Spring 2022

Quadratic Neural Network Architecture as Evaluated Relative to Conventional Neural Network Architecture

Reid Taylor
University of South Carolina

Quadratic Neural Network Architecture as Evaluated Relative to Conventional Neural Network Architecture

By

Reid Taylor

Submitted in Partial Fulfillment
of the Requirements for
Graduation with Honors from the
South Carolina Honors College

May, 2022

Approved:

Paula Vasquez
Director of Thesis

Andrei Medved
Second Reader

Steve Lynn, Dean
For South Carolina Honors College

Abstract

Current work in the field of deep learning and neural networks revolves around several variations of the same mathematical model for associative learning. These variations, while significant and exceptionally applicable in the real world, fail to push the limits of modern computational prowess. This research does just that: by leveraging high order tensors in place of 2^{nd} order tensors, quadratic neural networks can be developed and can allow for substantially more complex machine learning models which allow for self-interactions of collected and analyzed data. This research shows the theorization and development of mathematical model necessary for such an idea to work appropriately in an analogous fashion to current models, and then explores through Monte-Carlo simulations the industry-standard measures of fit of such a model.

Contents

Quadratic Neural Network Architecture

As Evaluated Relative to Conventional Neural Network Architecture

Reid Taylor

April 12, 2022

1 Introduction

Artificial neural networks are powerful mathematical and computational tools used to interpret complex domains of data into discrete codomains of data. Common applications of these are to determine the features of an image or an audio sample, though these robust algorithms may be applied to any domain of data.

Artificial neural networks were conceived of as early as 1948, with Alan Turing's hypothesis of a "B-type unorganized machine" in his paper *Intelligent Machinery* [6]. Development on this class of algorithm advanced slowly over the next twenty years before stagnating due to limitations of the existing technology and raw processing power. During the 1980's, neural network advances continued as processing power had caught up with the requirements of theorized approaches, and with the development of a back-propagation algorithm allowing for the neural network to correct itself dynamically. Since the inception of these networks, though, the form and the defining forward-propagation equations have experienced minimal changes.

The general form of a neural network is shown in Figure 1. While this specific example translates input data from a 10-dimensional domain into a one dimensional output, through one *hidden* layer, the dimensions are arbitrary as is the number and size of all intermediate layers of the network, making this type of algorithm highly adaptable for computing and predicting with large domains or codomains of data.

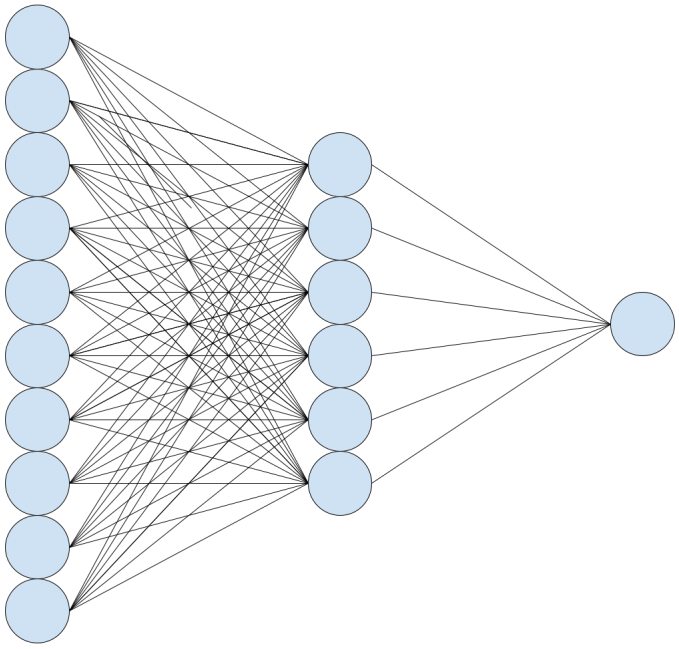

Figure 1: The organization of the traditional neural network model. Between each layer of nodes there exists a cross hatch of connections between every node, hence the term "fully connected layers". The degree of interconnection between layers, the number of intermediate layers, and the number and distribution of nodes throughout the network are user-configurable.

