Quadratic Neurons as Activation Functions in Deep Neural Networks

Mark Koszykowski & Andrew Lorber

The Cooper Union for the Advancement of Science and Art

ECE472 - Deep Learning

Prof. Curro

December 17th, 2020

Abstract *- In this paper, we build upon the work of Yaparla Ganesh and Rhishi Pratap Singh in Pattern Classification using Quadratic Neuron: An Experimental Study[1]. Their paper discusses the use of quadratic (QUAD) neurons to build quadratic neural networks, as opposed to the familiar linear neurons in multilayer perceptron (MLP) models. This paper displays the use of their quadratic neuron as a Quadratic Activation Function. Comparisons are made between the Sigmoid activation, ReLu activation, and two new types of Quadratic activations. Extensive experimentation was conducted on toy datasets, such as MNIST, CIFAR10, Spirals, Circles, and more. Empirical results display the use of these quadratic activation functions and motivates further study on the topic.*

1 Introduction

A good activation function is one of the most important parts of a successful model. When building linear layers, it is the non-linear activations between layers that allow the model to learn complex patterns in the data. Quadratic Neurons have been a topic of study for their use of a second-order equation $Ax^2 + By^2 + Cxy + Dx + Ey + F = 0$. By using Quadratic Neurons, non-linear decision surfaces can be found using simpler model architectures. This paper aims to adapt the use of Quadratic Neurons to activation functions. By taking advantage of Quadratic Neurons non-linearity, they can be included easily in a wide range of models as activations. Over the course of this paper, the Quadratic Activation Function will be introduced, thoroughly tested against alternative activations, and studied to determine its usefulness and effectiveness in everyday models.

2 Related Works

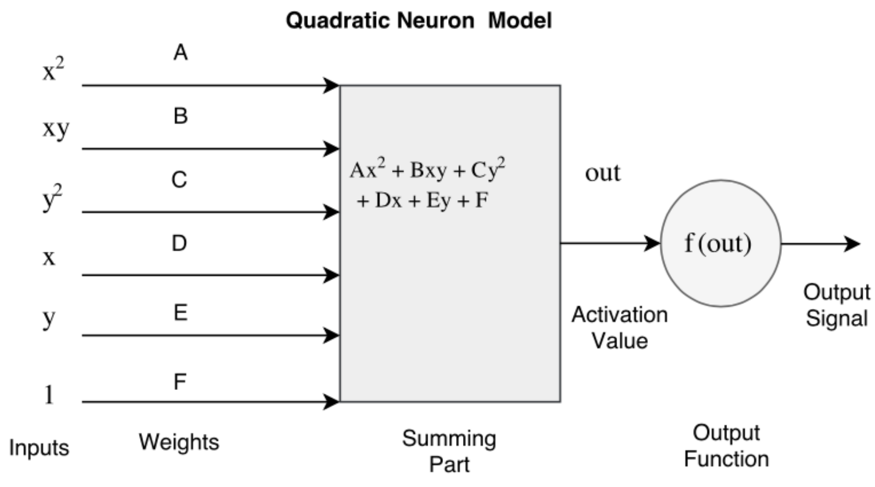


Figure 1: Generic Model of Quadratic Neuron

The work discussed in this paper is based on *Pattern Classification using Quadratic Neuron: An Experimental Study*[1], by Yaparla Ganesh and Rhishi Pratap Singh. In their paper, the authors introduce the Quadratic Neuron Model. As opposed to the commonly used perceptron $Wx + b$, This model is based on the second order generic equation in two dimensional space:

$$Ax^2 + By^2 + Cxy + Dx + Ey + F = 0 \tag{1}$$

Each neuron of this model has an input of two features and learns the parameters A, B, C, D, E, and F.

3 Quadratic Activation Function

The Quadratic Activation Function is an extension of Ganesh and Singh's Quadratic Neuron Model. Similar to their model, the output of this activation function is calculated using the non-linear equation (1). Since the activation function requires two input values, the width of each layer passing through this activation will be halved.

In Ganesh and Singh's model, each neuron had its own learned parameters A, B, C, D, E, and F. For the purpose of this study, two versions of the activation function were built: *The Quadratic Activation Function with Layer-Wide Coefficients* and *The Quadratic Activation Function with Per-Neuron Coefficients*. In these two versions of the activation, two neurons from the previous layer are chosen, used to calculate the rest of the input parameters, and used in equation (1) with the learned parameters. The activations described were implemented as custom Tensorflow Layers.

3.1 Layer-Wide Coefficients

This version of the Quadratic Activation Function uses a single set of coefficients (A, B, C, D, E, F) for each activation function within a given layer. For example, if this version of the Quadratic Activation Function were to be used following a 64 neuron dense layer, then this activation layer would learn only 6 coefficients.

