Binary Classification of Parabolic Data using Shallow Neural Networks

Warren Wu

June 6, 2025

1 Introduction

In this report, I will examine the efficacy of shallow neural networks with different numbers of neurons at a binary classification task where the classes are separated by a nonlinear decision boundary, specifically the parabola $X_2 = X_1^2$ (equivalent to $y = X^2$).

The idea for this project came from the Math 118 class, where my professor discussed the shortcomings of logistic regression at separating classes where the decision boundary is nonlinear, and that neural networks can perform this task more effectively. We also discussed the Universal Approximation Theorem[1], which demonstrates that any continuous function can be approximated by a feedforward neural network with one hidden layer and a sufficient number of neurons. It does not, however, say what this number of neurons is for a given function. I decided to take this idea further and examine how many neurons it would take for a shallow neural network (i.e. 1 hidden layer) to approximate nonlinear functions since it is theoretically possible.

Each neuron can be thought of as a hyperplane in R^2, which is a line. I decided to plot these hyperplanes that are constructed using the weights and biases from each individual neuron and see how the accuracy of the neural network changes as we increase the number of neurons (and hyperplanes).

My colab notebook is be linked in the citations and will also be submitted alongside this report.

2 Generating nonlinear data

To start with this project, I first generated random data from a 2-dimensional uniform distribution.

$$X = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Where $x_1, x_2 \sim Uniform(-1, 1)$

In order to make the two classes, $y = 0$ and $y = 1$, I assigned y above the curve to be $y = 1$ and values below the curve are assigned $y = 0$. This is the same as saying points above $x_1^2 - x_2 = 0$ belong to class $y = 0$ and $y = 1$ otherwise.

This is what my data X of 1000 points chosen looked like, along with the training and testing sets (split 80/20) which are subsets of X.

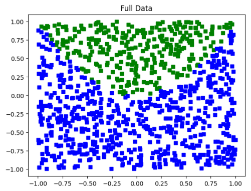

Figure 1: Full Data

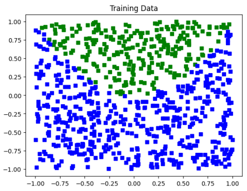

Figure 2: Training Data

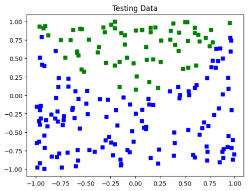

Figure 3: Testing Data

[1] Cybenko, G. Approximation by superpositions of a sigmoidal function. Math. Control Signal Systems 2, 303–314 (1989). https://doi.org/10.1007/BF02551274

3 Logistic Regression (and why it fails)

The logistic regression method seeks to minimize the following negative log likelihood equation:

$$L(f_{w,b}(x), y) = \sum_{n=1}^{N} -y_n ln(f_{w,b}(x_n)) - (1 - y_n)(1 - f_{w,b}(x_n))$$

Where:

- f is our model where w and b are weights and biases
- $f_{w,b}(x)$ is the predicted label given data point x
- y_n is the true label given data point x
- N is the total number of data points in the set of points

Importantly,

$$f_{w,b}(x) = \frac{1}{1 + e^{-(w^T x + b)}}$$

The decision boundary occurs where $f_{w,b}(x) = 0.5$

$$e^{-w^T x + b} = 1$$

$$w^T x + b = 0$$

We see here that, logistic regression models assume a decision boundary that is a hyperplane (linear in 2d space). This is why it fails to appropriately model our nonlinear decision boundary.

I trained a logistic regression model on the training data and plotted the decision boundary here against the testing data. Although the testing accuracy is relatively high (0.885) it does not approximate the theoretical decision boundary well at all.

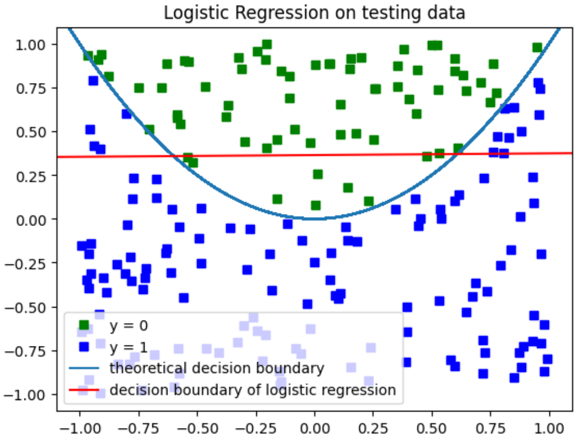

Figure 4: Logistic Regression Model

4 Feedforward Neural Network with 2 neurons

Next, I trained a shallow neural network (1 hidden layer) with two neurons. I used Binary Cross Entropy Loss and the Adam optimizer to perform backpropagation. I used 20000 epochs to train, which empirically was enough to reach the accuracy limit.

Each neuron corresponds to a hyperplane and this model therefore has more flexibility because the decision boundary is comprised of two hyperplanes.

The equation of the hidden layer is as follows:

$$h = \sigma(W_x X + b_x)$$

where W is the matrix of weights being applied to input layer X, and b is the vector of biases being applied to input layer X. σ is the Sigmoid activation function.

The i-th row in W (W_i) and b (b_i) are the weight and bias attributed to a singular hyperplane of the equation $W_i X + b_i = 0$). In this case, since there are two neurons in the hidden layer, two hyperplanes are created for the decision boundary.