Of particular interest is the fact that every node, or element, of layer (j-1) within the network is connected to every node of layer (j). The connections between layers can be modeled as:

$$h_j = \sigma_j(W_j \cdot \vec{x}_{j-1} + \vec{b}_j) \tag{1}$$

Where σ is the non-linear activation function, W is the weight matrix, and $\vec{b}$ is the bias vector. Each of these are unique to each layer within the network. Similarly the input vector, $\vec{x}_{j-1}$, is denoted to mean the output of the previous layer. In the case j=1, the input vector represents the input data to the network itself.

This traditional structure allows for the composition of an activation function, σ, upon a linear function $(W \cdot \vec{x} + \vec{b})$. This composition, complete with only one layer within the network as a whole, is theoretically capable of approximating any given function [1].

Many derivatives of this structure exist to fit specific needs. Convolutional Neural Networks are artificial neural networks where the dot product between the weight

matrix, W, and the input vector, $\vec{x}$, is replaced by a cyclic convolution operation. Convolutional neural networks are exceptionally suited for image recognition at high resolution with low computational cost, thanks to the fast Fourier transform and convolution rule [5].

Furthermore, recurrent neural networks exist as an evolution of neural networks such that both the weight matrix and the bias vector are shared across all layers of the neural network, and optionally with the ability to implement new features at any given layer of the network, and to dually draw conclusions from any layer of the network. This allows for a great deal of flexibility in the neural network itself, and requires very few parameters as compared to standard neural networks. However, due to the recursive definitions between layers, eigenvalue decomposition occurs within recurrent neural networks of several layers.

As it stands today, multiple neural network architectures exist to meet certain requirements, though each has certain disadvantages which must be weighed when determining what architecture would best serve a particular need.

This paper introduces a novel form of neural network architecture in which a new term is substituted in place of the weight matrix for the definition of each layer, taking advantage of higher ordered tensors to provide interactions among the input data values. The new definition is given by

$$h_j = \sigma_j(\vec{x}_{j-1} \cdot V_j \cdot \vec{x}_{j-1} + \vec{b}_j) \tag{2}$$

Where V is the third order weight tensor, and all else is as before.

This new layer definition is inspired by new biological evidence which indicates that "quadratic" interactions between neurons occur as defined above [4].

2 Methodology

2.1 Practical Implementation

In order to construct a novel neural network structure, it is necessary to do so without using existing architecture to ensure complete control over the interactions within and between layers. As such, this project is completed in python with the use only of the NumPy package. Only fundamental operations such as matrix operations are predefined and imported; all other operations used throughout the project are defined as needed. An object oriented programming approach is employed for best practice to allow modularity in implementation: this means that the neural network is described by several distinct layer objects grouped into a single network object; each of these objects have distinct properties and functions.

