# Project 3: Multilayer Perceptron Neural Network

Steven M. Hernandez
Department of Computer Science
Virginia Commonwealth University
Richmond, VA USA
hernandezsm@vcu.edu

***Abstract*— Multilayer Perceptron Neural Networks quite simply takes our previously explained definition of Perceptron Learning further by stringing a network of multiple perceptron neurons into one larger model which is thus able to handle more complex relationships within datasets.**

## I. INTRODUCTION

Multilayer Perceptron Neural Networks quite simply takes our previously explained definition of perceptron Learning further by stringing a network of multiple perceptron neurons into one larger model which is thus able to handle more complex relationships within datasets.

Specifically in this case, we look to have two layers of perceptron neurons; one hidden layer using tangent hyperbolic activation function followed by one output layer with linear activation function. This output layer contains only a single neuron in this case simply because the dataset provided contains only two classes.

### A. Data Exploration

To begin, we explore the given **cancer** dataset. Figure 1 shows the input values for the given dataset revealing two of the 32 dimensions have unproportionaly high values. These high values would likely skew our learnings, thus we normalize all dimensions such that each dimension follows a normal distribution as can be seen in fig. 2.

Additionally, fig. 3 shows the expected outputs for the learning where class 2 from the original dataset is positive $+1$ and 1 from the original dataset is negative $-1$. From this, it is apparent a higher number of negative elements exist within the dataset when compared to positive elements.

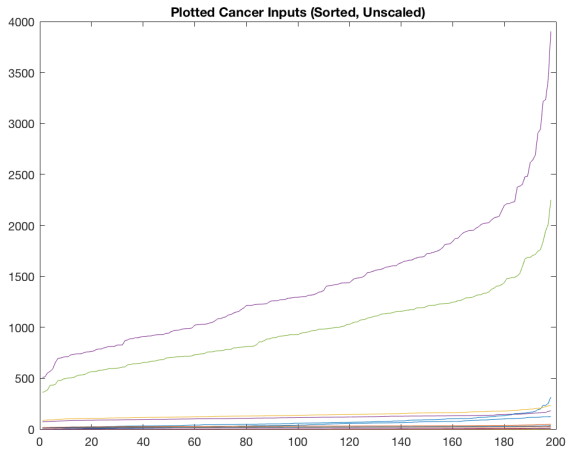


Fig. 1. Input values from the **cancer** dataset sort revealing 2 out of the 32 dimensions have unproportionally high values, which would likely skew the learning results.

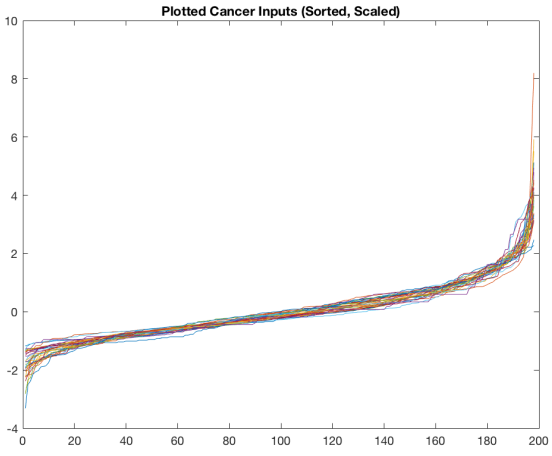


Fig. 2. Input values from the **cancer** dataset sorted after normalizing each dimension such that each value falls on a normal distribution.

## II. RESULTS

For learning, we tested percent error for when applying different parameters for both number of neurons in the hidden layer as well as the number of learning epochs completed. For experiment, we tried the following number of hidden layer neurons $N = \{5, 10, 15, 25, 50, 75, 100\}$ and the following number of learning epochs $I = \{100, 250, 500, 1000, 1500\}$.

### A. Calculating Error Signal without the sign function

Because we are using our model to identify classes, it may be considered that we should apply the **sign** function when calculating the error and error signal for the model. This operation is not specified in the definition of the Multilayer Perceptron Neural Networks from Box 4.1a in [1] and as

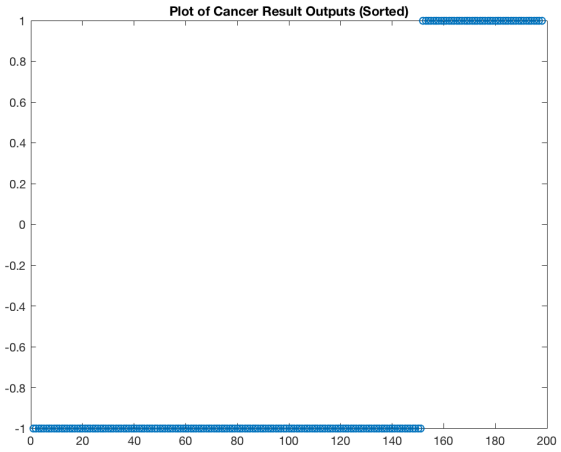


Fig. 3. Sorted outputs for **cancer** dataset showing a larger number of negative class elements when compared to the number of positive class elements.

