	0.269877	0.298780	0.267397
805	0.285249	0.423880	0.291338	0.220345	0.277274
806	0.224225	0.623547	0.462527	0.402360	0.510883
807	0.639440	0.475317	0.322194	0.434006	0.419663
808	0.238669	0.379675	0.402445	0.287711	0.314893
809	0.523672	0.501363	0.454918	0.506899	0.500672
810	0.431648	0.564224	0.482405	0.504752	0.537748
811	0.466260	0.253887	0.732518	0.666366	0.520717
812	0.327337	0.509039	0.249970	0.225967	0.313769
813	0.522620	0.583665	0.205236	0.299728	0.378987
814	0.062348	0.414144	0.538772	0.336918	0.388890
815	0.316036	0.469855	0.691707	0.615150	0.594555
816	0.524986	0.480153	0.143928	0.216389	0.269829
817	0.457174	0.437006	0.446451	0.449415	0.436732
818	0.435951	0.345271	0.510439	0.472848	0.416966
819	0.307582	0.699864	0.170004	0.195075	0.376245
820	0.394230	0.609728	0.431273	0.452144	0.522138
821	0.462744	0.530228	0.371684	0.408925	0.446051
822	0.329621	0.550777	0.261270	0.248823	0.349199
823	0.408104	0.331899	0.368387	0.325386	0.302108
824	0.467757	0.433548	0.417445	0.426975	0.416832
825	0.198575	0.346446	0.366047	0.225858	0.258059
826	0.712476	0.567996	0.625639	0.773046	0.709786
827	0.547667	0.403105	0.376029	0.419322	0.387963
828	0.414212	0.366231	0.363658	0.333364	0.322491
829	0.344794	0.468796	0.439407	0.397263	0.424555
830	0.528015	0.673349	0.608229	0.696578	0.720061
831	0.351380	0.475064	0.337290	0.308456	0.358826
832	0.327334	0.449048	0.357262	0.308057	0.350039
833	0.573127	0.318233	0.686756	0.693613	0.556712
834	0.700701	0.744248	0.207474	0.431710	0.528328
835	0.403962	0.461160	0.655821	0.622380	0.585585
836	0.449535	0.410460	0.457204	0.448347	0.425213
837	0.371608	0.431720	0.507229	0.462355	0.454659
838	0.477258	0.548610	0.391800	0.439416	0.475582
839	0.385126	0.616011	0.364507	0.388189	0.477258
840	0.390803	0.612060	0.385183	0.408830	0.490568
841	0.640199	0.733219	0.198641	0.391333	0.500066

842	0.224177	0.564635	0.348664	0.281791	0.393191
843	0.346034	0.583964	0.533812	0.515894	0.565279
844	0.453832	0.351815	0.501164	0.474771	0.419129
845	0.441645	0.476844	0.446502	0.452815	0.458665
846	0.266266	0.404997	0.518464	0.414424	0.419204
847	0.442121	0.488420	0.633691	0.627965	0.597157
848	0.426204	0.420544	0.551626	0.526454	0.491985
849	0.332349	0.363261	0.412016	0.337330	0.334146
850	0.346033	0.506718	0.490237	0.454845	0.484905
851	0.449792	0.370694	0.549138	0.521985	0.463873
852	0.467715	0.605679	0.000000	0.090851	0.236116
853	0.504024	0.614236	0.373629	0.453586	0.511891
854	0.204584	0.290528	0.677494	0.499296	0.441206
855	0.344791	0.271159	0.292848	0.208884	0.194391
856	0.474854	0.613832	0.279490	0.352986	0.438588
857	0.752694	0.652828	0.639547	0.828395	0.784260
858	0.518988	0.411803	0.614283	0.626442	0.553096
859	0.565984	0.537535	0.400445	0.487245	0.496412
860	0.411198	0.326306	0.643390	0.577698	0.491562
861	0.466964	0.581620	0.570234	0.607170	0.619136
862	0.383050	0.334075	0.525605	0.458123	0.407257
863	0.404610	0.588713	0.550636	0.560960	0.594595
864	0.419231	0.608365	0.616044	0.633409	0.656644
865	0.565917	0.360424	0.538456	0.565552	0.478484
866	0.523274	0.465038	0.667774	0.692116	0.625952
867	0.404608	0.495935	0.467602	0.459466	0.476594
868	0.540288	0.369654	0.358480	0.390528	0.352257
869	0.650153	0.677403	0.501290	0.658635	0.678102
870	0.418660	0.337818	0.518921	0.470235	0.413807
871	0.444348	0.398727	0.357258	0.350930	0.346462
872	0.472856	0.782746	0.381515	0.491700	0.618560
873	0.468906	0.602110	0.347793	0.409586	0.477307
874	0.378948	0.556350	0.594782	0.580230	0.598212
875	0.470084	0.767398	0.125019	0.250814	0.428599
876	0.372835	0.696059	0.418963	0.454042	0.564012
877	0.358549	0.406897	0.350884	0.305808	0.326067
878	0.432073	0.457168	0.217253	0.232462	0.283279
879	0.383352	0.719511	0.351576	0.403691	0.534642
880	0.287747	0.316747	0.333588	0.231106	0.238225
881	0.388228	0.212993	0.281405	0.203532	0.159553
882	0.343051	0.533226	0.229693	0.221558	0.319101
883	0.520529	0.702929	0.381945	0.493387	0.579086
884	0.481544	0.438315	0.485659	0.497537	0.471021
885	0.433051	0.468092	0.598544	0.585783	0.557204
886	0.620495	0.962077	0.308008	0.544567	0.719519
887	0.506792	0.568219	0.552581	0.606581	0.607982
888	0.444035	0.746425	0.617790	0.684658	0.753189
889	0.497281	0.343272	0.565299	0.552307	0.469202
890	0.485910	0.410281	0.233938	0.261026	0.277968
891	0.362874	0.173657	0.340924	0.235155	0.169491
892	0.650846	0.467862	0.533297	0.631197	0.565285
893	0.403150	0.776549	0.728325	0.774524	0.839835
894	0.654729	0.502503	0.556609	0.663913	0.604926

895	0.449468	0.535955	0.282006	0.321778	0.383759
896	0.386638	0.233364	0.595511	0.496540	0.391966
897	0.464578	0.310136	0.459393	0.430274	0.365651
898	0.482593	0.584860	0.235914	0.308843	0.391313
899	0.401684	0.540854	0.578257	0.571838	0.582268
900	0.553722	0.621922	0.197548	0.318150	0.406016
901	0.657990	0.615259	0.388980	0.542424	0.561358
902	0.417543	0.717548	0.384880	0.450253	0.565116
903	0.404275	0.541561	0.421576	0.429515	0.473803
904	0.536630	0.596804	0.427265	0.513820	0.546198
905	0.449355	0.262746	0.354892	0.314098	0.258197
906	0.424988	0.516460	0.550325	0.550828	0.552740
907	0.380995	0.611001	0.486717	0.496964	0.558458
908	0.482395	0.479024	0.751340	0.752836	0.683314
909	0.491981	0.408560	0.470029	0.480129	0.443450
910	0.665528	0.556947	0.379408	0.521385	0.518856
911	0.631076	0.420559	0.581063	0.652564	0.563238
912	0.682759	0.315746	0.607002	0.672782	0.526444
913	0.228213	0.738599	0.409639	0.387135	0.549213
914	0.419877	0.408973	0.602482	0.566903	0.518508
915	0.303260	0.599380	0.535809	0.501241	0.566057
916	0.317261	0.648478	0.578361	0.560448	0.630993
917	0.436520	0.572199	0.196224	0.246686	0.343987
918	0.268314	0.589726	0.453240	0.405941	0.493441
919	0.365927	0.348436	0.424884	0.361336	0.341869
920	0.126032	0.349665	0.304870	0.135511	0.199422
921	0.388243	0.507457	0.449315	0.437913	0.467204
922	0.246110	0.199180	0.458632	0.293645	0.239417
923	0.482929	0.619075	0.340003	0.413847	0.486287
924	0.303104	0.362364	0.423691	0.333653	0.334499
925	0.466555	0.390951	0.503602	0.493834	0.449263
926	0.708374	0.610281	0.538592	0.702720	0.675219
927	0.199951	0.287782	0.761849	0.573710	0.497274
928	0.238555	0.756854	0.439005	0.424061	0.584099
929	0.498208	0.571618	0.457188	0.515824	0.541350
930	0.612632	0.571649	0.428658	0.545000	0.549714
931	0.123227	0.485033	0.589441	0.432190	0.485172
932	0.543883	0.614924	0.590495	0.672048	0.673839
933	0.563561	0.596176	0.510338	0.602902	0.610542
934	0.292449	0.136507	0.559856	0.391851	0.281170
935	0.188380	0.206389	0.463452	0.272111	0.233170
936	0.366189	0.610933	0.240020	0.263416	0.382245
937	0.395869	0.517271	0.547910	0.534748	0.544374
938	0.465778	0.432854	0.239458	0.262510	0.291434
939	0.708425	0.611943	0.394583	0.571056	0.575605
940	0.266567	0.837131	0.306561	0.337973	0.550273
941	0.306060	0.396422	0.314073	0.243785	0.280579
942	0.297571	0.483129	0.704898	0.621943	0.607788
943	0.267554	0.588773	0.431486	0.385353	0.477426
944	0.485105	0.346333	0.346327	0.346325	0.315048
945	0.835198	0.462611	0.639895	0.816749	0.682019
946	0.000000	0.300332	0.626370	0.356103	0.361202
947	0.589133	0.239135	0.487669	0.497104	0.370354

948	0.344900	0.346034	0.343503	0.275836	0.278211
949	0.060431	0.438350	0.557374	0.359660	0.417062
950	0.471701	0.455692	0.364938	0.386742	0.395397
951	0.444150	0.496181	0.478281	0.488459	0.493999
952	0.713610	0.672951	0.374009	0.571322	0.601912
953	0.568490	0.638822	0.247971	0.376170	0.455846
954	0.520226	0.604019	0.504107	0.578362	0.600535
955	0.461011	0.735424	0.397130	0.487394	0.595979
956	0.549818	0.454745	0.523078	0.569376	0.524683
957	0.456000	0.675212	0.422734	0.492043	0.573745
958	0.608863	0.252726	0.423024	0.451036	0.338809
959	0.148614	0.515963	0.416460	0.294178	0.390471
960	0.684879	0.335653	0.334053	0.428780	0.348964
961	0.552723	0.572048	0.267997	0.368710	0.422812
962	0.172186	0.296848	0.313749	0.151580	0.182922
963	0.519626	0.522890	0.402600	0.462806	0.476995
964	0.183158	0.283307	0.564029	0.382849	0.351901
965	0.238917	0.571031	0.781622	0.687944	0.703723
966	0.411681	0.342539	0.506697	0.456930	0.406583
967	0.525748	0.679948	0.399830	0.506055	0.578035
968	0.593048	0.991396	0.117192	0.364195	0.598242
969	0.208866	0.223507	0.207769	0.052075	0.070503
970	0.465052	0.611462	0.490664	0.541369	0.582301
971	0.615731	0.291452	0.527764	0.561027	0.438767
972	0.563195	0.533829	0.466452	0.545453	0.539485
973	0.678408	0.318890	0.592114	0.657873	0.516988
974	0.232059	0.514237	0.395334	0.314684	0.395217
975	0.635154	0.796238	0.496875	0.679735	0.748076
976	0.578072	0.539988	0.531353	0.613881	0.592526
977	0.512930	0.577776	0.286435	0.367943	0.429566
978	0.636728	0.630021	0.254009	0.412316	0.471255
979	0.295378	0.424292	0.660233	0.563852	0.538003
980	0.540978	0.403199	0.436899	0.471966	0.428934
981	0.396459	0.642827	0.323591	0.363440	0.468774
982	0.496611	0.864202	0.448853	0.587192	0.724145
983	0.526861	0.503824	0.362653	0.424451	0.438533
984	0.504073	0.317884	0.485691	0.475621	0.398812
985	0.486950	0.764286	0.430897	0.538795	0.644651
986	0.613136	0.398439	0.176882	0.266981	0.261880
987	0.554387	0.460263	0.387589	0.448767	0.434635
988	0.678312	0.298114	0.335161	0.416383	0.323864
989	0.450408	0.364295	0.426431	0.407944	0.374087
990	0.434543	0.520614	0.261651	0.291698	0.355925
991	0.396393	0.556348	0.376782	0.388633	0.450084
992	0.327653	0.347684	0.294429	0.222914	0.240696
993	0.284390	0.299863	0.429722	0.313091	0.293711
994	0.291437	0.598929	0.538427	0.497802	0.564674
995	0.206271	0.515466	0.536292	0.431888	0.488202
996	0.444942	0.467279	0.553232	0.549735	0.527934
997	0.370273	0.594924	0.390510	0.399115	0.478145
998	0.449252	0.702832	0.363070	0.441565	0.548199
999	0.171507	0.661391	0.670761	0.578254	0.667908

Appendix B

National University Institutional Review Board Study Approval

National University

9388 Lightwave Ave.
San Diego, California 92123
(800) NAT-UNIV
NU.edu

Protocol ID Number: 2023-066
Date: February 21, 2023
PI Name: Leo Ravelo
Chair Name (if applicable): John Enamait
Application Type: Initial Submission
Review Level: N/A – Not Human Subjects Research (NHSR)
Study Title: Towards Multivariate Regression using Quantum Neural Networks

Date of NHSR Determination: February 21, 2023

Dear Leo:

Your study was determined to be Not Human Subjects Research (NHSR). IRB review and oversight are not required.

If you change your study to include human participants, please fill out a new application. Remember that we have office hours and group writing sessions to support you.

If you are conducting this study for your dissertation, you will upload this letter to the Doctoral Record for both Study Approval ***and*** Study Closure. Separate closure forms are not needed for NHSR studies.

Best wishes as you conduct your research!

