Approximating Sigmoid Activation with Piecewise Linear Functions

Bilal Ahmad

September 30, 2019

1 Problem

1.1 Setting Up a Feed Forward Neural Network

For this exercise, I primarily used code from the Machine Learning Mastery website uploaded by Dr Jason Brownlee. The code was extended in some areas to account for my dataset and the goals of this exercise. The feed forward network code used for this exercise comprises of initializing the network weights, forward propagating the input values, back propagating the error, and training the network.

1.2 Piecewise Approximation of Sigmoid Function

Sigmoids are favored for activation functions because they saturate and are continuous throughout. The continuity is important because the back propagation algorithm relies on the derivative of the activation function to back propagate the error. Sigmoids can be expensive to implement in hardware and thus it is sometimes beneficial to use simpler approximations. For this exercise we are approximating the sigmoid function with piecewise linear functions.

We approximate the sigmoid function by splitting it into segments (n = 6, 8, 10, 12 ,14). I split the sigmoid function along the x-axis in even segments and calculated the slope between the segments. Then, knowing the slope, I calculated the y-intercepts to get the line equations of each segment. I put the piecewise information in four arrays that I later used for the piecewise transfer function. This is not the most accurate way to approximate the sigmoid function for a given number of steps but I wanted to implement a way that was consistent and easy to follow [2]. I considered everything from -5 to 5 on the x-axis of a sigmoid plot to be the saturation zone so the functions take this into account. Below is the segmentation code.

```
# Values for the piecewise functions
points = list()
slopes = list()
intercepts = list()
y_val = list()

# Split the sigmoid function into the desired amount of segments
def segment_sigmoid (n_segments):
    width = 10.0/(n_segments - 2)
    x_val = -5.0
    prev_sig_val = 0.0
    for i in range(n_segments - 2):
        next_step = x_val + width
        sig_val = 1.0 / (1.0 + exp(-1 * next_step))

        points.append(next_step)
        slopes.append( (sig_val - prev_sig_val) / (next_step - x_val))
        intercepts.append(sig_val - (slopes[-1] * next_step))
        y_val.append(sig_val)

        x_val = next_step
        prev_sig_val = sig_val
```

With the equations for the piecewise linear functions, it is easy to implement the transfer derivative since each piece has a constant slope. The transfer derivatives of the piecewise functions are used when back propagation occurs with the piecewise functions.

1.3 Dataset

For this exercise, I used a dataset from the University of California Irvine's public machine learning dataset [3]. The data set I used has 10 attributes (9 inputs and one expected classification). The dataset consisted of information about men (alcohol consumption, age, etc) and made a classification about whether the men had a normal or altered fertility diagnosis. Some of the attributes are non-numerical so they were converted into enumerations and given a numerical value.

2 Results

2.1 Error After 5000 Epochs

I ran each network for 5000 epochs using the same initial network and learning rate of 0.15. I chose the learning rate after a little trial and error and kept it fixed because I wanted the number of segments to be the only differentiating factor between the piecewise approximations. After 5000 epochs, the mean squared error for each network is shown in Table 1 .

	N = 6	**N = 8**	**N = 10**	**N = 12**	**N = 14**	**Sigmoid**
Error	0.041	0.040	0.041	0.040	0.040	0.008

Table 1: Final Mean Squared Error After 5000 Epochs

The sigmoid function has the lowest final error and the approximations all have the same approximate error. This is a little misleading though because the error for a network doesn't continuously decrease over the duration of training. As seen in Figure 1, there are local min and max error values that occur during the network training. In addition, the errors taper towards an asymptote so it may be more valuable instead to see how many epochs it takes for a network to reach a desired error value instead of what the last error value is when training is complete.

2.2 Epochs to MSE Less Than .042

Table 2 shows the number of epochs elapsed until the network reaches a mean square error of less than .042. I chose this error value because it is near where all the piecewise approximations taper off. As expected, the sigmoid function is the fastest to reach the desired error. In addition, the approximations with more segments are faster at reaching the desired error.

	N = 6	**N = 8**	**N = 10**	**N = 12**	**N = 14**	**Sigmoid**
Epochs	4651	2805	2476	2332	2268	1507

Table 2: Epochs Elapsed Until Error Less Than .042

3 Conclusion

From this exercise we see that there may be times when it is more appropriate to implement sigmoid approximations in place of sigmoid activation functions. The approximations may save expenses on the hardware side while still providing acceptable results. It should be noted however there is a diminished amount of return on error mitigation by continuing to further segment the sigmoid function so this will have to be taken into account when trying to determine the most suitable approximation for a problem [2]. For further study into linear approximation, I would test using recursive formulas to better fit the piecewise segments onto the sigmoid function and use an adaptive learning rate to better train the networks to see how that affects the benefits of approximation.