2.2 A Granular Examination of the Network Implementation

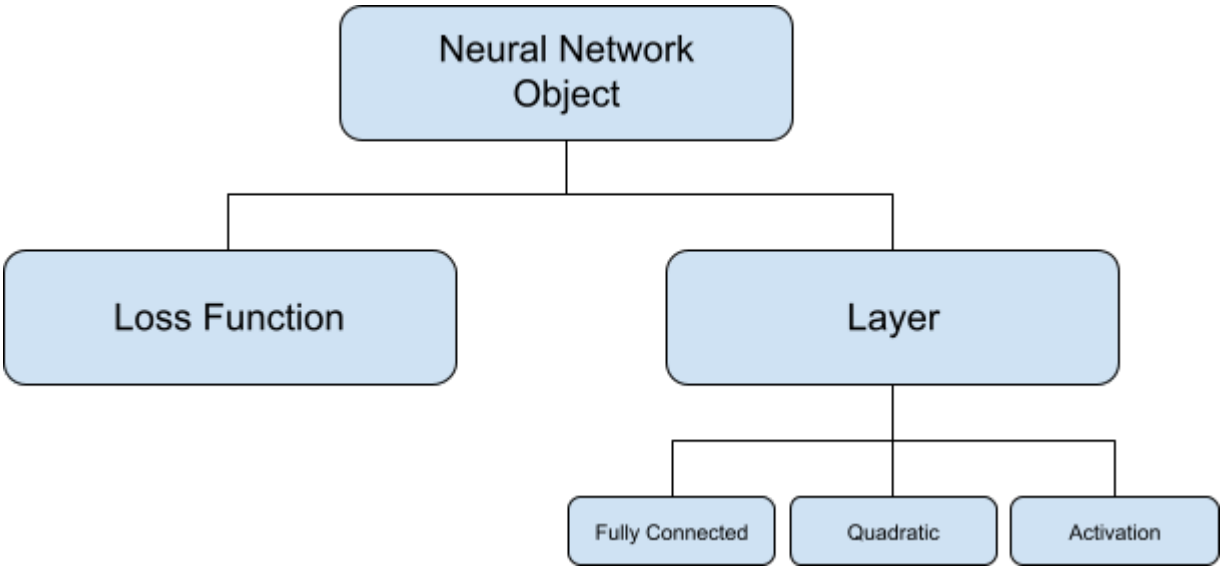

Figure 2: The structure of neural networks. Note: this structure applies to conventional artificial neural networks as well as the design proposed by this paper.

As shown in the figure, the distinction between the conventional artificial neural network and the new quadratic neural network exists only within the definition for the layer. Both models are otherwise identical in construction.

A neural network object has many properties and functions intrinsic to it. Most important of these is the loss function, the learning rate, and the network layers used. The loss function and the learning rate are configurable by the user, and do not directly affect the layers of the network, though they do eventually interact with the layers. The neural network object has three different layer types. In conventional neural networks, there are only fully connected layers and activation layers. In the neural network object created and experimented with within this paper, there exists a quadratic layer as well.

Fully connected layers and quadratic layers are interchangeable, and must be superseded by an activation. These properties and functions will be discussed in depth through the rest of this section.