Respectfully,

National University Institutional Review Board
Email: irb@nu.edu

Appendix C

Classical Neural Network Results

Table C1

Classical Neural Networks Raw Training Results Using 100 Samples

n_features	n_targets	n_samples	test_size	optimizer	iterations	training_time	prediction_time	r2	mae	mse	rmse
3	2	100	20	adam	100	0.0	0.000	-0.2061	0.2344	0.0772	0.2759
3	2	100	20	adam	1000	0.0	0.000	0.3175	0.1804	0.0455	0.2117
3	2	100	20	adam	10000	0.0	0.001	-0.0278	0.2206	0.0664	0.2571
3	2	100	20	adam	100000	0.0	0.000	0.1611	0.2009	0.0552	0.2347
3	2	100	20	lbfgs	100	0.0	0.000	0.8248	0.0907	0.0110	0.1012
3	2	100	20	lbfgs	1000	0.0	0.000	0.8294	0.0883	0.0107	0.0999
3	2	100	20	lbfgs	10000	0.0	0.000	0.8327	0.0867	0.0104	0.0976
3	2	100	20	lbfgs	100000	0.0	0.000	0.8438	0.0847	0.0097	0.0945
3	2	100	20	sgd	100	0.0	0.000	0.2479	0.1839	0.0486	0.2202
3	2	100	20	sgd	1000	0.0	0.000	0.0830	0.1983	0.0596	0.2440
3	2	100	20	sgd	10000	0.0	0.000	0.2787	0.1785	0.0478	0.2178
3	2	100	20	sgd	100000	0.0	0.001	-0.1310	0.2157	0.0736	0.2713
3	2	100	30	adam	100	0.0	0.000	0.5472	0.1062	0.0196	0.1365
3	2	100	30	adam	1000	0.0	0.000	0.5239	0.1092	0.0199	0.1410
3	2	100	30	adam	10000	0.0	0.000	0.6375	0.0934	0.0150	0.1222
3	2	100	30	adam	100000	0.0	0.000	0.3766	0.1261	0.0260	0.1612
3	2	100	30	lbfgs	100	0.0	0.001	0.9384	0.0414	0.0026	0.0508
3	2	100	30	lbfgs	1000	0.0	0.000	0.9393	0.0414	0.0026	0.0504
3	2	100	30	lbfgs	10000	0.0	0.000	0.9380	0.0420	0.0026	0.0510
3	2	100	30	lbfgs	100000	0.0	0.000	0.9427	0.0400	0.0024	0.0489
3	2	100	30	sgd	100	0.0	0.000	0.3693	0.1183	0.0257	0.1593
3	2	100	30	sgd	1000	0.0	0.000	0.2666	0.1335	0.0313	0.1752
3	2	100	30	sgd	10000	0.0	0.000	0.3516	0.1254	0.0270	0.1644
3	2	100	30	sgd	100000	0.0	0.000	0.2555	0.1357	0.0307	0.1751
3	2	100	40	adam	100	0.0	0.000	0.2440	0.1315	0.0278	0.1629
3	2	100	40	adam	1000	0.0	0.001	0.4475	0.1198	0.0206	0.1425
3	2	100	40	adam	10000	0.0	0.000	0.3822	0.1199	0.0235	0.1530
3	2	100	40	adam	100000	0.0	0.000	0.4354	0.1199	0.0215	0.1463
3	2	100	40	lbfgs	100	0.0	0.000	0.9367	0.0391	0.0024	0.0485
3	2	100	40	lbfgs	1000	0.0	0.000	0.9425	0.0356	0.0022	0.0463
3	2	100	40	lbfgs	10000	0.0	0.000	0.9383	0.0385	0.0023	0.0477
3	2	100	40	lbfgs	100000	0.0	0.000	0.9353	0.0393	0.0024	0.0492
3	2	100	40	sgd	100	0.0	0.000	0.2449	0.1256	0.0292	0.1677
3	2	100	40	sgd	1000	0.0	0.000	0.1430	0.1400	0.0324	0.1800
3	2	100	40	sgd	10000	0.0	0.001	0.2707	0.1231	0.0278	0.1662
3	2	100	40	sgd	100000	0.0	0.000	0.1684	0.1331	0.0316	0.1775