This 2-neuron model produced the following hyperplanes:

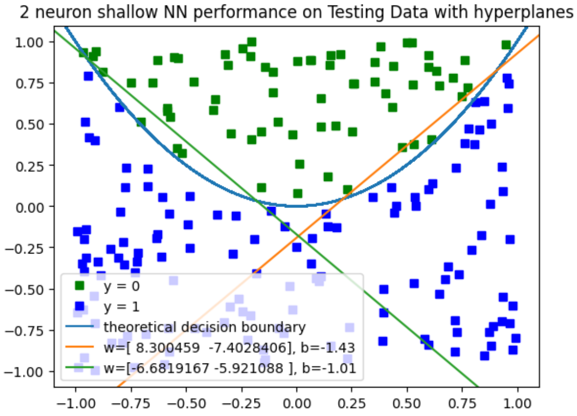

Figure 5: Model with 2 neurons

It is clear that this model is more flexible than the logistic regression model, as points above the intersection of the two hyperplanes are assigned $y = 0$ and points elsewhere are assigned $y = 1$. This model had an accuracy of 0.97 and followed the theoretical decision boundary much better than the logistic regression model.

5 Reaching 0.99 or 1 accuracy

Using these shallow neural networks, I wanted to investigate if it was possible to reach 0.99 or 1 accuracy in a nontrivial case (where the number of neurons $\ll$ the size of the input data).

To investigate this, I trained shallow neural networks from 3 to 10 neurons using the same loss function (Binary Cross Entropy), optimizer (Adam), and number of epochs (20000).

I got the following accuracies:

$$[0.995, 0.995, 0.995, 0.995, 0.995, 0.995, 1.0]$$

Models with 3-8 neurons reach a cap of 0.995 accuracy (only 1 point from the testing data incorrectly labeled out of 200), but 9 neurons reaches 1.0 accuracy.

6 Conclusion

The universal approximation theorem says that it's theoretically possible to approximate this parabola using a shallow neural network. For this experiment, I trained my models on 800 training points and tested on 200 testing points.

I achieved 0.885 testing accuracy using logistic regression, which produced one hyperplane. This model did not approximate the decision boundary parabola well at all.

I achieved 0.97 testing accuracy using two neurons, which produced two hyperplanes. This model sufficiently approximated the decision boundary parabola.

Models with 3-8 neurons and 3-8 hyperplanes achieved 0.995 testing accuracy. Lastly, the model with 9 neurons and 9 hyperplanes achieved 1.0 testing accuracy.

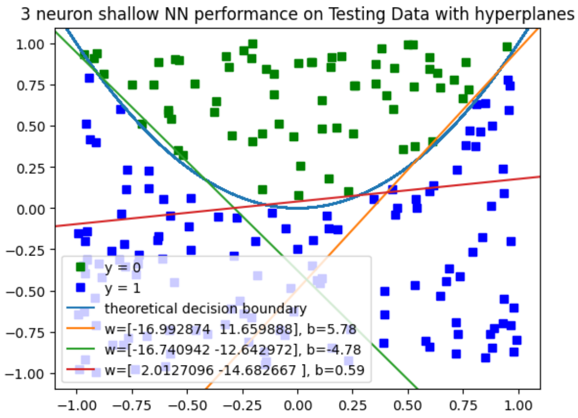

Figure 6: 3 neuron neural network: 0.995 accuracy

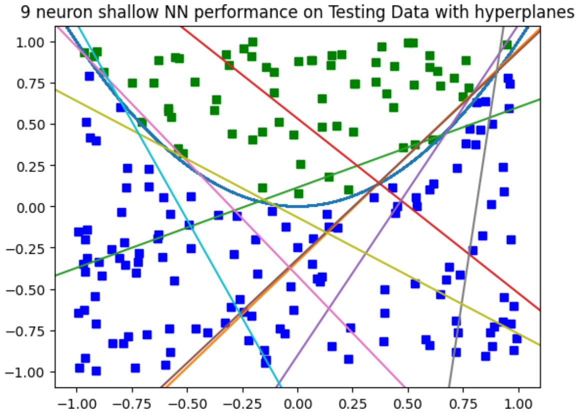

Figure 7: 9 neuron neural network: 1.0 accuracy

Given more time and computing power I would expand the number of points of the data in this experiment and expand the number of dimensions of the input data. I predict that the number of neurons needed to achieve the same degree of accuracy for the 2-dimensional input data will be much higher and therefore much more computationally expensive. I would also continue to investigate how the number of training/testing points is correlated with the number of neurons needed to achieve near-perfect accuracy.

Ultimately, I conclude that with the data generated for this experiment, 2 hyperplanes is sufficient to approximate the parabola decision boundary to a very high degree. With more hyperplanes, it is possible to achieve higher accuracy and better approximate the decision boundary, but this is not guaranteed with different data and is not necessarily ”worth it” considering the computational power needed to do so.

7 Citations

https://docs.pytorch.org/tutorials/beginner/basics/buildmodel_tutorial.html

Cybenko, G. Approximation by superpositions of a sigmoidal function. Math. Control Signal Systems 2, 303–314 (1989). https://doi.org/10.1007/BF02551274

My Colab Notebook