2.2.1 The Layer Archetype

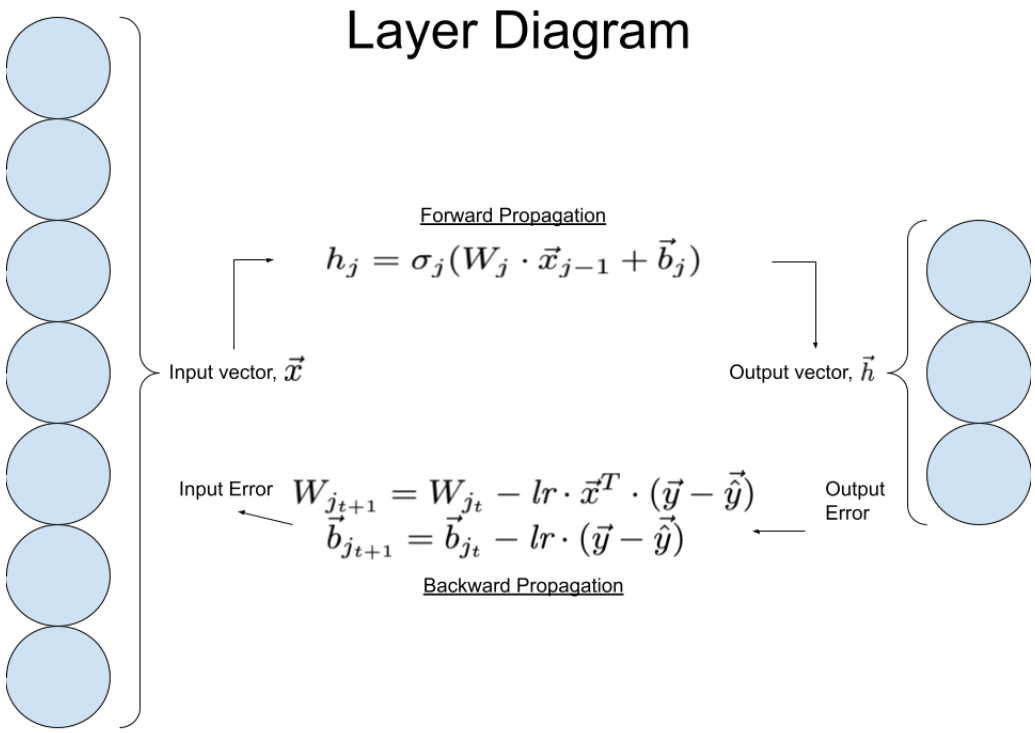

Figure 3: The structure of a conventional neural network layer, where "lr" represents the learning rate parameter.

The layer is the most base level of the neural network object, and has one function for forward propagation and several for backward propagation. All classes of layers consist of these properties and methods, each building upon them uniquely as required by the network and as supported by theory. The input and the output of any given layer is a vector of configurable dimension.

The forward propagation method is defined to take as its argument the input vector of data and to return an output vector.

The backward propagation method takes as arguments the output error value and a learning rate. The output error vector details the magnitude by which each of the layer's respective nodes varies from the "true" value, the value which would contribute to a perfectly accurate prediction. The learning rate is a network parameter which determines to what magnitude the neural network should adjust its weight parameters, or how "quickly" the network should learn. This is important as a network which learns too slowly may be computationally inefficient, while one which learns too quickly might overstep and could never achieve a local minimum when minimizing the loss function.

The learning rate is a scalar commonly restricted to be an element of [0, 1) which scales the amount by which individual corrections are applied to each node within the network.

2.2.1.1 The Fully Connected Layer

The first child class of the Layer Archetype is a fully connected layer. The forward-propagation method of this child class is defined by

$$h_j = W_j \cdot \vec{x}_{j-1} + \vec{b}_j \tag{3}$$

The weight matrix, $\mathbf{W}$, is an element of $\mathbf{R}^{n \times m}$, where n represents the dimension of the data used as input to the fully connected layer, and m represents the dimension of the output data vector from this fully connected layer. At instantiation, the weight matrix is constructed of appropriate dimensions with each entry a random value selected from the interval [-0.5, 0.5)

The bias, $\vec{\mathbf{b}}$, is a vector of dimension m, with m likewise representing the output data vector dimensions. At instantiation, the bias vector is constructed with each entry set to zero.

The backward propagation function of a fully connected layer is designed to calculate by what magnitude each of the layer's contained parameters should adjust

in response to the cost function of the neural network.

$$W_{t+1} = W_t - lr \cdot \vec{x}^T \cdot (\vec{y} - \vec{\hat{y}}) \tag{4}$$

Where the backward propagation calculation for a fully connected layer's weights is given by equation 4. W_{t+1} represents the new weight matrix based on the backward propagation, while W_t represents the weight matrix at the time of calculation of the layer's output. lr denotes the learning rate property defined as a hyper-parameter in section 2.2.1. $(\vec{y} - \vec{\hat{y}})$ denotes the error calculated to originate from the layer's output.

The second component of the backward propagation calculation for a fully connected layer is given by

$$\vec{b}_{t+1} = \vec{b}_t - lr \cdot (\vec{y} - \vec{\hat{y}}) \tag{5}$$

with $\vec{b}$ representing the bias vector as given by equation 5

The error calculated by the layer's output is discussed in depth in section 2.2.3.

It is important to note that a fully connected layer of input dimension n and output dimension m contains at most $(n \cdot m + m)$ unique parameters.

2.2.1.2 The Quadratic Layer

The next child class of the Layer Archetype is the newly designed quadratic layer.

The tensor V is defined as an element of $\mathbf{R}^{n \times m \times n}$, with n and m still representing the dimensions of the layer's input and output, respectively. At instantiation, this tensor is randomly filled with values from the interval [-0.5,0.5).