Figure C1

Classical Neural Networks Performance Metrics - 100 Samples and 80/20 Split

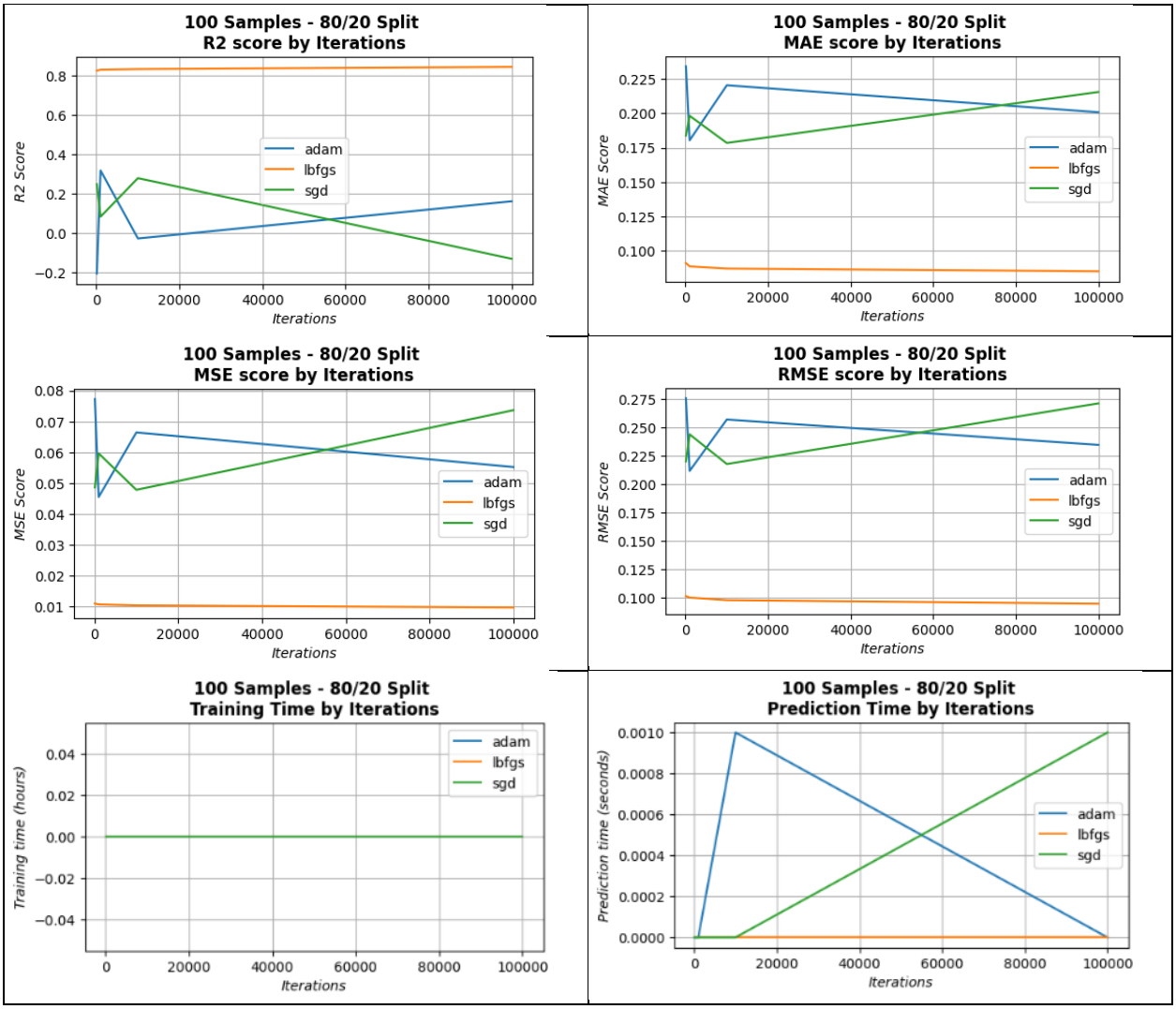


Figure C2

Classical Neural Networks Performance Metrics - 100 Samples and 70/30 Split

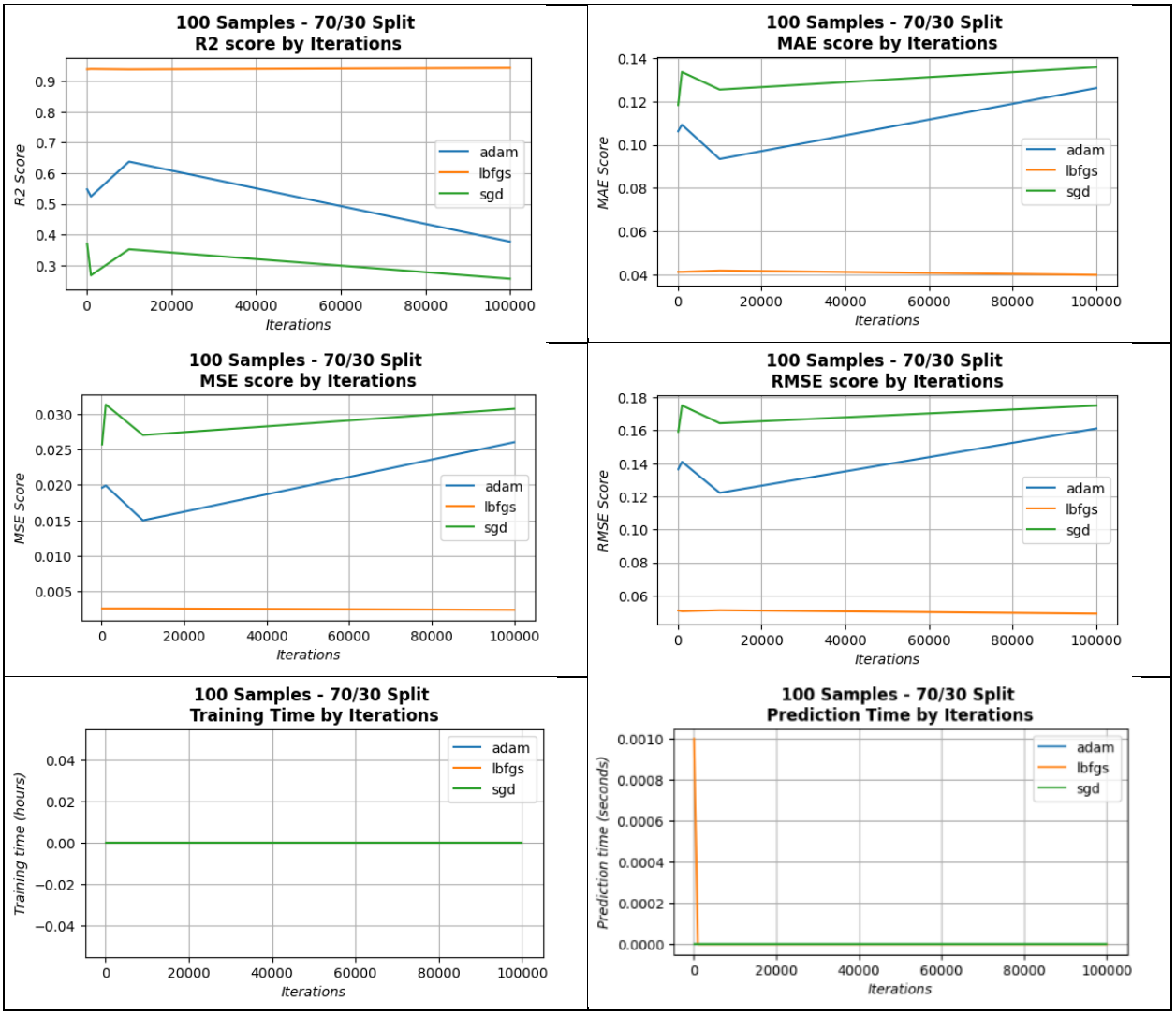


Figure C3

Classical Neural Networks Performance Metrics - 100 Samples and 60/40 Split

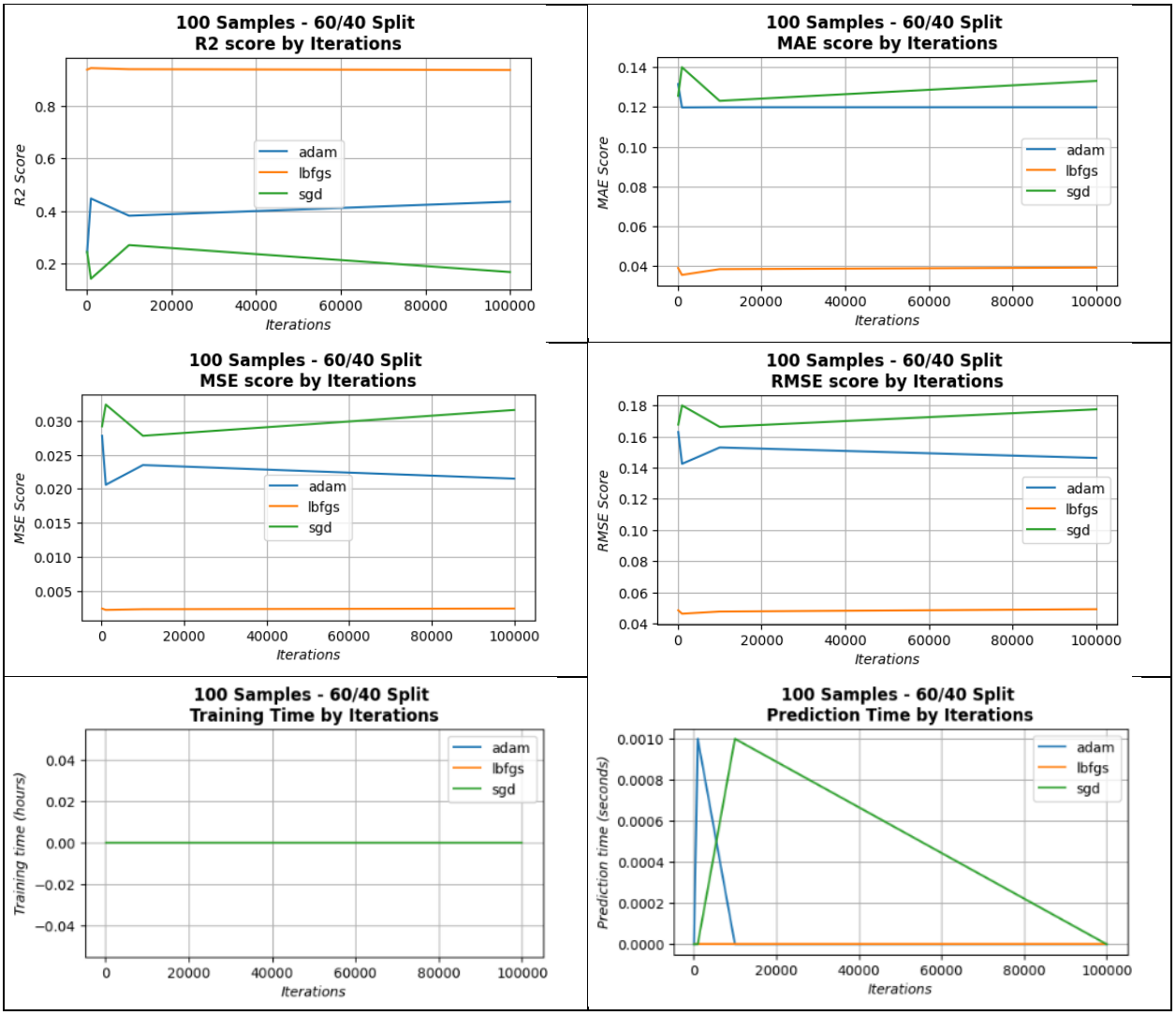


Table C2

Classical Neural Networks Raw Training Results Using 500 Samples

n_features	n_targets	n_samples	test_size	optimizer	iterations	training_time	prediction_time	r2	mae	mse	rmse
3	2	500	20	adam	100	0.0	0.000	0.7488	0.1226	0.0209	0.1437
3	2	500	20	adam	1000	0.0	0.000	0.9034	0.0704	0.0085	0.0839
3	2	500	20	adam	10000	0.0	0.000	0.8929	0.0733	0.0094	0.0889
3	2	500	20	adam	100000	0.0	0.000	0.9101	0.0663	0.0080	0.0802
3	2	500	20	lbfgs	100	0.0	0.000	0.9263	0.0616	0.0062	0.0774
3	2	500	20	lbfgs	1000	0.0	0.000	0.9299	0.0589	0.0059	0.0757
3	2	500	20	lbfgs	10000	0.0	0.000	0.9246	0.0612	0.0064	0.0782
3	2	500	20	lbfgs	100000	0.0	0.001	0.9298	0.0596	0.0059	0.0758
3	2	500	20	sgd	100	0.0	0.000	0.4830	0.1808	0.0419	0.2048
3	2	500	20	sgd	1000	0.0	0.000	0.3451	0.1957	0.0541	0.2319
3	2	500	20	sgd	10000	0.0	0.000	0.4443	0.1841	0.0453	0.2128
3	2	500	20	sgd	100000	0.0	0.000	0.0659	0.2340	0.0749	0.2731
3	2	500	30	adam	100	0.0	0.000	0.9267	0.0433	0.0042	0.0581
3	2	500	30	adam	1000	0.0	0.000	0.9239	0.0489	0.0044	0.0627
3	2	500	30	adam	10000	0.0	0.000	0.9120	0.0588	0.0054	0.0731
3	2	500	30	adam	100000	0.0	0.000	0.9234	0.0457	0.0044	0.0598
3	2	500	30	lbfgs	100	0.0	0.000	0.9787	0.0277	0.0013	0.0352
3	2	500	30	lbfgs	1000	0.0	0.000	0.9808	0.0270	0.0011	0.0333
3	2	500	30	lbfgs	10000	0.0	0.000	0.9800	0.0268	0.0012	0.0336
3	2	500	30	lbfgs	100000	0.0	0.000	0.9812	0.0262	0.0011	0.0328
3	2	500	30	sgd	100	0.0	0.001	0.2339	0.1775	0.0465	0.2156
3	2	500	30	sgd	1000	0.0	0.000	0.5421	0.1415	0.0277	0.1666
3	2	500	30	sgd	10000	0.0	0.000	0.0124	0.2058	0.0613	0.2431
3	2	500	30	sgd	100000	0.0	0.000	-0.0061	0.2056	0.0608	0.2467
3	2	500	40	adam	100	0.0	0.000	0.8795	0.0668	0.0076	0.0837
3	2	500	40	adam	1000	0.0	0.000	0.9379	0.0484	0.0039	0.0611
3	2	500	40	adam	10000	0.0	0.000	0.9145	0.0566	0.0054	0.0694
3	2	500	40	adam	100000	0.0	0.000	0.8837	0.0672	0.0071	0.0843
3	2	500	40	lbfgs	100	0.0	0.001	0.9527	0.0446	0.0028	0.0514
3	2	500	40	lbfgs	1000	0.0	0.000	0.9476	0.0464	0.0031	0.0541
3	2	500	40	lbfgs	10000	0.0	0.000	0.9510	0.0453	0.0029	0.0520
3	2	500	40	lbfgs	100000	0.0	0.000	0.9525	0.0446	0.0028	0.0511
3	2	500	40	sgd	100	0.0	0.000	0.1657	0.1781	0.0519	0.2234
3	2	500	40	sgd	1000	0.0	0.000	0.3988	0.1543	0.0366	0.1914
3	2	500	40	sgd	10000	0.0	0.001	0.3348	0.1640	0.0411	0.2012
3	2	500	40	sgd	100000	0.0	0.000	0.4545	0.1468	0.0336	0.1824