3.2 Per-Neuron Coefficients

This version of the Quadratic Activation Function uses a different learned set of coefficients (A, B, C, D, E, F) for each activation function within a given layer. For example, if this version of the Quadratic Activation Function were to be used following a 64 neuron dense layer, then this activation layer would learn 192 ($\frac{64}{2} \times 6$) different coefficients, 6 per activation function.

3.3 Convolutional Layers

In addition to the activation functions described above, a variation of each was built in order to work with convolutional layers. While Ganesh and Singh did not discuss convolutional networks in their paper, we decided it would be worthwhile to test this activation layer following a convolutional layer. Unlike the Quadratic activation layer following a dense layer, in which the selection of neurons for x and y could be done randomly, the Quadratic activation layer following a convolution layer needed to use neighboring neurons, as to maintain the original image. Two versions of The Quadratic Activation Function for Convolutional Layers were made as well: Layer-Wide Coefficients and Per-Neuron Coefficients.

4 Comparison with Other Activation Functions

In this section, the two Quadratic Activation Functions (Layer-Wide & Per-Neuron Coefficients) will be compared to the Sigmoid activation and ReLu activation in multiple experiments. In each of the following experiments, the model architectures are identical except for the activation functions used. Tests were conducted on datasets using different amounts of hidden layers, in an attempt to conclude if relative accuracy changed as model complexity increased. For the purpose of these experiments, the linear and convolutional Quadratic Activation Functions were used interchangeably depending on the preceding layer.

4.1 Experiment: MNIST - Single Hidden Layer

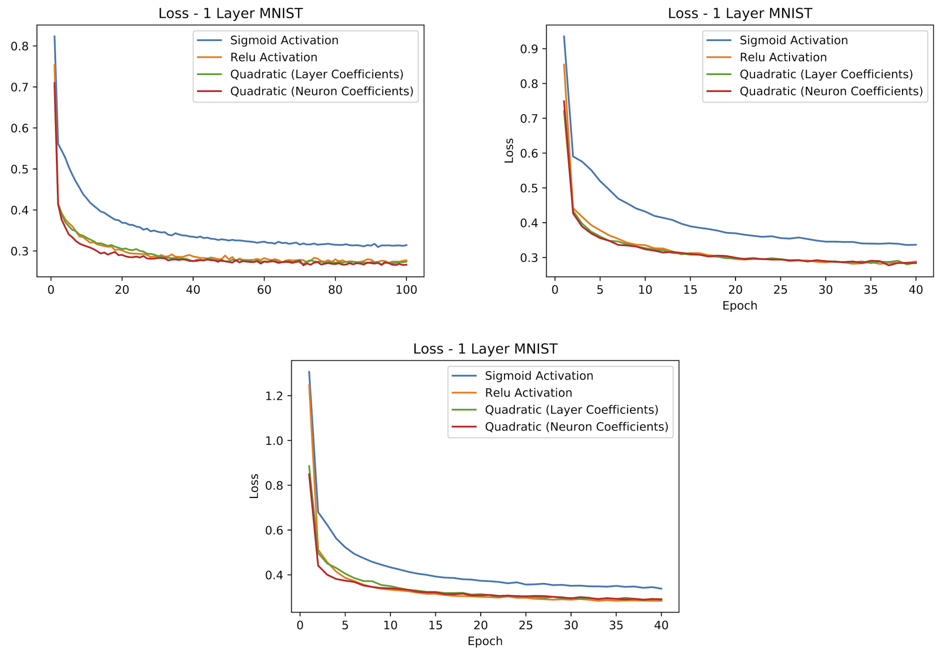
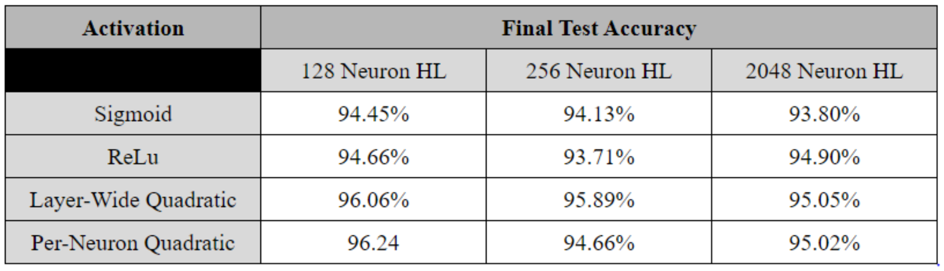


Figure 2: Loss vs. Epoch Graphs of Single-Hidden-Layer Models Trained on MNIST (Top Left) A 128 neuron hidden layer (Top Right) A 256 neuron hidden layer (Bottom) A 2048 neuron hidden layer