The forward propagation function for a quadratic layer takes a different form than that of the fully connected layer in accommodating the tensor V, though the essence of the calculation is preserved

$$h_j = \vec{x}_{j-1} \cdot V_j \cdot \vec{x}_{j-1} + \vec{b}_j \tag{6}$$

The backward propagation function for a quadratic layer inherits from the fully connected layer the calculations for the bias vector's adjustments. The backward propagation function introduces a new weight tensor error calculation for V, as

$$V_{t+1} = V_t - lr \cdot \vec{x}^T \cdot (\vec{y} - \vec{\hat{y}}) \cdot \vec{x} \tag{7}$$

Where V_{t+1} represents the new quadratic weight tensor based on the backward propagation, while V_t represents the weight tensor at the time of calculation of the layer's output. The other parameters present are as they were for the back-propagation algorithm in the fully connected layer.

Observe here that a quadratic layer of input dimension n and output dimension m contains at most $(n \cdot m \cdot n + m)$ unique parameters. That is, this layer contains at most $(n \cdot m \cdot (n\text{-}1))$ more unique parameters than a fully connected layer does.

2.2.1.3 The Activation Layer

You might observe that the forward propagation functions for each of the previous two layers look similar to the definition of a layer discussed earlier. Specifically, the formula for a conventional layer's forward propagation is

$$h_j = W_j \cdot \vec{x}_{j-1} + \vec{b}_j \tag{8}$$

while the formula given to define a neural network layer at the beginning of this paper was

$$h_j = \sigma_j(W_j \cdot \vec{x}_{j-1} + \vec{b}_j) \tag{9}$$

with σ representing the non-linear activation function of a layer.

These two formulae vary only by the presence of this activation function. In the neural network objects created for the purposes of this paper, σ exists as a unique and independent layer, to be composed upon the previous conventional or quadratic layer.

This activation layer does not have individual weights nor biases, but instead has a non-linear activation function associated with it. This class has a forward and backward propagation function, though these too are unique from the other layers discussed.

The specific activation function is to be chosen from an available list of defined functions. These are the standard activation functions used in other libraries of machine learning: *tanh*, *ReLU*, *Sigmoid*, and the *identity* functions.

For the purpose of this research, the sigmoid and the sigmoid prime functions are implemented in research and experimentation. The exact definitions of these used are as follows:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{10}$$

$$\sigma'(x) = \frac{1}{1 + e^{-x}} \left(1 - \frac{1}{1 + e^{-x}}\right) \tag{11}$$

The forward propagation function for the activation layer applies the chosen activation function to the layer's input vector in an element-wise operation.

The backward propagation function for an activation layer does not act in the same manner as the backward propagation functions discussed for other layers. There are no parameters associated with the activation layer, so the role of backward propagation is to simply feed backwards an appropriate error value to previous layers.

Once this layer is composed upon either a conventional OR a quadratic layer, then the layer definition given by 9 is complete.

2.2.2 The Loss Function

The neural network object additionally has a loss function. This function is necessary to determine the extent to which a prediction made by the neural network is incorrect. Traditionally, the chosen loss function is the mean squared error, given by

$$MSE = \frac{1}{n}\sum_{k=0}^{n}(y - \hat{y})^2 \tag{12}$$

This loss function determines the output error of the last layer of the neural network object. As the final layer completes backwards propagation, as detailed throughout section 2.2.1, the final layer adjusts the weight matrix and the bias vector according to the values which would have theoretically contributed to a perfect prediction–this statement is equivalent to the loss function being minimized.

The adjustments made to all layers prior to the final layer in the organization of the neural network are difficult to visualize without the help of a useful diagram. Please see the following useful diagram:

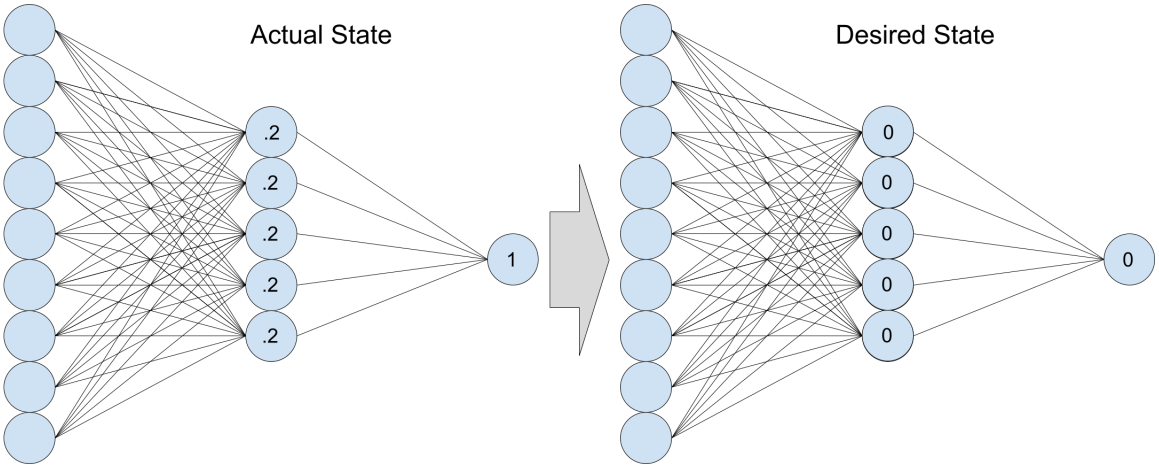

Figure 4: Transitioning from actual to desired results in a neural network object, broken down between layers.

If the final output needs to correct from predicting a 1 to predicting a 0, we can imagine the backwards propagation of error as communicating to each componenet node from the previous layer the following: "You contributed .2 to my total; however, if you had instead contributed 0, then the outcome would have been perfect. The output error for you is 0.2."

The node which was off by 0.2 would then communicate to every node in the layer before it how much those nodes could each increase or decrease by in order to assist in achieving a perfect prediction. This recursive communication continues until the very first layer has been adjusted based on these "back-propagated" values.

This hypothetical situation and diagram assume equal contributions from each node in the previous layer, which is unrealistic, but it does lend a helpful visual for how this process works.

It is important to note that the weights and bias parameters are never changed to the exact extent that the error dictates, but rather a fraction of that magnitude. Specifically, the learning rate of the neural network represents this fraction by which a model adjusts itself. This will be better explained in the next section in the context of a training cycle as a whole.

2.2.3 The Training Function

The network is able to be trained on any amount of data. When training is initiated, all weight matrices are reduced to random valued entries and the bias vectors of the network's layers are reduced to zero vectors, effectively stripping the network

of any previous training. The training function requires input data of the correct dimension and labels of these data to judge the network's output when evaluating these data.

In addition, it accepts as input the number of epochs to iterate over and the learning rate to implement. An epoch is a complete cycle for all data: put in other words, one epoch is complete when the training cycle has been successfully completed for every point of data provided to the model. Training a model over multiple epochs is good practice; however, excessive epochs in training are both computationally expensive and dangerous, as a model can become too used to the specific training data and might lose the ability to generalize to real world data if it is trained on the same data for too many epochs.

For each epoch, every sample of the input data is randomly fed through the network through each sequential layer's forward propagation method, and then the final output of this process is used in the calculation of the loss function. The value returned by the loss function is then used in calculating the error of the model, and thus we gain insight into what changes to the model's weight parameters need be implemented through the backward propagation method in order to make the network as a whole more accurate. This process corrects the network's layer's parameters at a rate proportional to the learning rate before the next epoch is run.

3 Results

In order to properly assess the neural network structure in comparison to a traditional network, both created models are limited to a shallow network design, only with an input layer and either the fully connected or the quadratic layer following.

Both networks are constructed with identical hyper-parameters, meaning: the number of layers, the number of nodes within each layer, the learning rate, and the activation functions in each respective layer, the network's loss function, as well as the number of epochs in use during the training stage. In maintaining experimental control, the same data is used to train and score the networks.

3.1 Methodology of Results

In adherence to the requirements of this research, the networks are each trained and evaluated with the MNIST database of images of handwritten digits and the associated labels provided[3].