Figure C4

Classical Neural Networks Performance Metrics - 500 Samples and 80/20 Split

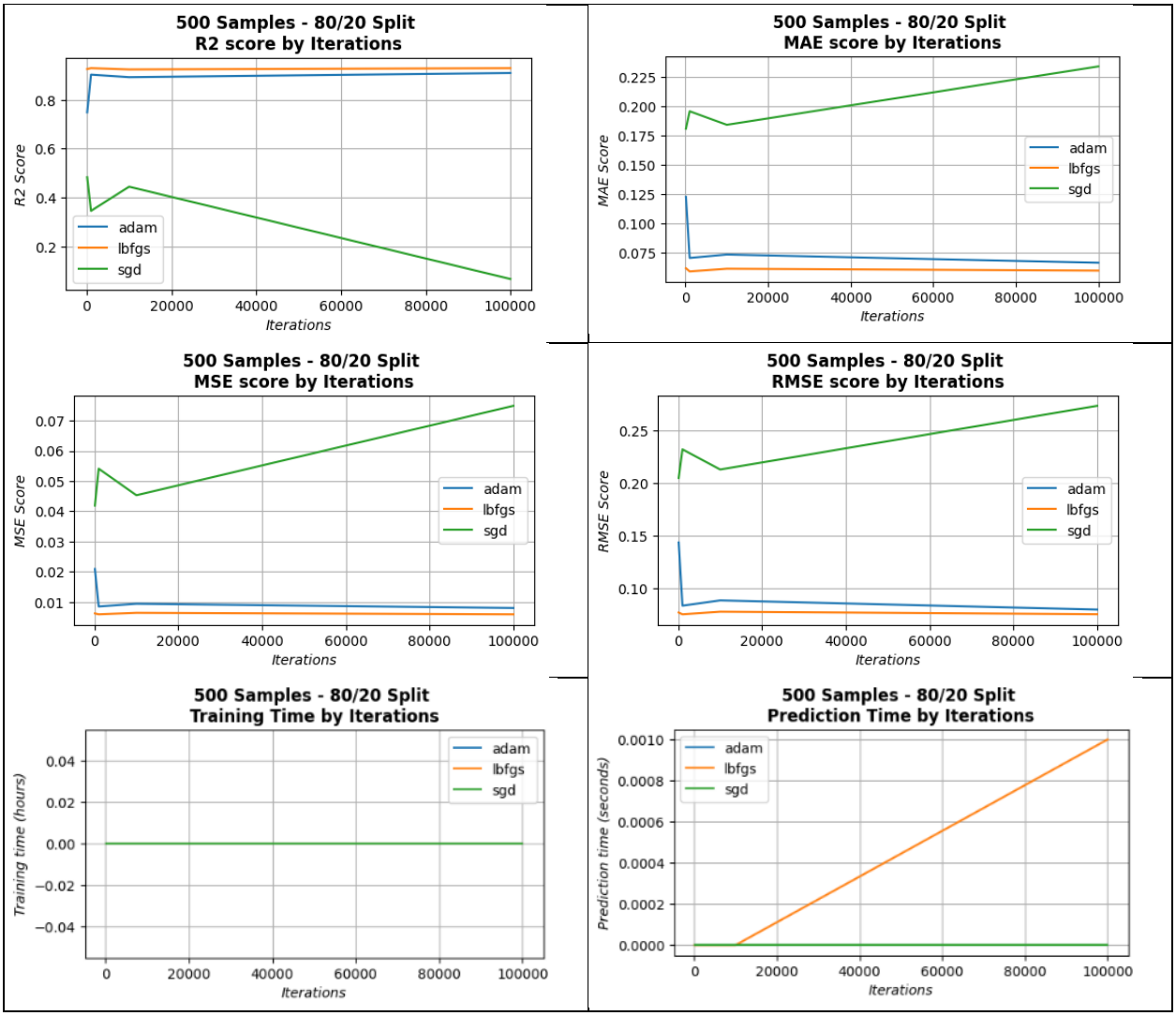


Figure C5

Classical Neural Networks Performance Metrics - 500 Samples and 70/30 Split

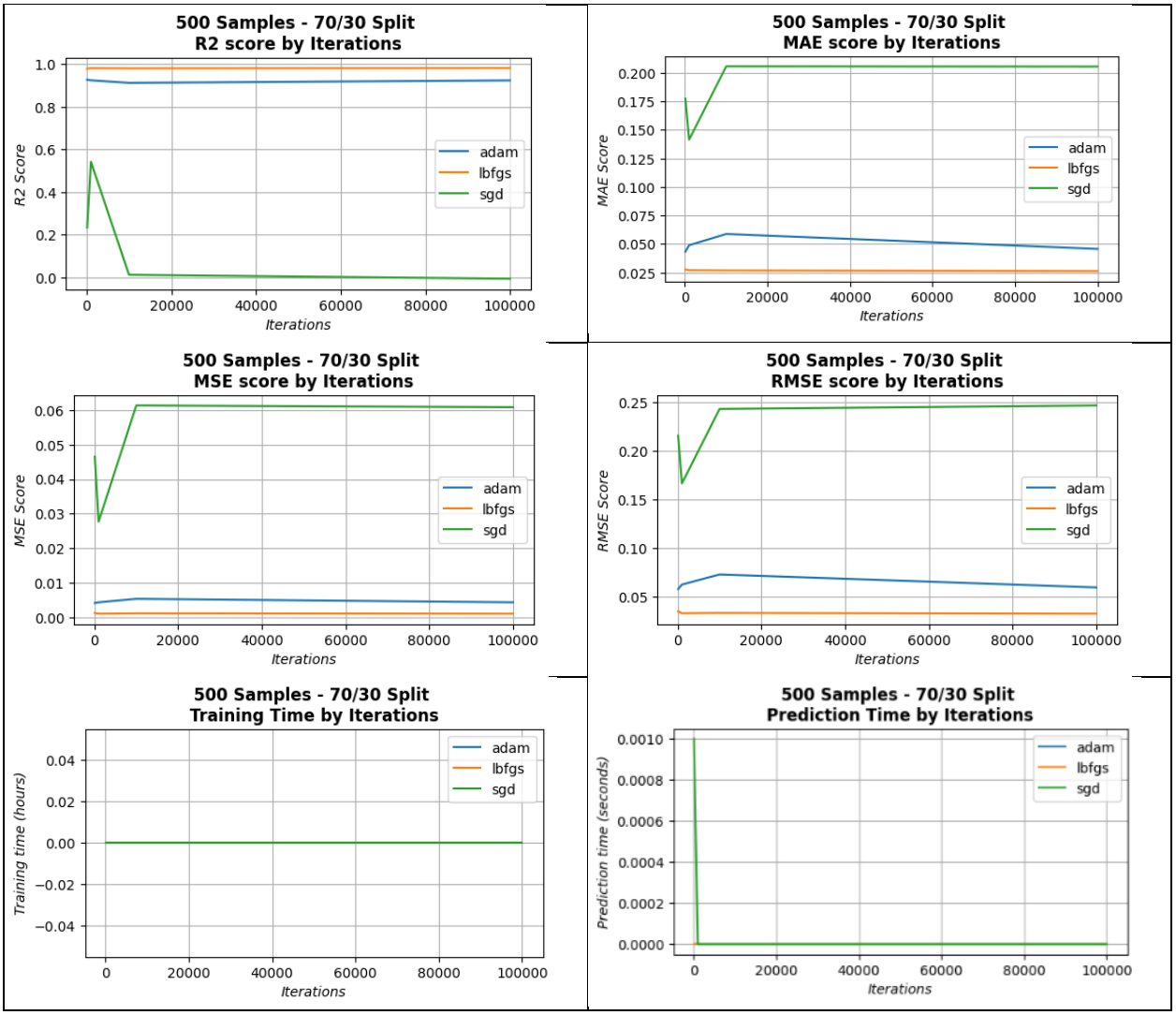


Figure C6

Classical Neural Networks Performance Metrics - 500 Samples and 60/40 Split

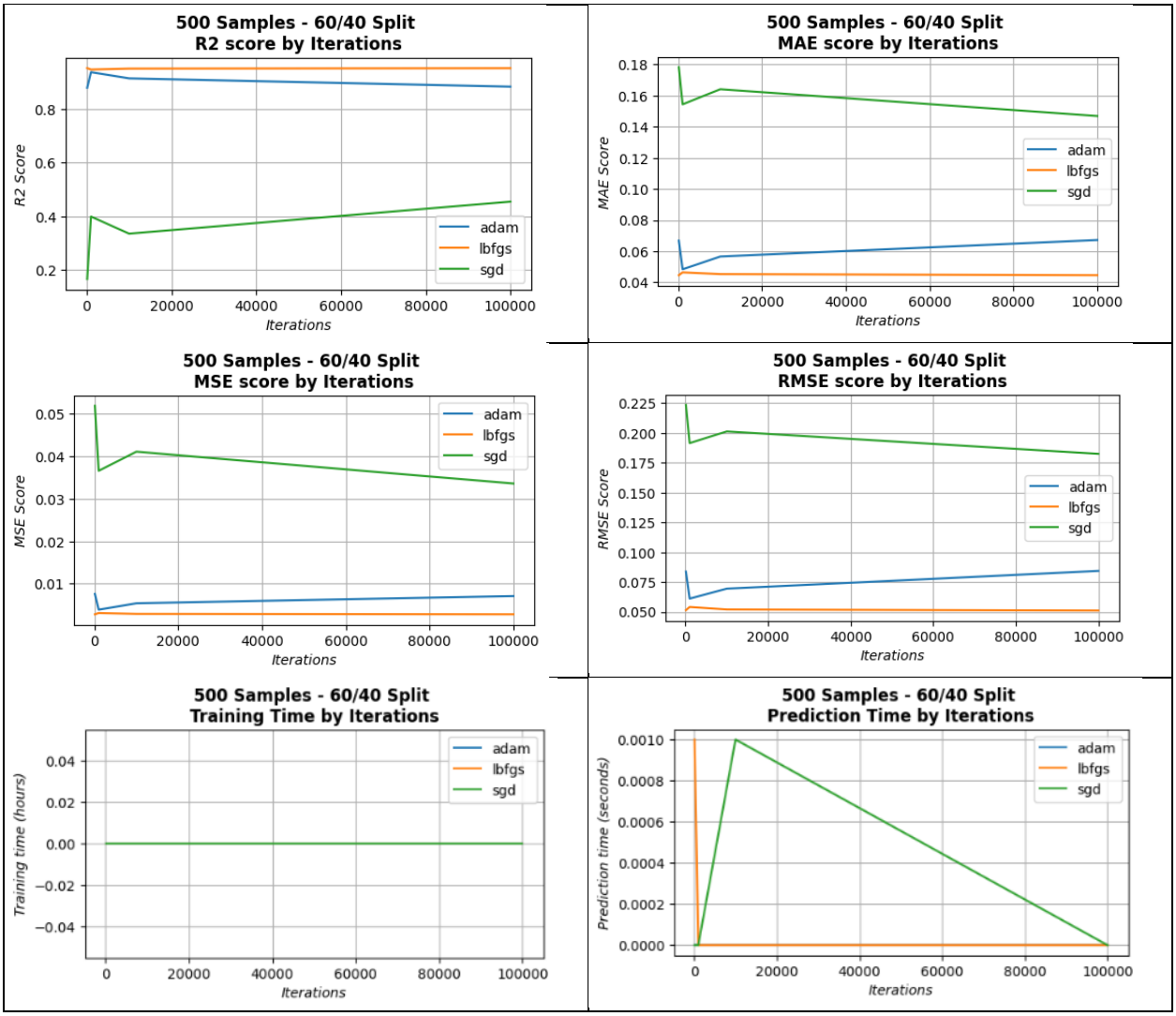


Table C3

Classical Neural Networks Raw Training Results Using 1000 Samples

n_features	n_targets	n_samples	test_size	optimizer	iterations	training_time	prediction_time	r2	mae	mse	rmse
3	2	1000	20	adam	100	0.0	0.000	0.8433	0.0878	0.0109	0.1043
3	2	1000	20	adam	1000	0.0	0.000	0.8414	0.0895	0.0112	0.1053
3	2	1000	20	adam	10000	0.0	0.000	0.8748	0.0817	0.0085	0.0923
3	2	1000	20	adam	100000	0.0	0.000	0.8704	0.0842	0.0090	0.0946
3	2	1000	20	lbfgs	100	0.0	0.000	0.8852	0.0793	0.0078	0.0885
3	2	1000	20	lbfgs	1000	0.0	0.001	0.8874	0.0768	0.0077	0.0880
3	2	1000	20	lbfgs	10000	0.0	0.000	0.8862	0.0771	0.0078	0.0884
3	2	1000	20	lbfgs	100000	0.0	0.000	0.8807	0.0782	0.0083	0.0909
3	2	1000	20	sgd	100	0.0	0.000	0.2879	0.1826	0.0501	0.2232
3	2	1000	20	sgd	1000	0.0	0.000	0.1844	0.1968	0.0571	0.2385
3	2	1000	20	sgd	10000	0.0	0.000	0.4742	0.1623	0.0372	0.1919
3	2	1000	20	sgd	100000	0.0	0.000	0.4844	0.1585	0.0362	0.1898
3	2	1000	30	adam	100	0.0	0.000	0.9215	0.0559	0.0051	0.0710
3	2	1000	30	adam	1000	0.0	0.001	0.9278	0.0533	0.0047	0.0676
3	2	1000	30	adam	10000	0.0	0.000	0.8603	0.0738	0.0092	0.0952
3	2	1000	30	adam	100000	0.0	0.000	0.8880	0.0661	0.0074	0.0857
3	2	1000	30	lbfgs	100	0.0	0.001	0.9603	0.0413	0.0026	0.0504
3	2	1000	30	lbfgs	1000	0.0	0.000	0.9595	0.0416	0.0027	0.0506
3	2	1000	30	lbfgs	10000	0.0	0.000	0.9638	0.0397	0.0024	0.0479
3	2	1000	30	lbfgs	100000	0.0	0.000	0.9622	0.0404	0.0025	0.0490
3	2	1000	30	sgd	100	0.0	0.000	0.4886	0.1454	0.0336	0.1832
3	2	1000	30	sgd	1000	0.0	0.000	0.2963	0.1717	0.0465	0.2140
3	2	1000	30	sgd	10000	0.0	0.000	0.1401	0.1861	0.0570	0.2319
3	2	1000	30	sgd	100000	0.0	0.000	0.3887	0.1593	0.0402	0.2001
3	2	1000	40	adam	100	0.0	0.000	0.9348	0.0521	0.0041	0.0638
3	2	1000	40	adam	1000	0.0	0.001	0.9174	0.0596	0.0052	0.0719
3	2	1000	40	adam	10000	0.0	0.000	0.9221	0.0578	0.0049	0.0702
3	2	1000	40	adam	100000	0.0	0.000	0.9349	0.0534	0.0041	0.0641
3	2	1000	40	lbfgs	100	0.0	0.000	0.9446	0.0495	0.0035	0.0590
3	2	1000	40	lbfgs	1000	0.0	0.000	0.9457	0.0485	0.0034	0.0583
3	2	1000	40	lbfgs	10000	0.0	0.000	0.9449	0.0489	0.0035	0.0589
3	2	1000	40	lbfgs	100000	0.0	0.000	0.9463	0.0484	0.0034	0.0581
3	2	1000	40	sgd	100	0.0	0.000	0.1834	0.1838	0.0516	0.2272
3	2	1000	40	sgd	1000	0.0	0.000	0.3225	0.1665	0.0429	0.2069
3	2	1000	40	sgd	10000	0.0	0.001	0.3869	0.1575	0.0388	0.1968
3	2	1000	40	sgd	100000	0.0	0.001	0.3557	0.1641	0.0408	0.2018

Figure C7

Classical Neural Networks Performance Metrics - 1000 Samples and 80/20 Split

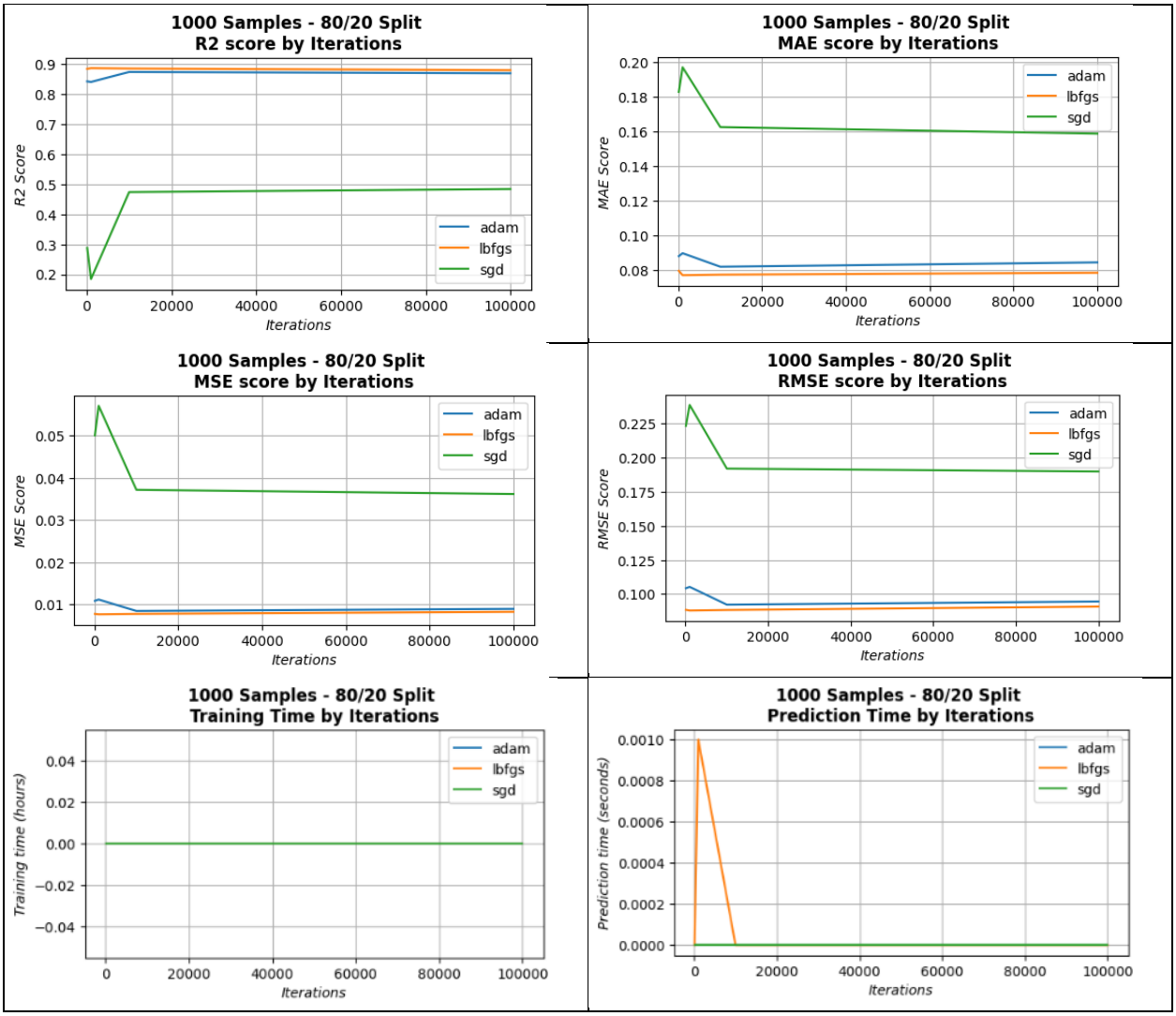


Figure C8

Classical Neural Networks Performance Metrics - 1000 Samples and 70/30 Split

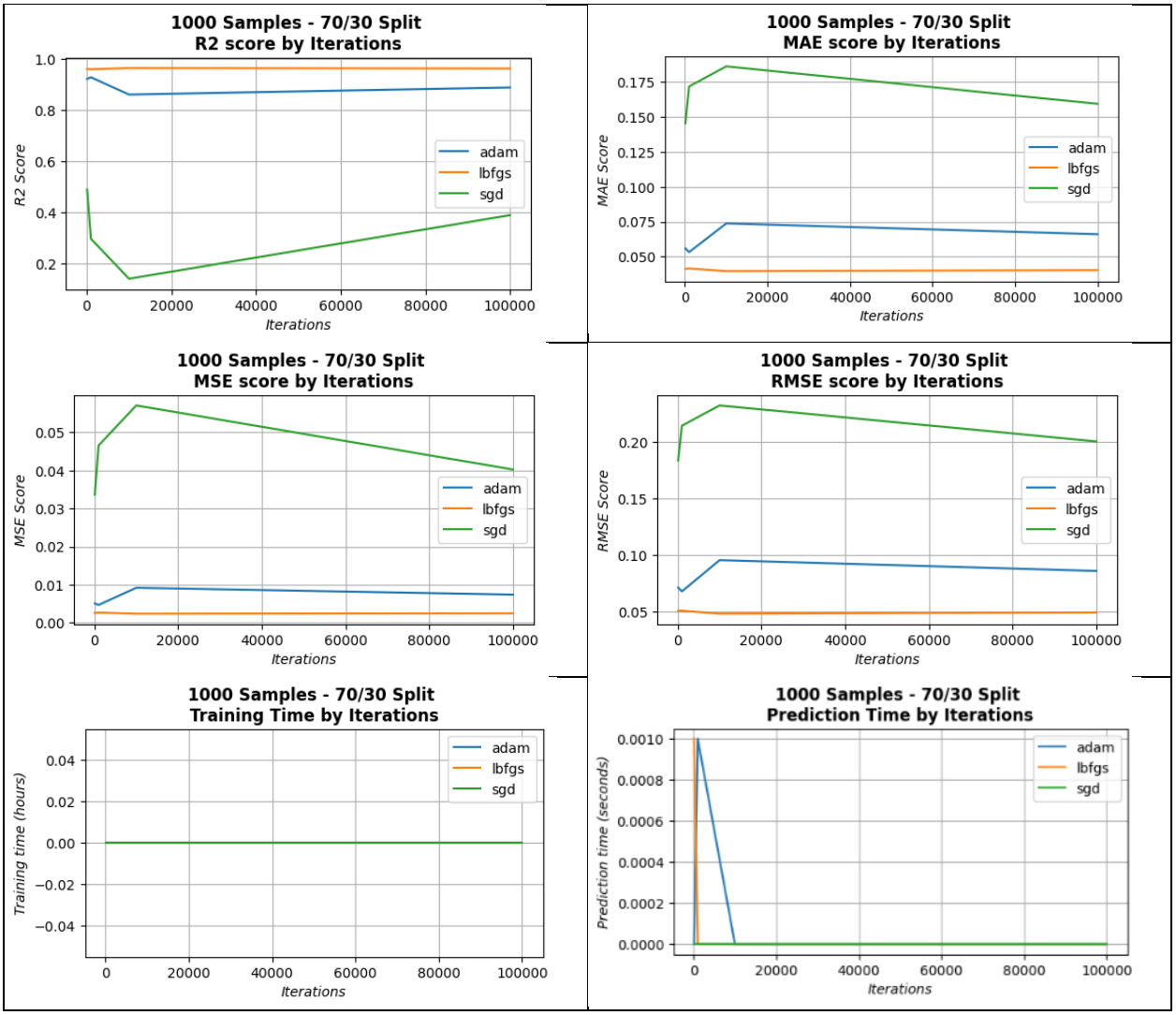
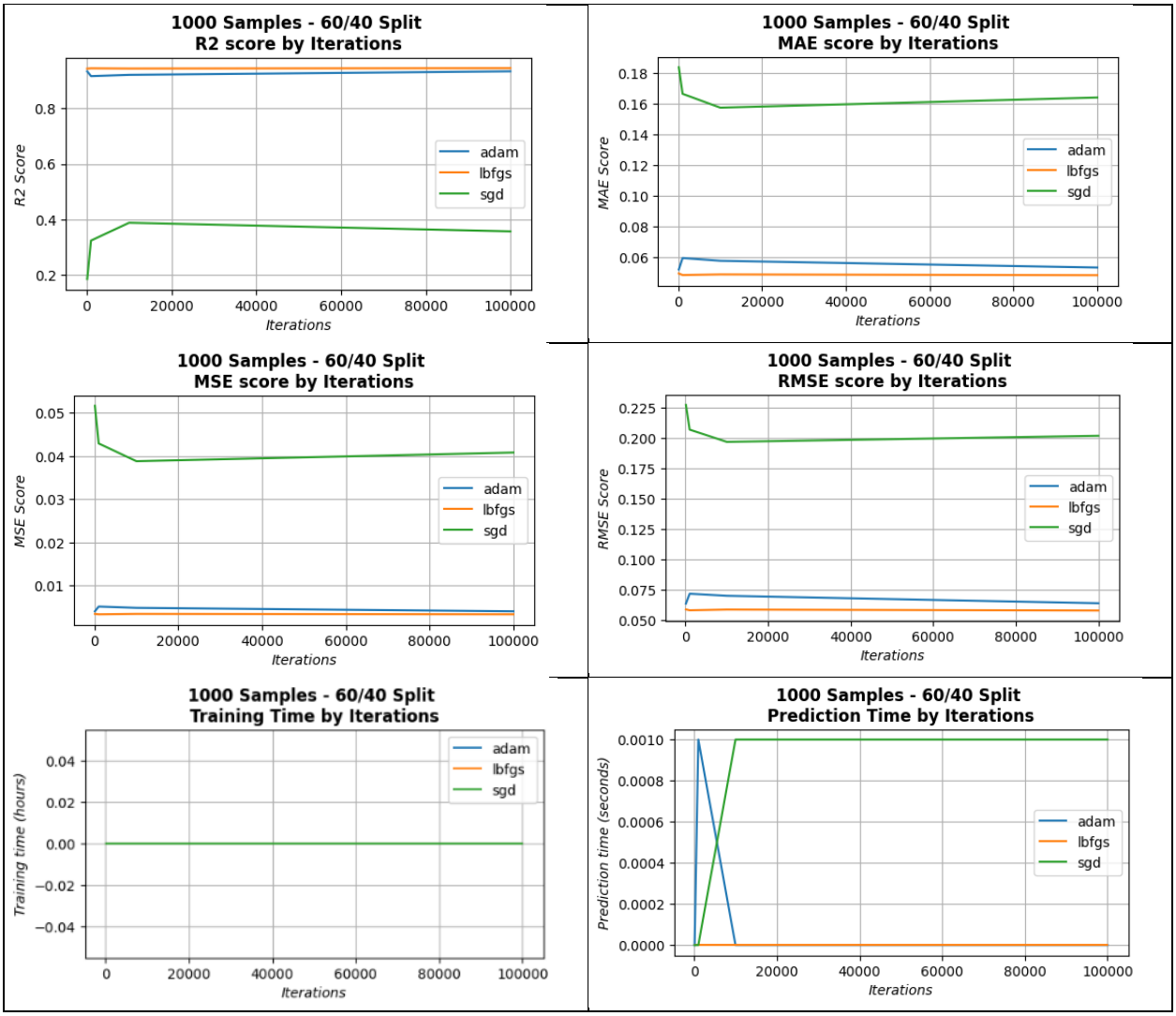