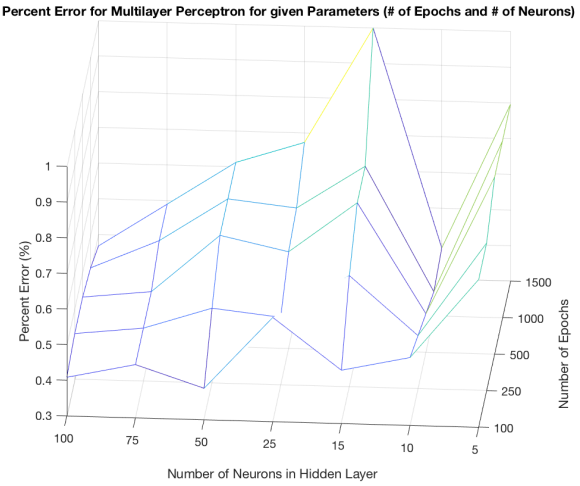


Fig. 4. 3D Plot showing the relation between different number of epochs and different numbers of hidden layer neurons on the percent error produced by our model on our testing dataset.

discussed later may contradict the model definition itself. So first, we look at following the model definition more exactly. In the following section, we will look at the implementation and the effects of using the **sign** function.

Table I shows the result of the model after learning on the training set and fig. 4 presents the same data visualized into a 3D plot for more convenient understanding of the data. The table present the percent error achieved when the model predicts the class on the testing dataset. We can see that the best case is when number of hidden layer neurons $n = 10$ and number of learning epochs $i = 1000$. We can note that as we have a higher number of epochs such as $i = 1500$, we actually achieve a worst percent error on the testing dataset. The data

Using these best parameters, we achieve a $50.0\%$ error when classifying the positive class and an error of only $14.3\%$ error when classifying the negative class.

TABLE I

PERCENT ERROR RESULTS FOR GIVEN # OF LEARNING EPOCHS (COLUMNS) AND # OF NEURONS (ROWS) ON THE **TESTING** DATASET WHEN CALCULATING ERROR **WITHOUT THE SIGN FUNCTION**

| | 100 | 250 | 500 | 1000 | 1500 |
|---|---|---|---|---|---|
| 5 | 0.7143 | 0.7143 | 0.7959 | 0.7959 | 0.7959 |
| 10 | 0.4898 | 0.4490 | 0.4082 | **0.3673** | 0.3878 |
| 15 | 0.4490 | 0.6122 | 0.7143 | 0.7143 | 1.0000 |
| 25 | 0.5918 | 0.4898 | 0.5714 | 0.5918 | 0.6735 |
| 50 | 0.3878 | 0.5102 | 0.6122 | 0.6122 | 0.6122 |
| 75 | 0.4490 | 0.4490 | 0.4490 | 0.4898 | 0.4898 |
| 100 | 0.4082 | 0.4286 | 0.4286 | 0.4082 | 0.3673 |

### *B. Calculating Error Signal with the sign function*

As mentioned above, because the model is used for classification, it may make sense to calculate error signal by applying the **sign** function to the output of the network. In this section we evaluate this approach.

We use the **sign** function when calculating the output layer neuron error signal $\delta_{okp}$. If the sign value of $sign(o_{kp}) = d_{kp}$ this indicates there was no error or $e_{kp} = 0$ and thus $\delta_{okp} = 0$ meaning the model does not require updating. Beyond this application of **sign**, it is also used when calculating error across the final error on the trained model.

Issues with this method are understanding applying **sign** to $o_{kp} = f_o(u_{kp}$ may imply that $f_o(x) = sign(x)$. This may be find, however later on in our definition, we must take the derivative of $f_o()$, $f'_o()$ in calculating $\delta_{okp}$. In this case, we consider $f_o(x) \neq sign(x)$, instead allowing $f_o(x)$ to be a derivable linear function and the application of the **sign** function simply being an additional step in calculating $\delta_{okp}$.