In evaluating the results of the study, proper experimental design requires appropriate and pre-defined criteria by which to determine an objectively superior model architecture: each model, relative to the other, will be evaluated on the following:

1. The accuracy of the model at the optimal epoch of performance
2. The number of epochs required to attain this level of accuracy
3. The total computational time required in training to reach this epoch
4. The average time delta between epochs and the associated gain in accuracy per epoch

3.2 Experimental Results

Below are the four measures of this experiment, visualized while prioritizing intuitive results.

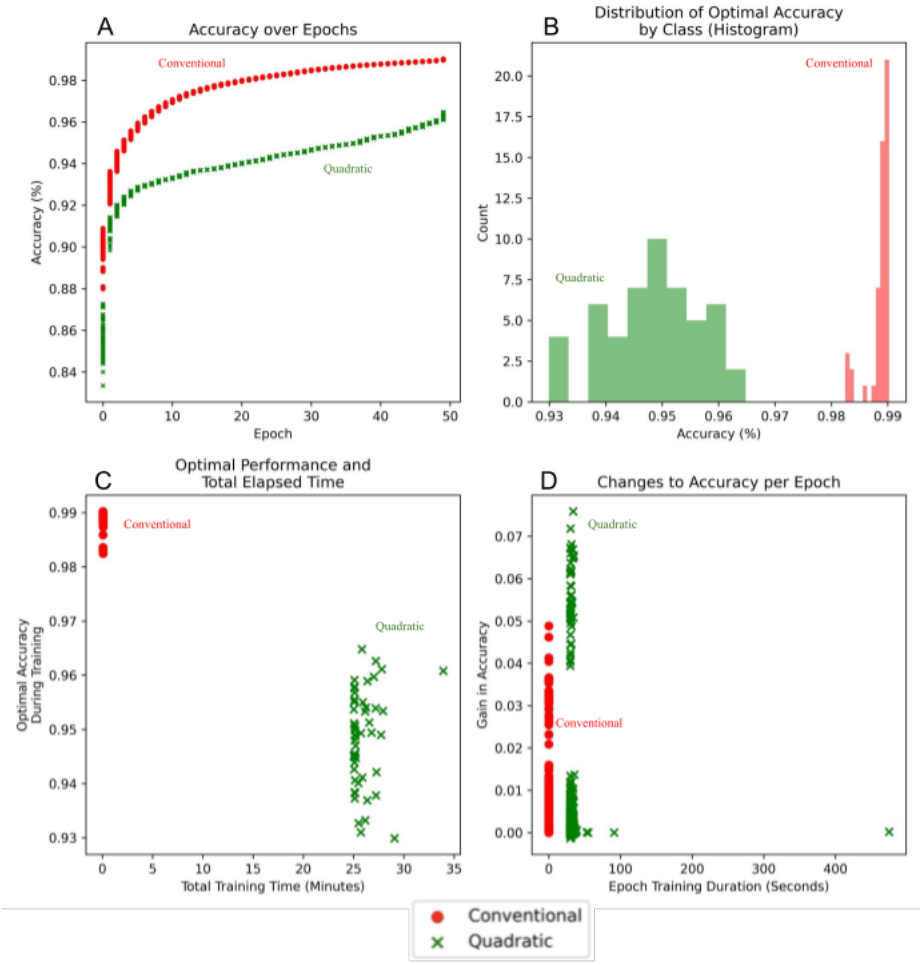

Figure 5: All figures display both the quadratic structure (green) and the conventional structure (red). A: Training accuracy over the course of 50 epochs as the model trains itself to the data provided. The lines shown are a mean representation of exactly 51 random trials for each neural network structure. B: Histogram of the distribution of each of the 51 trials per class of neural network as delimited by the optimal level of accuracy achieved through each trial. C: Scatter plot display of the optimal level of accuracy achieved as a function of the total training time of the model up to that achievement. D: Scatter plot displays the change in accuracy from one epoch to another as a function of the duration of each respective epoch.

Upon first glance, it is clear that there are obvious discrepancies in all collected measures of performance between the two models.