Figure C9

Classical Neural Networks Performance Metrics - 1000 Samples and 60/40 Split



Appendix D

Multitarget Quantum Neural Network Regressor Python Module

```
import numpy as np
import pickle
from qiskit import QuantumCircuit
from qiskit.algorithms.gradients import *
from qiskit.algorithms.optimizers import *
from qiskit.circuit import Parameter
from qiskit.circuit.library import RealAmplitudes
from qiskit_machine_learning.algorithms.regressors import NeuralNetworkRegressor
from qiskit_machine_learning.neural_networks import EstimatorQNN
from qiskit.primitives import Estimator as LocalEstimator
from qiskit_ibm_runtime import QiskitRuntimeService, Estimator, Session, Options
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.preprocessing import MinMaxScaler
from typing import List, Literal


class MultiTargetQuantumNNRegressor:
    r"""MultiTargetQuantumNNRegressor

    This class creates a multivariate multiple regression model by instantiating a separate quantum
    neural network for each target output. This class automatically constructs the necessary
    quantum circuits for the feature map and an ansatz for the trainable weights to encode
    classical data into a quantum state.
    """
    def __init__(
            self,
            n_features: int,
            n_targets: int,
            optimizer: Literal['cobyla', 'adam', 'lbfgsb', 'slsqp'],
            maxiter: int = 50,
            use_quantum_cloud: bool = False,
            backend: str = None
            ) -> None:
        r"""
        Parameters
        ----------
        n_features: int
            The number of features in the regression model.

        n_targets: int
            The number of regression targets i.e. the dimension of the y output vector.

        optimizer: {'cobyla', 'adam', 'lbfgsb','slsqp'}, default = 'slsqp'
            The solver for weight optimization.

        maxiter: int, default = 50
            Maximum number of iterations. The solver iterates until convergence.

        use_quantum_cloud: bool
            Determines whether to run on CPU or IBM Quantum Cloud.

        backend: str, default = None
            If running on quantum cloud, name of quantum hardware or simulator instance.
        """
```

```
self._n_features = n_features
        self._n_targets = n_targets

        self._use_quantum_cloud = use_quantum_cloud
        self._backend = backend
        if self._use_quantum_cloud:
            if self._backend is None:
                self._backend = 'ibmq_qasm_simulator'
        if optimizer is None or optimizer == 'slsqp':
            self._optimizer = SLSQP(maxiter=maxiter)
        elif optimizer == 'cobyla':
            self._optimizer = COBYLA(maxiter=maxiter)
        elif optimizer == 'adam':
            self._optimizer = ADAM(maxiter=maxiter)
        elif optimizer == 'lbfgsb':
            self._optimizer = L_BFGS_B(maxiter=maxiter)
        else:
            raise Exception(f'The selected optimizer is invalid: {optimizer}')
        self._neural_networks: List[NeuralNetworkRegressor] = []
        for _ in range(n_targets):
            feature_map = self._create_feature_map()
            ansatz = self._create_ansatz()
            qc = self._create_quantum_circuit(feature_map=feature_map, ansatz=ansatz)
            estimator_qnn = self._create_estimator_qnn(qc=qc, feature_map=feature_map, ansatz=ansatz)
            regressor = self._create_neural_network_regressor(qnn=estimator_qnn)
            self._neural_networks.append(regressor)

    def _create_feature_map(self):
        """
        Creates a parameterized quantum circuit based on the number of features in the regression model.
        """
        feature_map = QuantumCircuit(self._n_features)
        parameters = []
        for i in range(self._n_features):
            parameter = Parameter(f"x{i}")
            parameters.append(parameter)
            feature_map.ry(parameter, i)
        return feature_map

    def _create_ansatz(self):
        """
        Creates an ansatz for the trainable weights.
        """
        ansatz = RealAmplitudes(self._n_features, reps=self._n_features)
        return ansatz
```

```
    def _create_quantum_circuit(self, feature_map: QuantumCircuit, ansatz: RealAmplitudes):
        """
        Builds the quantum circuit that will be used to encode the classical data.
        """
        qc = QuantumCircuit(self._n_features)
        qc.compose(feature_map, inplace=True)
        qc.compose(ansatz, inplace=True)
        return qc

    def _get_estimator(self):
        """
        Creates an Estimator primitive for use on a classical machine or IBM Quantum Cloud.
        """
        if self._use_quantum_cloud:
            service = QiskitRuntimeService(channel="ibm_quantum")
            backend = service.backend(self._backend)
            session = Session(service=service, backend=backend,max_time=18000)
            options = Options(max_execution_time=18000)
            options.execution.shots = 1024
            estimator = Estimator(session=session, options=options)
            return estimator
        else:
            return LocalEstimator()

    def _create_estimator_qnn(self, qc:QuantumCircuit, feature_map: QuantumCircuit, ansatz:
RealAmplitudes):
        """
        Creates the Estimator Quantum Neural Network using the constructed quantum circuit and ansatz.
        """
        estimator = self._get_estimator()
        qnn = EstimatorQNN(
            circuit=qc,
            estimator=estimator,
            input_params=feature_map.parameters,
            weight_params=ansatz.parameters
        )
        return qnn

    def _create_neural_network_regressor(self, qnn: EstimatorQNN):
        """
        Creates the trainable regression model using an underlying Estimator Quantum Neural Network.
        """
        regressor = NeuralNetworkRegressor(
            neural_network=qnn,
            loss="squared_error",
            optimizer=self._optimizer
        )
        return regressor
```

```
    def fit(self, X: np.ndarray, y: np.ndarray, scale_data: bool = False):
        """
        Begins training the neural network corresponding to each output variable.

        Parameters
        ----------
        X: numpy array of shape (n_samples, n_features)
            The input data.

        y: numpy array of shape (n_samples, n_targets)
            The target values.

        scale_data: bool, default = False
            Set this value to true if the dataset is not already scaled down to values between 0 and 1.
            Note: Model training convergence is better when the transformation is applied.
        """
        if X.shape[1] != self._n_features:
            raise ValueError(f"Shapes don't match, X features: {X.shape[1]}, n_features:
{self._n_features}!")

        if y.shape[1] != self._n_targets:
            raise ValueError(f"Shapes don't match, y targets: {y.shape[1]}, n_targets:
{self._n_targets}!")

        if X.shape[0] != y.shape[0]:
            raise ValueError(f"Shapes don't match, X samples: {X.shape[0]}, y samples: {y.shape[0]}!")

        if scale_data:
            X, y = self.get_scaled_data(X, y)

        self._fit(X,y)

    def _fit(self, X: np.ndarray, y: np.ndarray):
        """
        Internally called from self.fit() to begin training each model.
        """
        for i in range(self._n_targets):
            self._neural_networks[i].fit(X, y[:, i])

    @staticmethod
    def get_scaled_data(X: np.ndarray, y: np.ndarray):
        """
        Scales the dataset down to values between 0 and 1.
        Note: Model training convergence is better when the transformation is applied.

        Parameters
        ----------
        X: numpy array of shape (n_samples, n_features)
            The input data.

        y: numpy array of shape (n_samples, n_targets)
            The target values.
        """
        scaler = MinMaxScaler()
        X = scaler.fit_transform(X)
        if y is not None:
            y = scaler.fit_transform(y)
        return X, y
```

```
    @staticmethod
    def get_unscaled_data(X: np.ndarray, y: np.ndarray):
        """
        Reverses the scaling of the X and y values to original values.

        Parameters
        ----------
        X: numpy array of shape (n_samples, n_features)
            The input data.

        y: numpy array of shape (n_samples, n_targets)
            The target values.
        """
        scaler = MinMaxScaler()
        X = scaler.inverse_transform(X)
        if y is not None:
            y = scaler.inverse_transform(y)
        return X, y

    def predict(self, X: np.ndarray, scale_data: bool = False):
        """
        Predict the target values given the input vector.

        Parameters
        ----------
        X: numpy array of shape (n_samples, n_features)
            The input data.

        Returns
        ----------
        y: numpy array of shape (n_samples, n_targets)
            The target values.
        """
        if scale_data:
            X, _ = self.get_scaled_data(X, None)
        if X.shape[1] != self._n_features:
            raise ValueError(f"Shapes don't match, X features: {X.shape[1]}, n_features:
{self._n_features}!")
        return self._predict(X)

    def _predict(self, X: np.ndarray):
        """
        Internally called from self.predict().
        """
        predictions = []
        for i in range(self._n_targets):
            prediction = self._neural_networks[i].predict(X)
            predictions.append(prediction)
        final_result = np.reshape(np.stack((predictions), axis= 1), (X.shape[0], len(predictions)))
        return final_result
```

```
    def score(self, X: np.ndarray, y: np.ndarray, scale_data: bool = False):
        """
        Compute the coefficient of determination i.e. the R-squared (R2) score.

        Parameters
        ----------
        X: numpy array of shape (n_samples, n_features)
            The input test data.

        y: numpy array of shape (n_samples, n_targets)
            The true target values.

        Returns
        ----------
        r2: float
            The r-squared (R2) score of the predictions.
        """
        if X.shape[1] != self._n_features:
            raise ValueError(f"Shapes don't match, X features: {X.shape[1]}, n_features:
{self._n_features}!")

        if y.shape[1] != self._n_targets:
            raise ValueError(f"Shapes don't match, y targets: {y.shape[1]}, n_targets:
{self._n_targets}!")

        if X.shape[0] != y.shape[0]:
            raise ValueError(f"Shapes don't match, X samples: {X.shape[0]}, y samples: {y.shape[0]}!")
        if scale_data:
            X, y = self.get_scaled_data(X, y)

        return self._score(X, y)

    def _score(self, X: np.ndarray, y: np.ndarray):
        """
        Internally called from self.score().
        """
        scores = []

        for i in range(self._n_targets):
            performance_score = self._neural_networks[i].score(X, y[:,i])
            scores.append(performance_score)

        return np.mean(scores)
```

```
def mae(self, y_true, y_pred):
    """
    Computes the Mean Absolute Error.

    Parameters
    ----------
    y_true: numpy array of shape (n_samples, n_targets)
        The true target values.

    y_pred: numpy array of shape (n_samples, n_targets)
        The predicted target values.

    Returns
    ----------
    mae: float
        The mean absolute error regression loss.
    """
    return mean_absolute_error(y_true=y_true, y_pred=y_pred)

def mse(self, y_true, y_pred):
    """
    Computes the Mean Squared Error.

    Parameters
    ----------
    y_true: numpy array of shape (n_samples, n_targets)
        The true target values.

    y_pred: numpy array of shape (n_samples, n_targets)
        The predicted target values.

    Returns
    ----------
    mae: float
        The mean squared error regression loss.
    """
    return mean_squared_error(y_true=y_true, y_pred=y_pred, squared=True)

def rmse(self, y_true, y_pred):
    """
    Computes the Root Mean Squared Error.

    Parameters
    ----------
    y_true: numpy array of shape (n_samples, n_targets)
        The true target values.

    y_pred: numpy array of shape (n_samples, n_targets)
        The predicted target values.

    Returns
    ----------
    mae: float
        The root mean squared error regression loss.
    """
    return mean_squared_error(y_true=y_true, y_pred=y_pred, squared=False)
```

```
    def save_model(self, model_name: str = 'model'):
        """
        Saves the model to disk.

        Parameters
        ----------
        model_name: str, default = 'model'

        """
        with open(f'{model_name}', 'wb') as file:
            pickle.dump(self, file)

    @staticmethod
    def load_model(model_name: str = 'model'):
        """
        Loads the saved model from disk.

        Parameters
        ----------
        model_name: str, default = 'model'

        Returns
        ----------
        model: MultiTargetQuantumNNRegressor
            The saved instance of the MultiTargetQuantumNNRegressor.
        """
        with open(model_name, 'rb') as file:
            model = pickle.load(file)
            class_type = MultiTargetQuantumNNRegressor
            if isinstance(model, class_type):
                return model
            else:
                raise Exception(f"The model loaded is not an instance of the {class_type.__name__}
class.")
```