Results of this method are shown in table II and visually presented in fig. 5 showing that the best error percentage of $26.53\%$ with number of neurons $n = 10$ and number of epochs $i = 1000$.

TABLE II

PERCENT ERROR RESULTS FOR GIVEN # OF LEARNING EPOCHS (COLUMNS) AND # OF NEURONS (ROWS) ON THE **TESTING** DATASET WHEN CALCULATING ERROR WITH THE **SIGN** FUNCTION

| | 100 | 250 | 500 | 1000 | 1500 |
|---|---|---|---|---|---|
| 5 | 0.3265 | 0.3265 | 0.2857 | 0.3061 | 0.3061 |
| 10 | 0.4694 | 0.4490 | 0.3878 | **0.2653** | 0.2653 |
| 15 | 0.3673 | 0.4082 | 0.4082 | 0.2653 | 0.2653 |
| 25 | 0.3265 | 0.3265 | 0.3265 | 0.3265 | 0.3265 |
| 50 | 0.4082 | 0.3469 | 0.3469 | 0.3469 | 0.3469 |
| 75 | 0.3878 | 0.3878 | 0.3878 | 0.3878 | 0.3878 |
| 100 | 0.3469 | 0.3469 | 0.3469 | 0.3469 | 0.3469 |

Table III shows the percent error on the training dataset calculated by the model with each parameter. As we can see, with the model quickly reaches $0\%$ error on the training dataset which explains why for example with 100 neurons, the whole row of table II contains the same error, the model

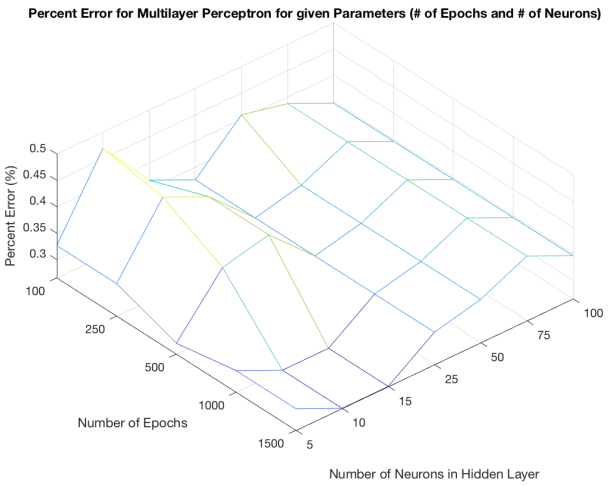


Fig. 5. 3D Plot showing the relation between different number of epochs and different numbers of hidden layer neurons on the percent error produced by our model on our testing dataset.

learned all there was to learn in less than 100 epochs, thus no changes were made as more epochs were given.

With these parameters, the results in error percentage for the positive class is $57.1\%$ and the error percentage for the negative class is $14.3\%$. We can see without sign, we surprisingly have a better results for the positive class. This could be attributed to the model reaching a stopping point when learning with the sign function applied, but when no sign function is used in calculating the error signal, the model can continue to better optimize the weights of the hidden and output layers.

TABLE III

PERCENT ERROR RESULTS FOR GIVEN # OF LEARNING EPOCHS (COLUMNS) AND # OF NEURONS (ROWS) ON THE **TRAINING** DATASET WHEN CALCULATING ERROR WITH THE **SIGN** FUNCTION

| | 100 | 250 | 500 | 1000 | 1500 |
|---|---|---|---|---|---|
| 5 | 0.0671 | 0.0336 | 0.0940 | 0 | 0 |
| 10 | 0.0067 | 0.0067 | 0.0067 | 0 | 0 |
| 15 | 0.0134 | 0.0067 | 0.0067 | 0 | 0 |
| 25 | 0 | 0 | 0 | 0 | 0 |
| 50 | 0.0134 | 0 | 0 | 0 | 0 |
| 75 | 0 | 0 | 0 | 0 | 0 |
| 100 | 0 | 0 | 0 | 0 | 0 |

## III. CONCLUSION

This report presents the usage of Multilayer Perceptron Neural Network learning on cancer data revealing with 10 hidden layer neurons and 1000 training epochs, our best percent error achieved was $26.53\%$ across our training dataset. For each class, the best model produces a $50.0\%$ error on classifying positive classes and an error of only $14.3\%$ when classifying the negative class.

Moving forward, it may be useful to also use cross validation along with testing different values of $\eta$ which for this report we had kept static at $\eta = 0.001$.

## REFERENCES

[1] V. Kecman (2001). Learning and Soft Computing. Mit Press, p.255-311.