The first experiment was conducted on the MNIST dataset, which contains 70,000 (60,000 training & 10,000 testing) 28x28 images of handwritten digits. The resulting loss vs. epochs graphs displayed that while the two Quadratic activations and ReLu stayed fairly close



Activation	Final Test Accuracy		
	128 Neuron HL	256 Neuron HL	2048 Neuron HL
Sigmoid	94.45%	94.13%	93.80%
ReLu	94.66%	93.71%	94.90%
Layer-Wide Quadratic	96.06%	95.89%	95.05%
Per-Neuron Quadratic	96.24	94.66%	95.02%

Figure 3: Final Test Results of Single-Hidden-Layer Models Trained on MNIST

together, the Quadratic activations achieved a lower training loss for smaller hidden layers and ReLu achieved a lower training loss for larger hidden layers. However, in all cases the Quadratic activations achieved a higher test accuracy, albeit by a couple percentage.

4.2 Experiment: MNIST - Two Hidden Layers

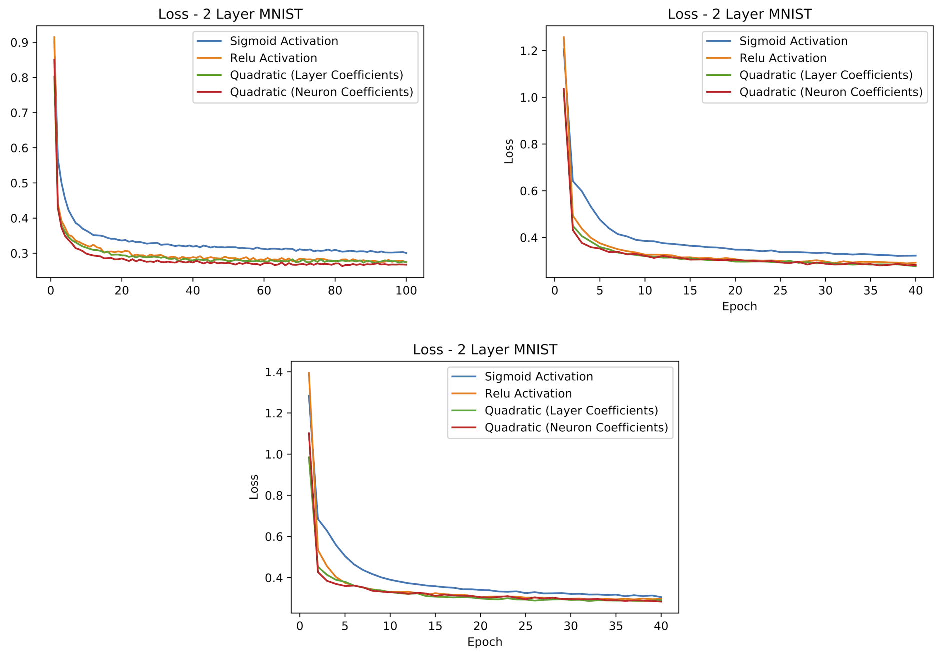


Figure 4: Loss vs. Epoch Graphs of Double-Hidden-Layer Models Trained on MNIST (Top Left) 64, 32 neuron hidden layers (Top Right) 256, 128 neuron hidden layers (Bottom) 512, 128 neuron hidden layers

In an attempt to determine if model complexity affected which activation was more effective, the MNIST dataset was used with two hidden layers of varying sizes. While a Quadratic activation always obtained the highest test accuracy, attesting to its usefulness, the final results were significantly closer, especially as the size of the hidden layers increased.

It would appear that the Quadratic activation works best with layers of smaller sizes. One hypothesis is that when one set of coefficients are used for the entire layer, a larger

Activation	Final Test Accuracy		
	64, 32 Neuron HL	256, 128 Neuron HL	512, 128 Neuron HL
Sigmoid	94.40%	93.44%	95.64%
ReLu	94.99%	95.02%	94.46%
Layer-Wide Quadratic	95.38%	95.70%	95.74%
Per-Neuron Quadratic	93.69%	95.54%	95.91%

Figure 5: Final Test Results of Double-Hidden-Layer Models Trained on MNIST

layer results in those 6 coefficients representing more functions, which would make them less accurate for each. When coefficients are trained per-neuron, then more coefficients are required to be trained, increasing the training time of the model.