Appendix E

Quantum Neural Network Results

Table E1

Quantum Neural Networks Raw Training Results Using 100 Samples

n_features	n_targets	n_samples	test_size	optimizer	iterations	training_time	prediction_time	r2	mae	mse	rmse
3	2	100	40	cobyla	100	0.0183	0.4122	0.5526	0.0990	0.0170	0.1306
3	2	100	30	cobyla	100	0.0211	0.3130	0.6388	0.0969	0.0152	0.1234
3	2	100	20	cobyla	100	0.0241	0.1939	0.2583	0.1784	0.0500	0.2237
3	2	100	40	cobyla	300	0.0545	0.4193	0.8490	0.0585	0.0057	0.0753
3	2	100	30	cobyla	300	0.0635	0.3177	0.7338	0.0796	0.0110	0.1047
3	2	100	20	cobyla	300	0.0720	0.1949	0.2633	0.1809	0.0497	0.2229
3	2	100	40	cobyla	500	0.0901	0.4129	0.8440	0.0600	0.0060	0.0774
3	2	100	30	cobyla	500	0.1049	0.4038	0.7481	0.0784	0.0107	0.1036
3	2	100	20	cobyla	500	0.1187	0.2063	0.6984	0.1169	0.0196	0.1400
3	2	100	40	lbfgsb	100	0.3330	0.4075	0.9228	0.0418	0.0029	0.0539
3	2	100	30	lbfgsb	100	0.3887	0.3057	0.9070	0.0468	0.0040	0.0633
3	2	100	20	lbfgsb	100	0.4374	0.2052	0.7495	0.1067	0.0158	0.1257
3	2	100	40	adam	100	0.2793	0.4132	-1.6090	0.2283	0.0973	0.3119
3	2	100	30	adam	100	0.3176	0.3056	-12.2426	0.6731	0.5797	0.7614
3	2	100	20	adam	100	0.3649	0.2065	-4.3832	0.4742	0.3792	0.6158
3	2	100	40	lbfgsb	300	0.5409	0.3906	0.9249	0.0414	0.0028	0.0532
3	2	100	30	lbfgsb	300	0.7260	0.3345	0.9004	0.0493	0.0043	0.0652
3	2	100	20	lbfgsb	300	0.7275	0.2185	0.7563	0.1055	0.0154	0.1240
3	2	100	40	adam	300	0.8133	0.3890	-1.8461	0.2529	0.1110	0.3331
3	2	100	40	lbfgsb	500	0.4880	0.4122	0.9161	0.0442	0.0032	0.0564
3	2	100	30	adam	300	0.9526	0.4027	-1.5555	0.2397	0.1022	0.3197
3	2	100	40	slsqp	100	0.1640	0.4337	0.8986	0.0483	0.0039	0.0623
3	2	100	40	slsqp	300	0.1359	0.3933	0.8876	0.0501	0.0043	0.0657
3	2	100	20	adam	300	1.0901	0.2487	-0.1966	0.2149	0.0829	0.2880
3	2	100	40	slsqp	500	0.1134	0.4366	0.9026	0.0476	0.0037	0.0606
3	2	100	30	lbfgsb	500	0.6779	0.3053	0.9085	0.0466	0.0039	0.0627
3	2	100	30	slsqp	100	0.1139	0.3362	0.8787	0.0501	0.0052	0.0724
3	2	100	20	lbfgsb	500	0.9039	0.2090	0.7573	0.1055	0.0153	0.1237
3	2	100	30	slsqp	300	0.1685	0.3156	0.8949	0.0496	0.0045	0.0672
3	2	100	20	slsqp	100	0.1563	0.2099	0.7432	0.1065	0.0162	0.1271
3	2	100	30	slsqp	500	0.1601	0.3152	0.8793	0.0536	0.0051	0.0712
3	2	100	20	slsqp	300	0.2665	0.2202	0.7411	0.1098	0.0165	0.1283
3	2	100	40	adam	500	1.3768	0.5064	-0.4170	0.1608	0.0519	0.2279
3	2	100	20	slsqp	500	0.1728	0.2030	0.7698	0.1003	0.0146	0.1208
3	2	100	30	adam	500	1.6028	0.3073	-0.3931	0.1745	0.0568	0.2383
3	2	100	20	adam	500	1.8341	0.2299	0.0075	0.2152	0.0641	0.2532