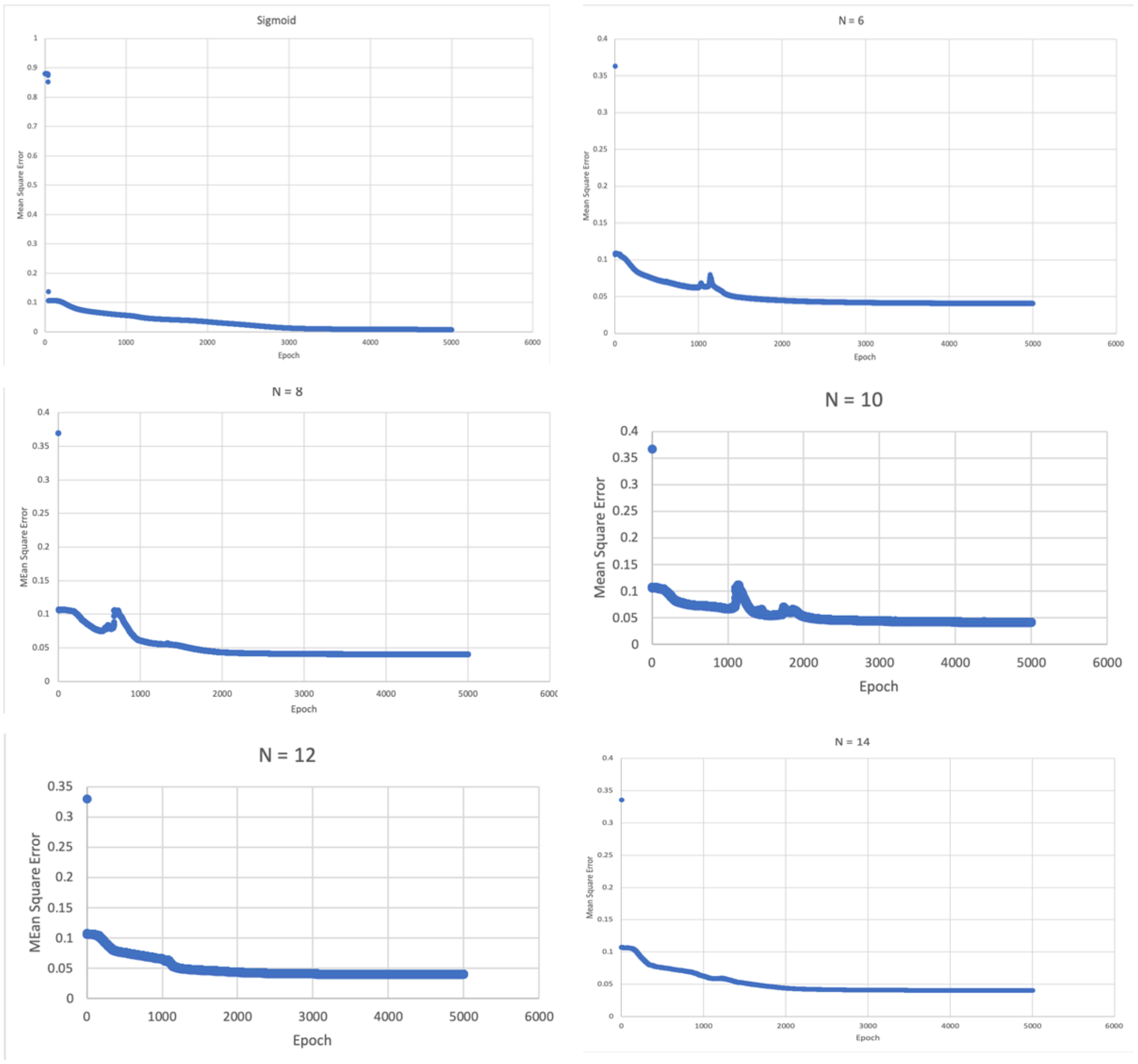

Figure 1: MSE vs Epoch for Each Network

4 References

[1] J. Brownlee How to Code a Neural Network with Backpropagation In Python

[2] K. Basterretxea, E. Alonso, J. M. Tarela, I. del Campo, PWL Approximation of Non-linear, Functions for the Implementation of NeuroFuzzy Systems, Proceedings of the IMACS/IEEE CSCC'99, Athens 1999.

[3] David Gil, Jose Luis Girela, Joaquin De Juan, M. Jose Gomez-Torres, and Magnus Johnsson. Predicting seminal quality with artificial intelligence methods. Expert Systems with Applications