4.3 Experiment: MNIST - Convolutional Neural Network

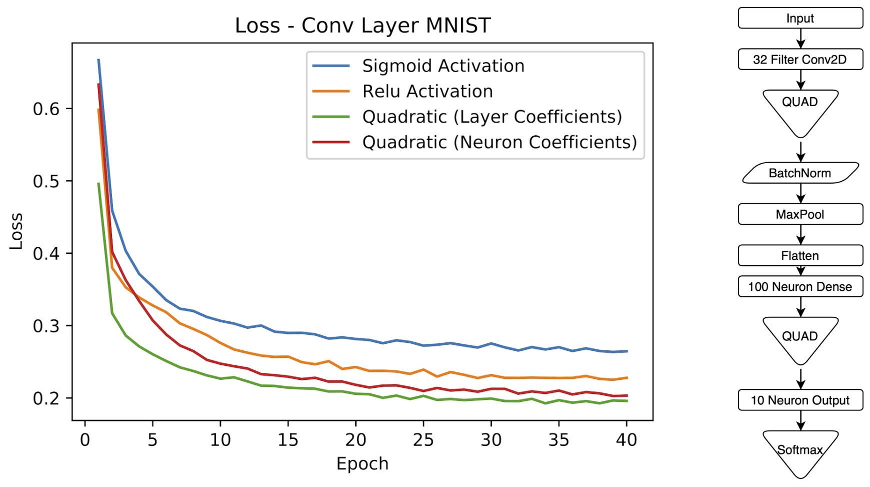


Figure 6: (Left) Loss vs. Epoch Graph of Convolution Neural Network Trained on MNIST (Right) CNN Architecture

Activation	Final Test Accuracy
Sigmoid	96.87%
ReLu	96.62%
Layer-Wide Quadratic	97.40%
Per-Neuron Quadratic	97.45%

Figure 7: Final Test Results of Convolution Neural Network Trained on MNIST

Another experiment was conducted on the MNIST dataset using a convolutional neural network; the model architecture is displayed in Figure 6. The use of a convolutional layer greatly increased the overall accuracy of the models, but the Quadratic activations achieved the two highest test accuracies. Viewing Figure 6, the Per-Layer Coefficients model understandably achieved a lower loss faster, while the Per-Neuron Coefficients model took more time to train the extra coefficients it has.

4.4 Experiment: CIFAR10 - Convolutional Neural Network

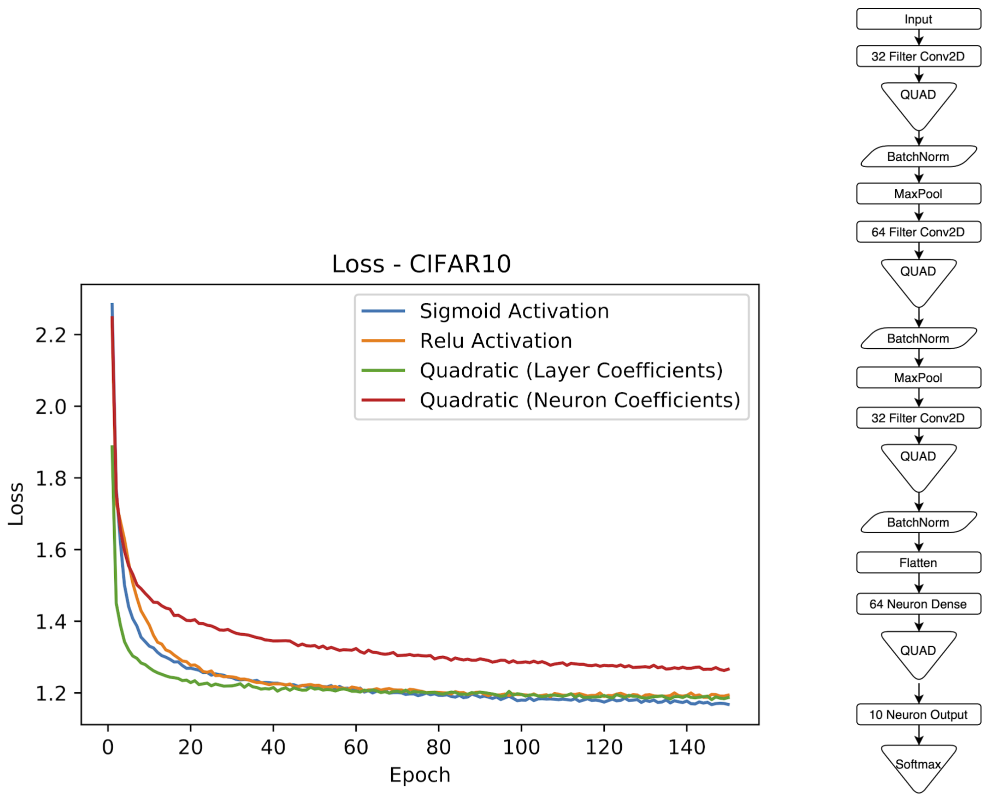
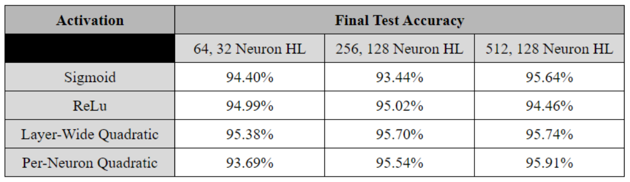