Figure E1

Quantum Neural Networks Performance Metrics - 100 Samples and 80/20 Split

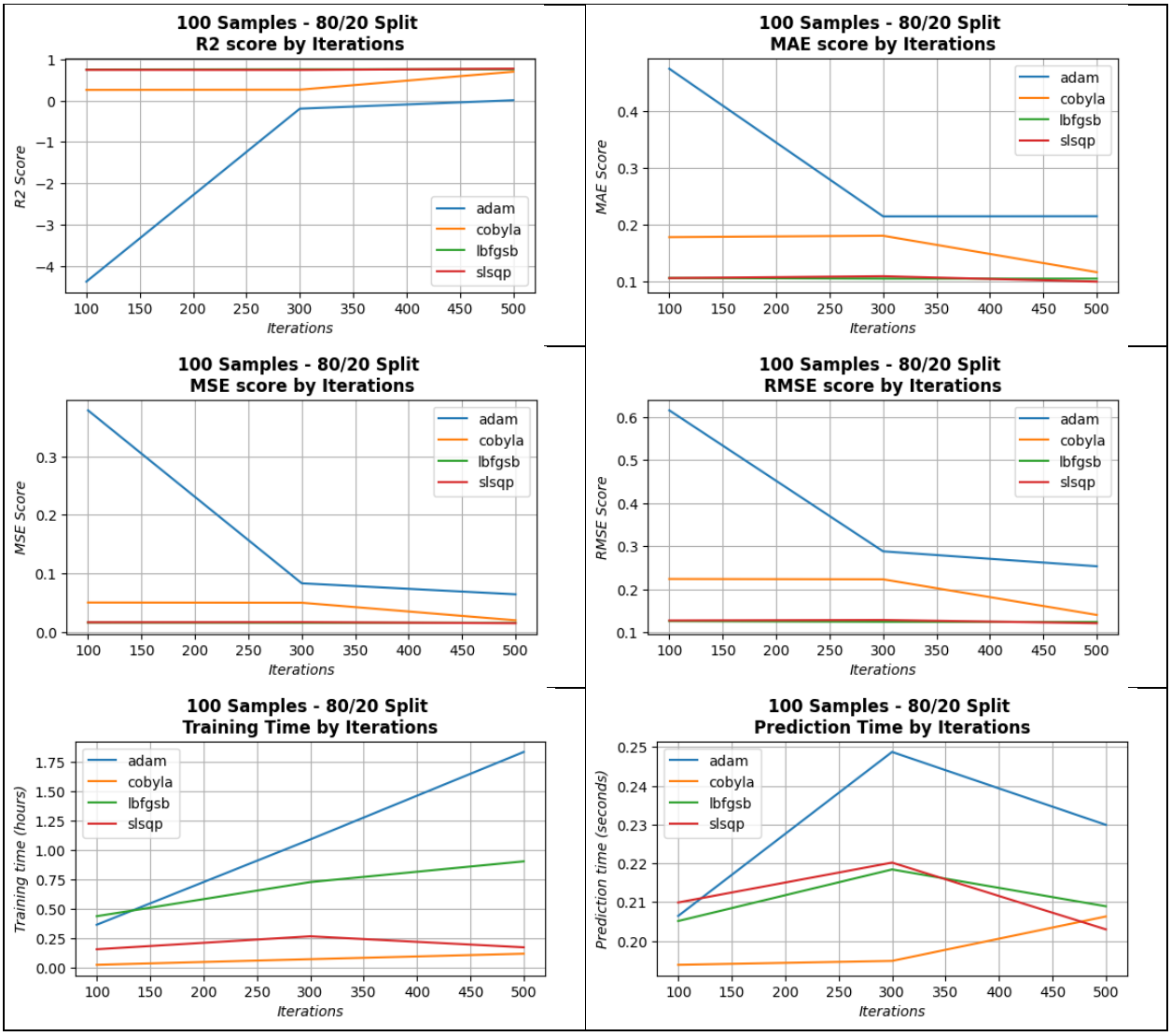


Figure E2

Quantum Neural Networks Performance Metrics - 100 Samples and 70/30 Split

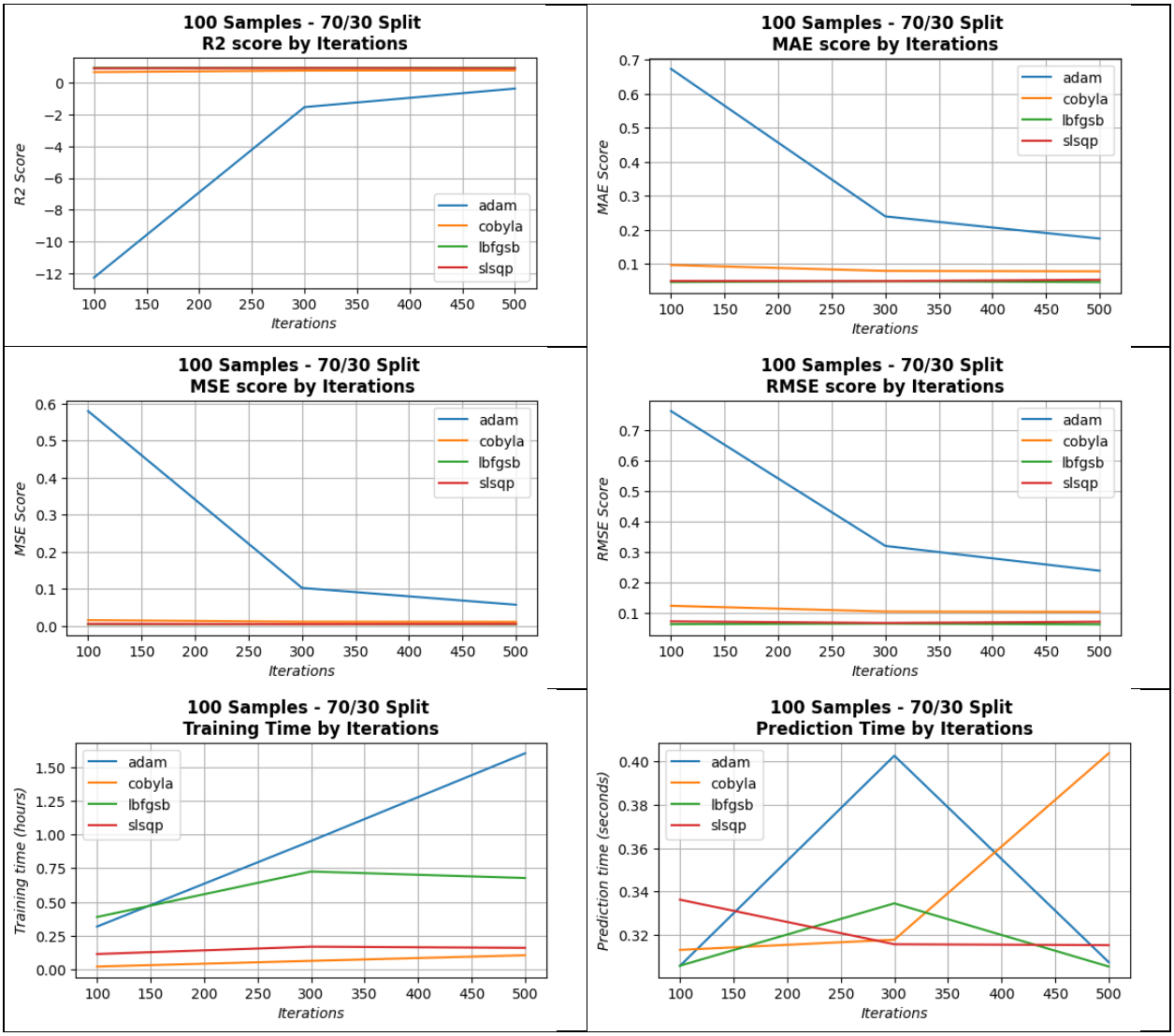


Figure E3

Quantum Neural Networks Performance Metrics - 100 Samples and 60/40 Split

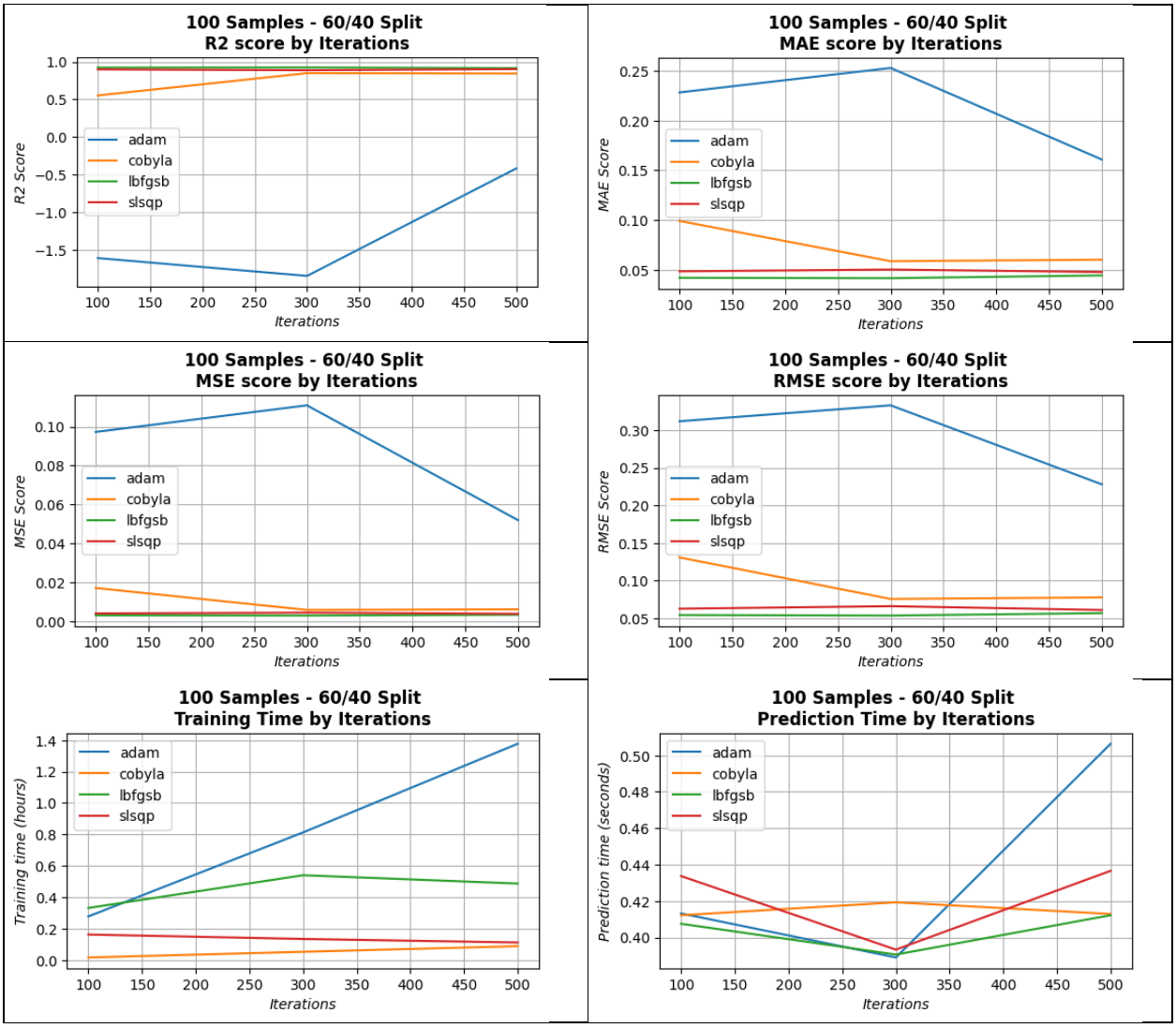


Table E2

Quantum Neural Networks Raw Training Results Using 500 Samples

n_features	n_targets	n_samples	test_size	optimizer	iterations	training_time	prediction_time	r2	mae	mse	rmse
3	2	500	40	cobyla	100	0.1390	0.3902	0.7881	0.0917	0.0131	0.1144
3	2	500	30	cobyla	100	0.1417	0.4156	0.8106	0.0835	0.0113	0.1065
3	2	500	20	cobyla	100	0.1452	0.2265	0.5269	0.1691	0.0400	0.2000
3	2	500	40	cobyla	300	0.4179	0.4090	0.9070	0.0572	0.0057	0.0756
3	2	500	30	cobyla	300	0.4249	0.3126	0.8524	0.0744	0.0088	0.0936
3	2	500	20	cobyla	300	0.4323	0.2839	0.8871	0.0725	0.0100	0.1001
3	2	500	40	cobyla	500	0.6882	0.5211	0.9068	0.0561	0.0058	0.0764
3	2	500	30	cobyla	500	0.7005	0.3038	0.9155	0.0553	0.0050	0.0705
3	2	500	20	cobyla	500	0.7145	0.3106	0.8219	0.0937	0.0156	0.1249
3	2	500	40	lbfgsb	100	2.0474	0.4342	0.9369	0.0496	0.0039	0.0624
3	2	500	30	lbfgsb	100	2.4656	0.3058	0.9461	0.0398	0.0031	0.0561
3	2	500	20	lbfgsb	100	2.5417	0.2049	0.9083	0.0652	0.0081	0.0900
3	2	500	40	adam	100	2.0968	0.4367	-3.0032	0.4448	0.2474	0.4974
3	2	500	30	adam	100	2.1331	0.3154	-2.8674	0.3831	0.2263	0.4757
3	2	500	20	adam	100	2.1804	0.2051	-1.7000	0.4047	0.2227	0.4719
3	2	500	30	lbfgsb	300	3.4232	0.3060	0.9501	0.0378	0.0029	0.0540
3	2	500	20	lbfgsb	300	4.0961	0.2104	0.9116	0.0646	0.0078	0.0883
3	2	500	40	lbfgsb	300	4.7603	0.4112	0.9368	0.0497	0.0039	0.0625
3	2	500	30	lbfgsb	500	3.0381	0.3317	0.9504	0.0377	0.0029	0.0539
3	2	500	30	slsqp	100	0.6677	0.3150	0.9381	0.0423	0.0036	0.0601
3	2	500	40	adam	300	6.3144	0.4317	-0.1775	0.2187	0.0716	0.2675
3	2	500	30	adam	300	6.4085	0.3127	0.4604	0.1483	0.0320	0.1790
3	2	500	20	adam	300	6.5455	0.2050	-0.3844	0.2725	0.1216	0.3487
3	2	500	30	slsqp	300	0.8991	0.3037	0.9407	0.0418	0.0035	0.0588
3	2	500	40	lbfgsb	500	3.7445	0.3938	0.9368	0.0497	0.0039	0.0625
3	2	500	30	slsqp	500	0.7934	0.3114	0.9415	0.0438	0.0034	0.0586
3	2	500	40	slsqp	100	0.8704	0.3931	0.9337	0.0496	0.0041	0.0641
3	2	500	40	slsqp	300	0.8486	0.4151	0.9323	0.0507	0.0042	0.0648
3	2	500	40	slsqp	500	0.6838	0.3874	0.9203	0.0537	0.0050	0.0706
3	2	500	20	lbfgsb	500	6.6972	0.2034	0.9133	0.0638	0.0076	0.0874
3	2	500	20	slsqp	100	0.7242	0.2111	0.8840	0.0731	0.0103	0.1013
3	2	500	20	slsqp	300	1.0996	0.1955	0.8889	0.0725	0.0098	0.0992
3	2	500	20	slsqp	500	0.9023	0.2010	0.8778	0.0770	0.0108	0.1040
3	2	500	40	adam	500	10.4428	0.4060	-0.1420	0.1825	0.0731	0.2704
3	2	500	30	adam	500	10.7049	0.3085	0.3561	0.1550	0.0380	0.1949
3	2	500	20	adam	500	10.9021	0.1697	0.5527	0.1526	0.0384	0.1961

