Connectionist Computing

CARLO FINNEGAN

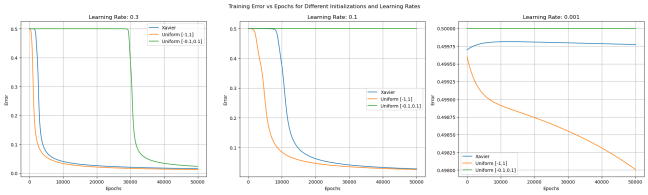

FIGURE 1: Weight Initializations for XOR Problem

I. INITIALIZING WEIGHTS

My first point of concern for my network was weights initialization as initially messing around with ranges I found starkly different results some converging much better then others. Therefore I took a look into methods to effectively initialize weights for the models. A simple approach to initialize weights randomly was to do so within a small range between minus 0.1 and 0.1. I started weight like this but quickly discovered although setting the weights between small range like this is a useful start in many cases it can sometimes cause an unnecessary slow convergence. For example XOR is a very simple problem but if the network isn't able to push outputs away from the 0.5 value it will struggle to learn and converge as seen in figure **??** where we can see there is no gradient evident in the case of the small weight range. After experiencing this I quickly found greatly increasing my learning rate could account for this or changing the range at which my weights would initialize to - 1 and 1 at would lead to much faster convergence. However I wanted to find a more concrete way to initialize the weights. This is when I came across Xavier Weight Initialization. Which is way to set the range to which we can sample out initial weights from in this case uniformly.

The Xavier weight initialization scale factor is given by:

$$\text{Scale} = \sqrt{\frac{2}{\text{hiddenSize} + \text{outputSize}}} \tag{1}$$

This is a well documented method of initialization which typically provides a stable sets of weights that are free from risk that are present when just taking a generic range. From looking at the chart below its clear that both a range of -1 and 1 and the range set by the Xavier initialization provide acceptable results across different learning rates. The initialization that performed the best constantly switched on different runs due to the random weights selection. I settled on using Xavier initialization as its reliability and also performance were acceptable for both my XOR, SINE and Letter Recognition problems.

For my digit recognition model, I implemented a similar approach to weight initialization but instead had to opt for

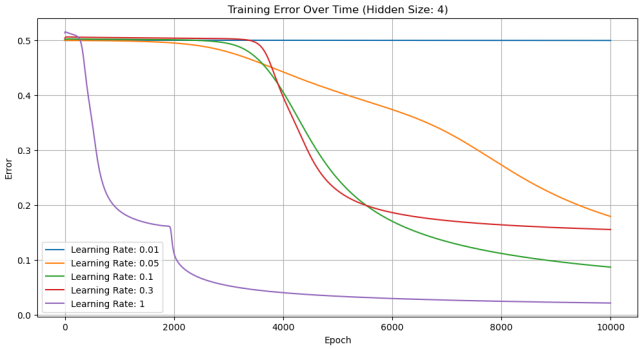

FIGURE 2: XOR Problem: Training Error Over Time with 4 Hidden Neurons

"He initialization" for the ReLU activation layers. This is very similar to Xavier but was specifically engineered for RELU activation functions. It was important I didnt use a range of values such as -0.1 to 0.1 as by nature the RELU function would effectively kill half the neurons immediately as all negative values would get set to 0 causing a much slower start towards converging. I couldn't use Xavier initialization as it assumes a symmetric activation function. However I used it for the second layer of that model which used the softmax function. These weight initializations were the baseline for all my further model results as I found them to be the best performers.

The He weight initialization scale factor is given by:

$$\text{Scale} = \sqrt{\frac{2}{\text{inputSize}}} \tag{2}$$

II. MODELS

1) Model 1 (XOR Problem)

For the XOR problem, my activation function of choice sigmoid was natural since we're dealing with binary classification where the outputs needed to be squashed between 0 and 1. Uses sigmoid activation functions in both hidden and output layers, with mean absolute error loss. I experimented with a number of different hidden layers and learning rates to see what converged well and what didn't. The outputs can be seen in table 1 and in the plots 2 ,3 and 4

The XOR results demonstrate how learning rate significantly impacts networks convergence and performance. Using higher learning rates (0.3, 1.0) led to faster convergence. Where as small learning rates seemed to struggle with convergence much more with 0.01 not really converging at all. We can see the pitfalls that normally occur with large learning rates aren't present for this XOR problem likely due to the simple binary nature of the problem and small input space(of 4 values). We can see in this case the network can make large

TABLE 1: XOR Problem: Training Error Rates at Different Epochs for Various Hidden Sizes and Learning Rates