Figure 8: (Left) Loss vs. Epoch Graph of Convolution Neural Network Trained on MNIST (Right) CNN Architecture



Activation	Final Test Accuracy		
	64, 32 Neuron HL	256, 128 Neuron HL	512, 128 Neuron HL
Sigmoid	94.40%	93.44%	95.64%
ReLu	94.99%	95.02%	94.46%
Layer-Wide Quadratic	95.38%	95.70%	95.74%
Per-Neuron Quadratic	93.69%	95.54%	95.91%

Figure 9: Final Test Results of Convolution Neural Network Trained on MNIST

This experiment used the CIFAR10 dataset, which consists of 60,000 (50,000 training & 10,000 testing) $32 \times 32 \times 3$ images. The CNN architecture is displayed in Figure 8. What is interesting about the resulting loss vs epoch graph is the Per-Neuron Quadratic activation. Due to the large amount of trainable variables, the model using that activation function was unable to compete with the other models within the given time frame. Another test was run, in which the Per-Neuron model was allowed to train for significantly longer, but there was no substantial accuracy increase observed. While it is possible that given more time it would perform better, it is uncertain how long it would need to train.

As seen in Figure 9, the models in this experiment did not perform close to state of the art. One drawback of the Quadratic Activation Function is that the input is halved at every layer, so the maximum amount of layers is dependent on the input size. This makes it extremely difficult to implement the more complex network architectures that are required to accomplish a high accuracy in more complex datasets, such as CIFAR100. Since these experiments are meant to compare activations in equal models, this architecture, which is possible with Quadratic activations, was chosen. As seen in the MNIST experiment, though, when a smaller architecture is sufficient for the problem, the Quadratic Activation Function is useful and effective in increasing the final accuracy.

4.5 Difference in Model Complexity - MNIST

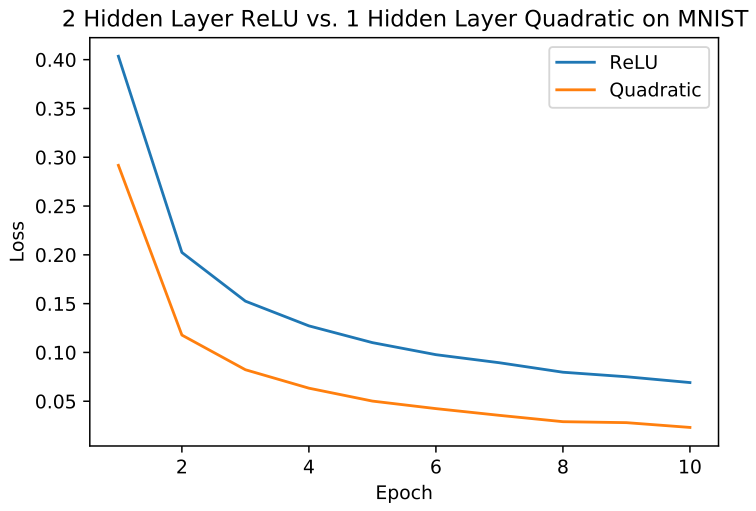


Figure 10: Loss vs. Epoch Graph of Models with different Complexities and Activation Functions

Given that ReLU and Quadratic both generally perform quite well, but Quadratic consistently outperformed ReLU on mirroring models, the two activation functions were put against each other in models of differing complexity. The intent was to demonstrate that while Quadratic did require extra computational time, this would result in a higher accuracy with a less complex model architecture. In this experiment, a two-hidden-layer network with ReLU's was compared to a one hidden-layer-network with Quadratic activations. The results were quite interesting.

In contrast to the mirroring models experiments, the computation of the Quadratic model was quicker than the computation of the ReLU model; albeit not by a significant amount (1 second), but by approximately 25%. Seeing as with this experiment, size 128 layers were used along with the Per-Neuron Quadratic, meaning that the Quadratic model had more parameters, this is quite ironic and surprising. Another thing that may come as a surprise was the general reduction in loss when using Quadratic. Though this may immediately sound like a benefit, this too was ironic. Even though the Quadratic produced marginally lower loses, its test accuracy was lower (by approximately one percent).