The average accuracy of the two neural network classes varies by a near constant 4% throughout all fifty epochs of training. However, it is important to note that both networks follow the same general trend of increases in accuracy over the course of the fifty epochs, despite this seeming vertical transform between the two classes. Over the course of more training epochs, one could hypothesize that the two classes might converge to the same level of accuracy, though that is a negligible point of argument as the models would likely overfit their parameters to the data used in training and would thus be unable to themselves extrapolate and accurately predict outcomes for novel data. Additionally, fifty epochs is an industry standard and the computational stress of training for a greater number of epochs outweighs the benefits provided by a 4% gain in accuracy.

The distribution of optimal levels of accuracy as separated by the two classes of neural networks is an interesting point to consider. This histogram provides evidence that the conventional neural network, on average, will consistently outperform quadratic neural network in measures of accuracy. It is statistically improbable that a quadratic neural network, evaluated on this dataset over 50 epochs, measures better accuracy than a conventional neural network.

Thirdly, the total training time necessary for a model to reach optimal performance is consistently and significantly greater for quadratic neural networks than it is for fully connected networks.

Lastly, the average duration of an epoch training period is again consistently and significantly greater for quadratic neural networks than it is for fully connected networks; however, it is important to note that the quadratic neural network consistently has significantly greater improvements in accuracy per any one epoch training period than the fully connected network has.

4 Conclusion Significance

It is important to note that, the results of this study aside, there are multiple factors which influence the efficiency of each model tested. For one, these models are vanilla in design, in that only the most fundamental operations are used to build the model, while neglecting many possible avenues of further growth (avenues which are regularly explored in commercial applications of neural networks). Many of the tools and methods of implementation of neural networks were not made available in this experimental design; these include, but are not limited to:

- Randomized dropout of neural connections
- Convolutional or Recurrent structuring
- Custom fit loss functions
- LSTM model style control gates
- Back propagation through calculations other than steepest descent gradient approach
- Data batching and mini batching
- Other, possibly more appropriate, activation functions [1]
- Adaptive learning rates

Exploration into these avenues of model refinement, and others, may prove fruitful in attaining even greater accuracy in the quadratic neural network architecture as given in this paper, and may even provide a way to reduce the number of parameters necessary to achieve such results, thereby favoring quadratic neural network architecture for challenges of regression or classification.

As technology advances further and further, we encroach upon the horizon of our current knowledge and capabilities to understand the knowledge we have access to. Quadratic neural network architecture is a promising way to better interpret and identify trends of high dimension data, and will certainly become a valuable tool for data science and machine learning in the coming years as its use is better researched, understood, and implemented in common avenues of the industry.

[1] It is interesting to consider the usage of activation functions in the context of quadratic neural networks; activation functions exist to introduce nonlinearity into the network. Further experimentation could reveal whether or not activation functions are truly beneficial for quadratic neural networks.

These results prove that while nearly equal performance is achievable, there exist significant differences in performance. As these models currently stand, the conventional neural network architecture is strongly favored over quadratic neural network architecture for computational efficiency.

References

[1] A.N. Gorban, D.C. Wunsch (1998) *The General Approximation Theorem*, IEEE, DOI: 10.1109/IJCNN.1998.685957

[2] F. Fan, W. Cong, G. Wang (2017) *Generalized Backpropagation Algorithm for Training Second-Order Neural Networks*, International Journal for Numerical Methods in Biomedical Engineering, DOI: 10.1002/CNM.2956

[3] Deng, L. (2012). *The MNIST database of handwritten digit images for machine learning research*, IEEE Signal Processing Magazine, 29(6), 141–142.

[4] P. Mantini, S. Shah (2020) *CQNN: Convolutional Quadratic Neural Networks*, Research Gate, DOI: 10.13140/RG.2.2.35842.71367

[5] G. Strang (2019) *Linear Algebra and Learning from Data*, Massachusetts Institute of Technology: Wellesley - Cambridge Press, ISBN 978-0-692-19638-0

[6] Copeland, B. Jack, ed. (2004). *The Essential Turing*. Oxford University Press. p. 403. ISBN 978-0-19-825080-7