Figure E4

Quantum Neural Networks Performance Metrics - 500 Samples and 80/20 Split

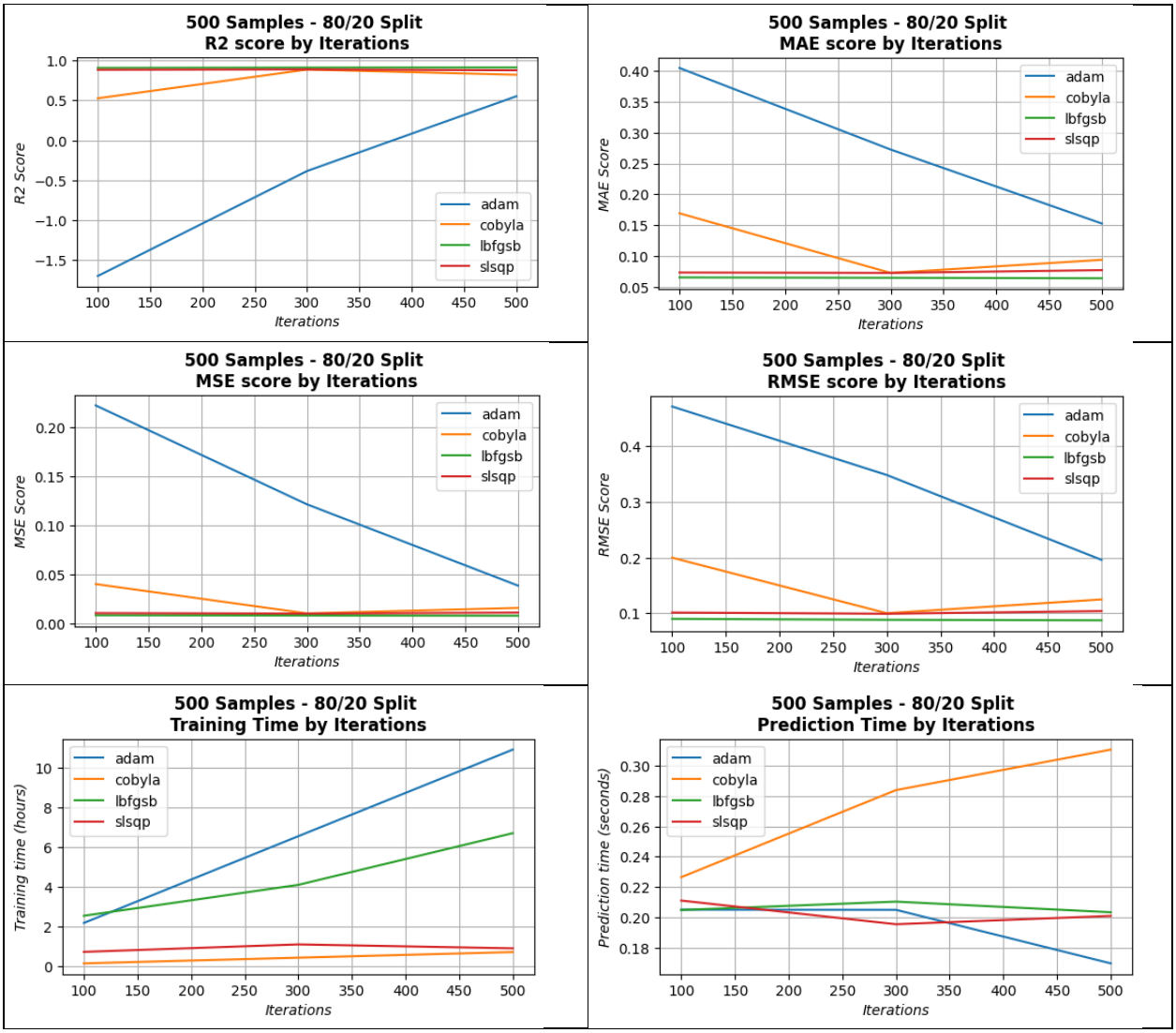


Figure E5

Quantum Neural Networks Performance Metrics - 500 Samples and 70/30 Split

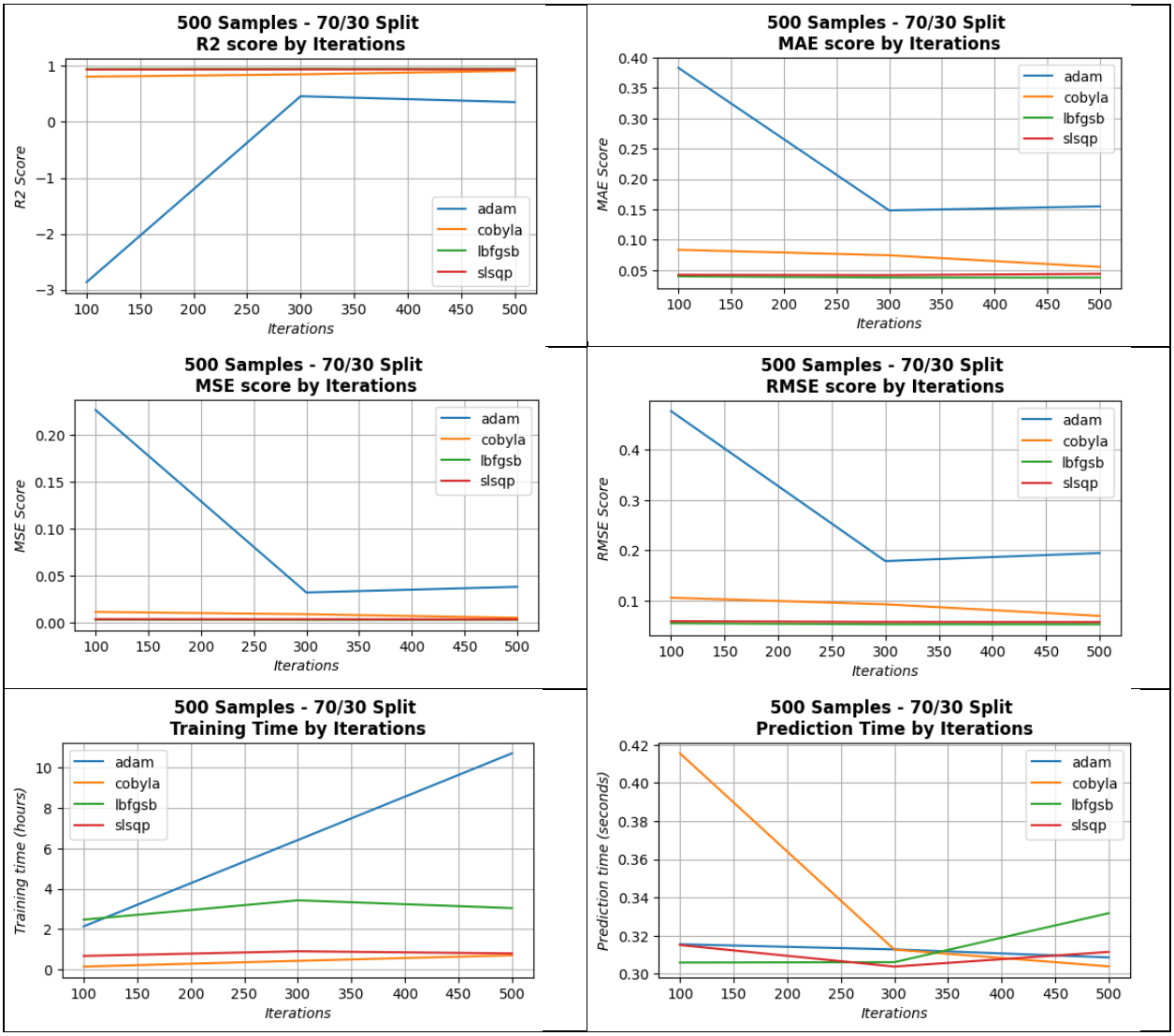


Figure E6

Quantum Neural Networks Performance Metrics - 500 Samples and 60/40 Split

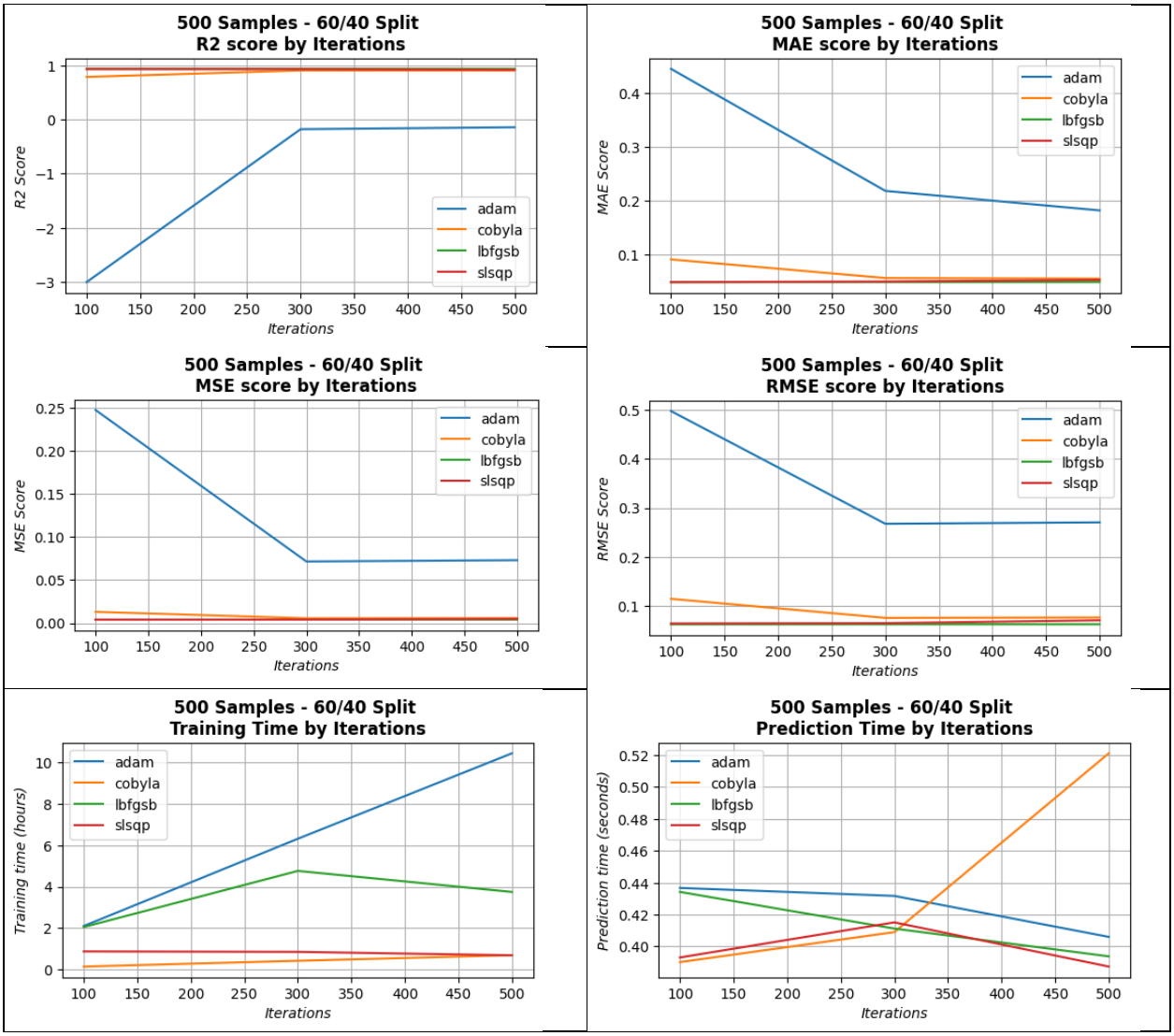


Table E3

Quantum Neural Networks Raw Training Results Using 1000 Samples

n_features	n_targets	n_samples	test_size	optimizer	iterations	training_time	prediction_time	r2	mae	mse	rmse
3	2	1000	30	cobyla	100	0.2945	0.3112	0.5260	0.1389	0.0312	0.1767
3	2	1000	20	cobyla	100	0.2980	0.2058	0.1837	0.1943	0.0562	0.2372
3	2	1000	30	cobyla	300	0.8782	0.3049	0.6816	0.1166	0.0210	0.1450
3	2	1000	20	cobyla	300	0.8857	0.2014	0.7608	0.1077	0.0167	0.1292
3	2	1000	40	cobyla	100	0.2950	0.4329	0.6792	0.1106	0.0203	0.1426
3	2	1000	30	cobyla	500	1.4713	0.3214	0.8519	0.0781	0.0098	0.0988
3	2	1000	20	cobyla	500	1.4887	0.2002	0.7252	0.1139	0.0191	0.1381
3	2	1000	40	cobyla	300	0.8740	0.4231	0.7669	0.0951	0.0148	0.1216
3	2	1000	40	cobyla	500	1.4608	0.5129	0.8320	0.0844	0.0106	0.1031
3	2	1000	30	lbfgsb	100	5.2341	0.2962	0.8934	0.0643	0.0070	0.0837
3	2	1000	20	lbfgsb	100	5.6265	0.2018	0.8379	0.0876	0.0113	0.1065
3	2	1000	30	adam	100	4.3896	0.2916	-8.4417	0.7126	0.6267	0.7917
3	2	1000	20	adam	100	4.4346	0.1957	-3.8717	0.4698	0.3231	0.5684
3	2	1000	40	lbfgsb	100	5.3579	0.3952	0.8848	0.0699	0.0073	0.0854
3	2	1000	40	adam	100	4.3621	0.3919	-5.7963	0.5433	0.4321	0.6574
3	2	1000	30	lbfgsb	300	8.9245	0.3256	0.8957	0.0641	0.0069	0.0828
3	2	1000	20	lbfgsb	300	10.7592	0.2090	0.8387	0.0879	0.0113	0.1062
3	2	1000	40	lbfgsb	300	11.3744	0.4130	0.8848	0.0695	0.0073	0.0854
3	2	1000	30	adam	300	13.1385	0.2616	-0.3732	0.2290	0.0912	0.3019
3	2	1000	20	adam	300	13.3049	0.2281	-0.5920	0.2808	0.1099	0.3316
3	2	1000	40	adam	300	13.0347	0.4095	-0.2178	0.2302	0.0772	0.2778
3	2	1000	20	lbfgsb	500	6.9201	0.2182	0.8275	0.0910	0.0121	0.1101
3	2	1000	20	slsqp	100	1.8487	0.2236	0.8034	0.0987	0.0136	0.1167
3	2	1000	20	slsqp	300	2.0982	0.2120	0.7859	0.0996	0.0152	0.1234
3	2	1000	30	lbfgsb	500	13.8765	0.3060	0.8956	0.0642	0.0069	0.0829
3	2	1000	20	slsqp	500	1.4257	0.1835	0.7694	0.1043	0.0161	0.1269
3	2	1000	40	lbfgsb	500	10.2676	0.4051	0.8836	0.0699	0.0074	0.0858
3	2	1000	30	slsqp	100	2.0808	0.3192	0.8629	0.0742	0.0090	0.0950
3	2	1000	40	slsqp	100	1.5894	0.4064	0.8786	0.0708	0.0077	0.0877
3	2	1000	30	slsqp	300	1.8703	0.3071	0.8727	0.0715	0.0084	0.0915
3	2	1000	40	slsqp	300	1.7669	0.4162	0.8824	0.0707	0.0074	0.0863
3	2	1000	30	slsqp	500	1.6160	0.3163	0.8709	0.0715	0.0085	0.0922
3	2	1000	40	slsqp	500	1.5968	0.4099	0.8760	0.0707	0.0078	0.0886
3	2	1000	30	adam	500	22.1203	0.3292	0.0058	0.2070	0.0658	0.2565
3	2	1000	20	adam	500	22.3477	0.1899	-0.3495	0.2648	0.0958	0.3095
3	2	1000	40	adam	500	21.9940	0.4419	-0.3058	0.2281	0.0829	0.2879

Figure E7

Quantum Neural Networks Performance Metrics - 1000 Samples and 80/20 Split

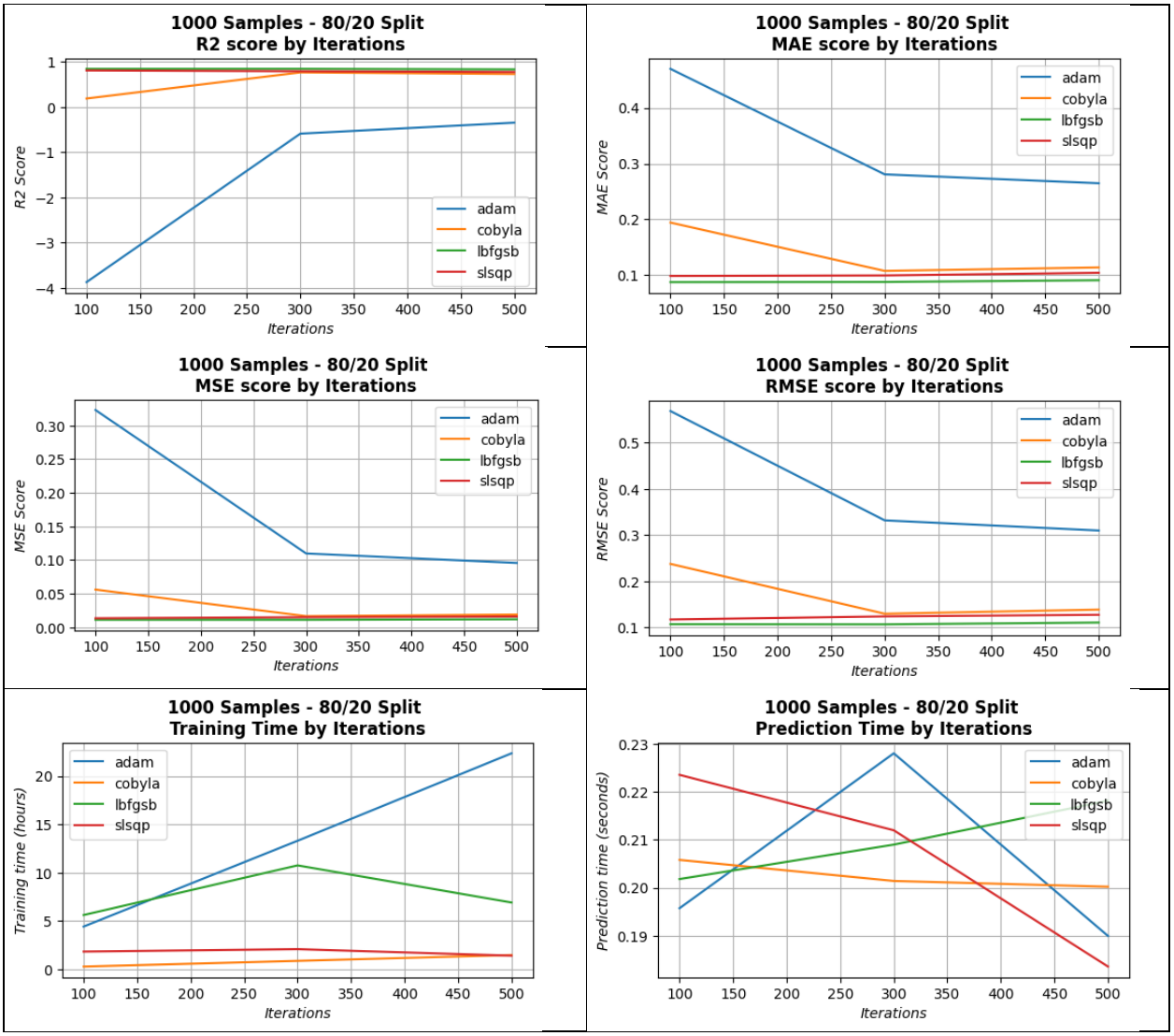


Figure E8

Quantum Neural Networks Performance Metrics - 1000 Samples and 70/30 Split

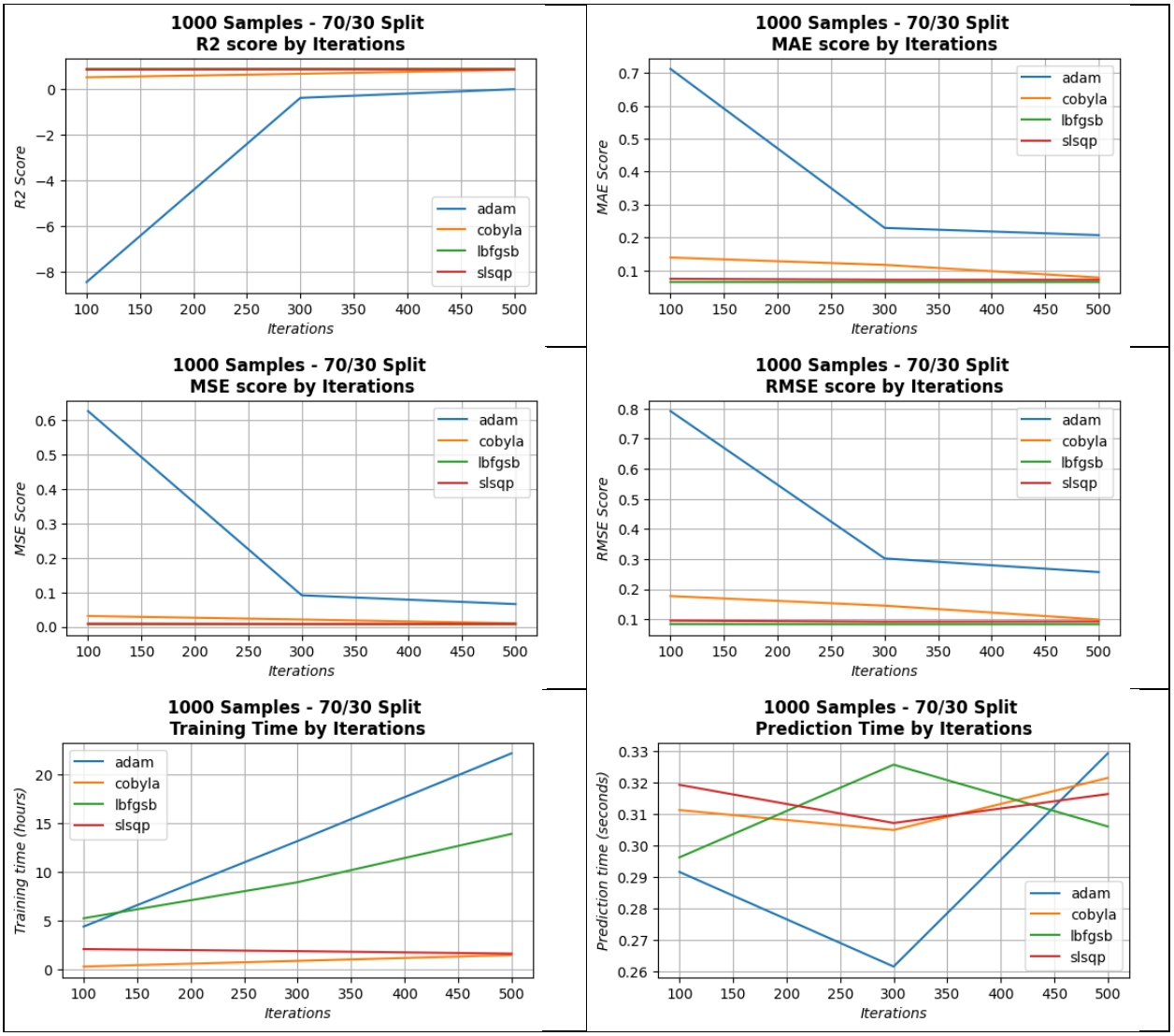


Figure E9

Quantum Neural Networks Performance Metrics - 1000 Samples and 60/40 Split