4.6 Experiment: Circles

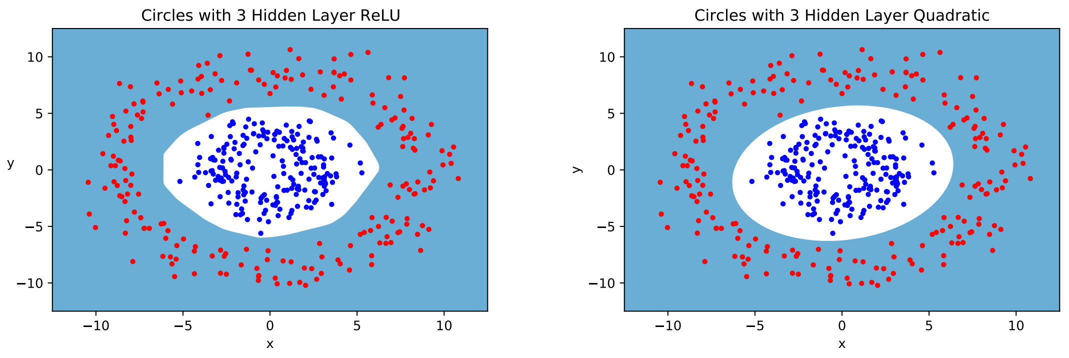


Figure 11: Binary Circle Classification with (Left) ReLU and (Right) Quadratic

One of the powerful features of this activation function was its ability to generalize well to classification boundaries that are inherently non-linear. A prime example of such a dataset is the circles dataset, as the classification boundary is ideally a perfect circle. The intent of this experiment was not to optimize accuracy, like previous ones, since it is expected that the model should produce a perfect accuracy, but rather to visualize the boundary that is consistently produced. Figure 10 demonstrates that the Quadratic Activation Function is a great candidate for sets like this. Not only are the training times naturally much quicker than on other activation functions such as ReLU, they produce a boundary that appears to avoid inevitable noise. This is shown by the linear parameterization of a circle with ReLU as opposed to a smooth circle with Quadratic.

4.7 Experiment: Circles Overlap

An area of future work that was explicitly listed by Ganesh and Singh was to test equation (1) on classification boundaries that are naturally open, that is to say, do not form a closed loop. One of the toy datasets that was created to test this was Circles Overlap. The goal with this experiment was to visualize what kind of classification boundary is produced when the dataset does not fit into previously tested constraints. The Circles Overlap proved to be an interesting experiment, as not only did it produce interesting results at the fine grain scale, but also at the big picture scale as well. In Figure 11, it can be seen that compared to ReLU, Quadratic seemed to produce a better estimation boundary for one of the specific circles groups. This result has many implications on application.

Due to the shape of the boundaries, it would appear as though ReLU would perform better if a simple binary classification of a dataset similar to this is needed; data that is not

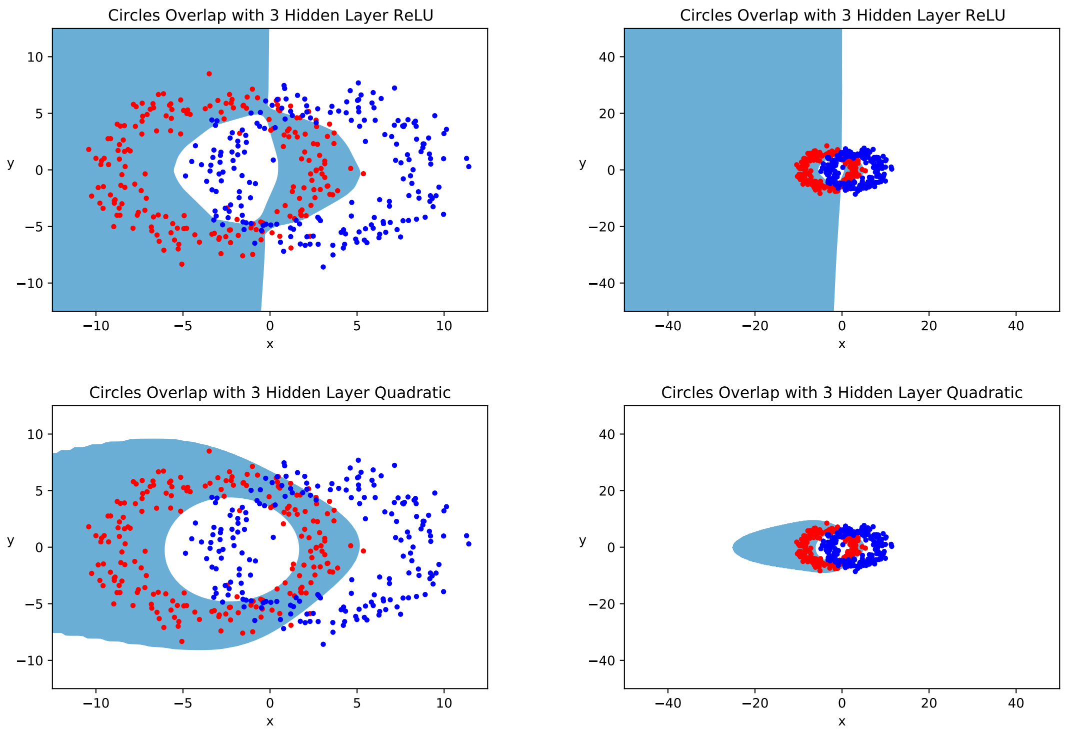


Figure 12: Binary Circles Overlap Classification with (Top) ReLU and (Bottom) Quadratic

in one group is simply implied to be in the other. However, Quadratic would likely be more optimal for an application where, perhaps, one wanted categorical classification, in which there are two main groups, and then an ”unknown” or ”misc.” group. Quadratic appears to learn the shape of the actual data of one group much better but simply then implies that everything not in the class must be in the other.

4.8 Experiment: Spirals

Another test that was used to test the future work by Ganesh and Singh was the Spirals dataset, since it does not fit into the conic section regions that Quadratic has been shown to perform well on. While there were initial difficulties which are explored later, the test was a success. At first glance, the more visible difference is the smoother boundary line that Quadratic is known to produce, because of the lack of linear parametrization. With that being said, though, ReLU apppeared to produce a more average plot in that the boundary line was more consistently equal between the two classification groups. Neither of these characteristics outweigh the other, but rather just provide different options for model designers.

What is interesting is the generalization of these activation functions to regions outside the trained one. Quite frankly, neither activation function performs better in this respect, as neither carries out the general pattern outside its known domain. With that being said, the same phenomenon that was seen in Circles Overlap was seen in Spirals; the Quadratic Activation Function tended to always formulate closed classification boundaries. While it may not always be visible in the trained domain, in the infinity of space, the ends meet.

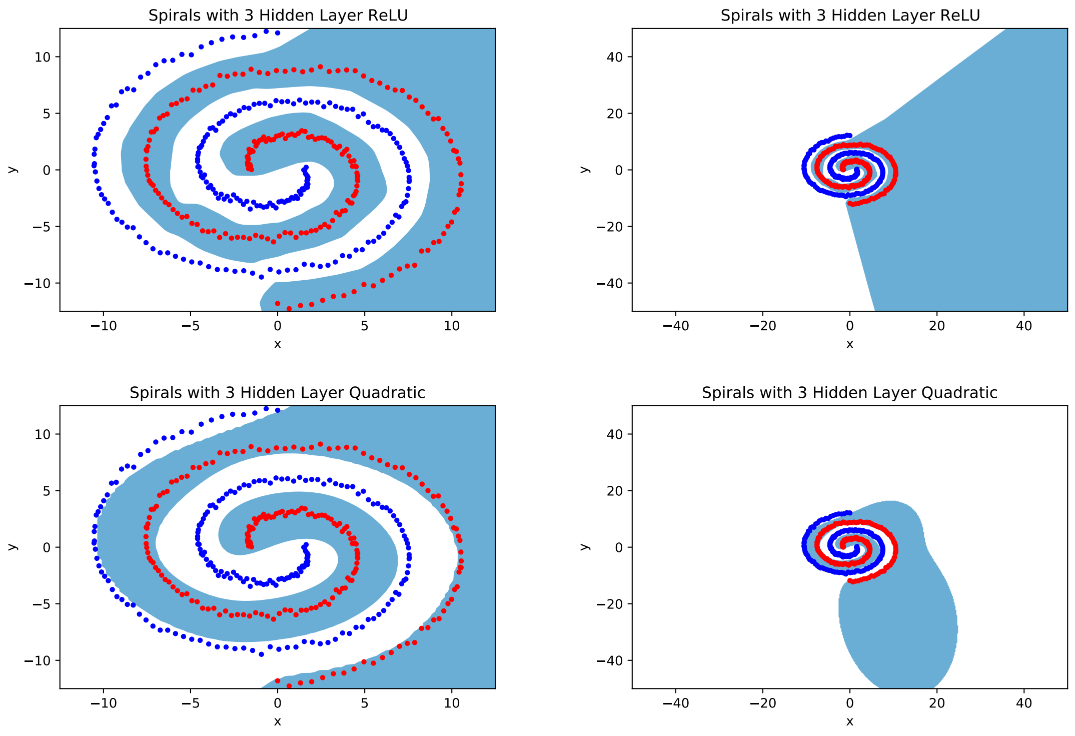


Figure 13: Binary Spirals Classification with (Top) ReLU and (Bottom) Quadratic

5 Normalization Techniques

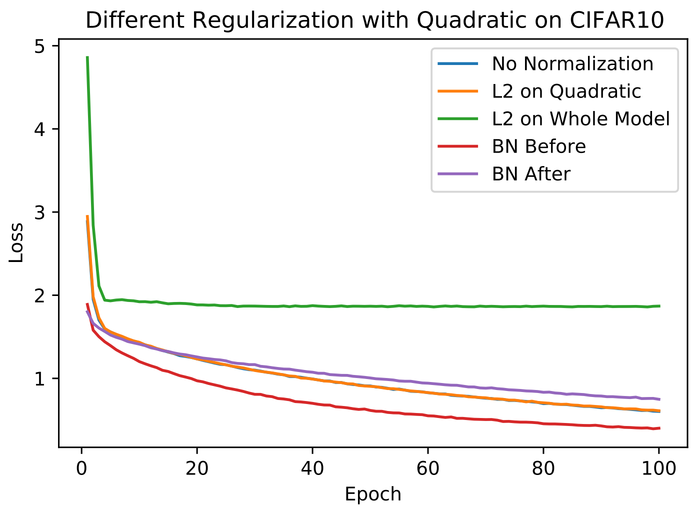


Figure 14: Loss vs. Epoch Graph with different normalization techniques

One of the first concerns that arose when attempting to implement this activation was how was regularization to be used and which types. When using a Quadratic function like Equation (1), there is a large potential for blow up, especially considering that the gradients of these functions are significantly larger than traditional activation functions. For this reason, L2 Norms and Batch Normalization were two schemes that were widely examined. In this test, one hidden layer was used on a complex dataset, like CIFAR10. This low complexity is to promote more overfitting and a general larger magnitude of parameters, seeing as parameters are more often than not positive.

In general, there was an interesting trend in which the more regularization that was applied, the poorer the model performed. Normalization was used in most previous experiments but they were not explicitly examined there. What has been a hot topic of debate is the application of normalization before or after activation functions, which is what this experiment was attempting to discover. As shown, when using a strong non-linearity it seemed that Batch Normalization prior to the Quadratic was most optimal for minimizing loss.

6 Initialization Methods

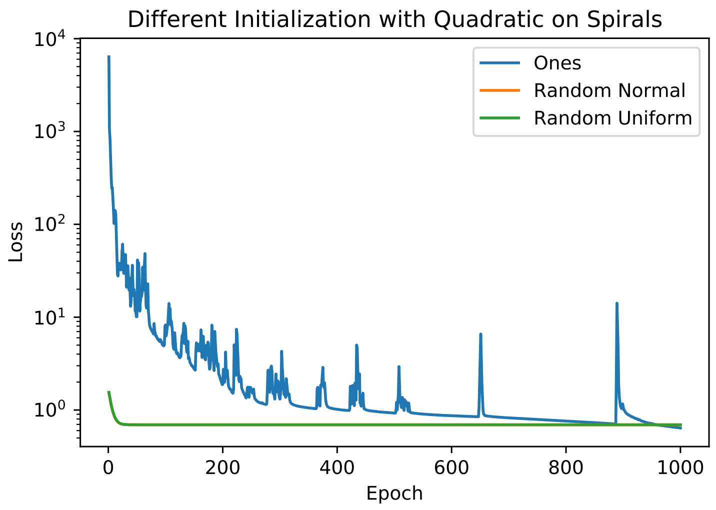


Figure 15: Loss vs. Epoch Graph with different initialization schemes

Something that was not immediately apparent which is often overlooked in complex deep neural networks is initialization. This was not originally a concern when designing models with this Quadratic Activation Function, however, it quickly became one once tests were conducted on Spirals. It was discovered that the initialization parameters of the coefficients on Quadratic proved to be a major influencing factor in the effectiveness of a model. This is shown in Figure 15. While it is hard to tell because of the use of a semi-log scale, when the initialization of the coefficients was set to a value less than one, the model refused to learn. Over thousands of epochs, it would fall into a local minimum in which losses converged and accuracies of fifty percent were consistent in binary classification. Even though the losses of these poor initialization schemes were lower, the test accuracy was considerably lower than when using an optimal initialization scheme. It is unknown whether this occurred because of the nature of the activation function or the nature of the dataset (not having naturally closed classification boundaries) it was applied to. This behavior was not noticed when testing other datasets such as MNIST and CIFAR10.

7 Conclusion

Overall, in relation to other activation functions, the Quadratic Activation Function is extremely effective in increasing final testing accuracy, but only when specific criteria are met due to the nature of the function. Since the function continuously halves the number of inputs, a limit is placed on the depth of the model. Furthering Ganesh and Singh's work, it was found that the Quadratic Activation Function can be applied to a variety of deep learning problems that range outside the initial tests. There are many opportunities for further exploration on this topic. The convolutional Quadratic activation could potentially be developed to alternate the axis that is halved, increasing the possible depth of the model. Studies could be done on the cause of the initialization errors and potential remedies to the issue. This paper served to introduce the concept and promote further research in the area of Quadratic Activation Functions.

References

[1] G. Yaparla, R. P. Singh, and G. Ramamurthy. *Pattern Classification using Quadratic Neuron: An Experimental Study*. Tech. rep. International Institute of Information Technology, 2017.