Hidden Size	Learning Rate	1000 Epochs	2000 Epochs	5000 Epochs	10000 Epochs
4	0.01	0.499079	0.499075	0.499066	0.498597
4	0.05	0.500300	0.499793	0.463937	0.334100
4	0.10	0.494406	0.439850	0.211260	0.078353
4	0.30	0.503300	0.294315	0.065032	0.033336
4	1.00	0.083309	0.046493	0.025943	0.017553
8	0.01	0.499589	0.499501	0.499115	0.497530
8	0.05	0.501480	0.501365	0.499385	0.364615
8	0.10	0.503099	0.502827	0.452172	0.103911
8	0.30	0.416010	0.149596	0.059427	0.037082
8	1.00	0.200186	0.051505	0.023710	0.014694
16	0.01	0.499949	0.499813	0.499269	0.497310
16	0.05	0.496479	0.471033	0.328718	0.154979
16	0.10	0.506583	0.505937	0.468485	0.103167
16	0.30	0.451711	0.164022	0.059337	0.036376
16	1.00	0.065220	0.038340	0.021608	0.014560

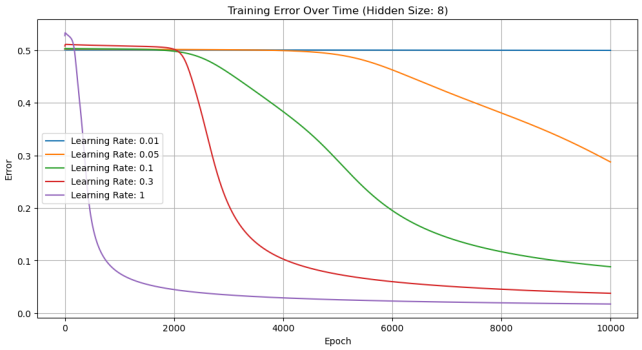

FIGURE 3: XOR Problem: Training Error Over Time with 8 Hidden Neurons

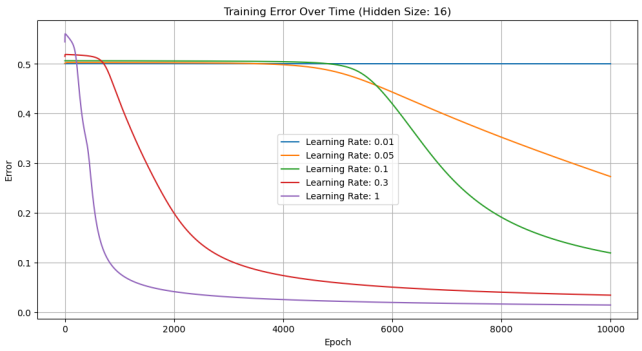

FIGURE 4: XOR Problem: Training Error Over Time with 16 Hidden Neurons

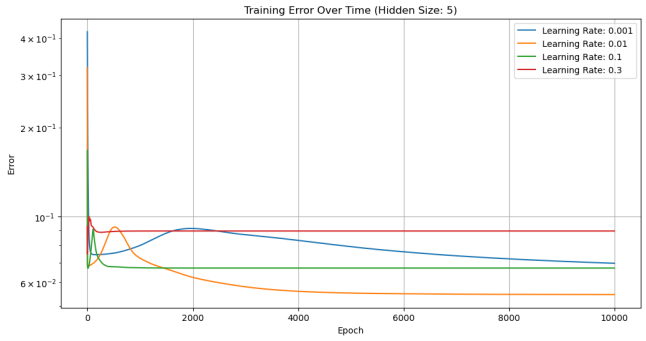

FIGURE 5: SINE Problem: Training Error Over Time with 5 Hidden Neurons

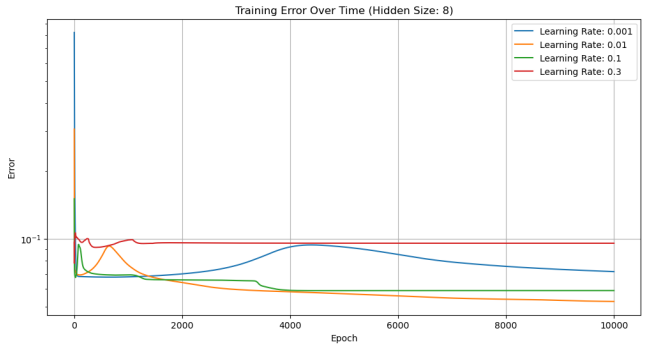

FIGURE 6: SINE Problem: Training Error Over Time with 8 Hidden Neurons

parameter updates without getting stuck in local minima or causing divergence(can be seen from the consistent and constant decrease in error).

2) Model 2 (Sine Function)

The sine function approximation is different, we have four input values between -1 and 1 to predict

$$\sin(x_1 - x_2 + x_3 - x_4)$$

For this, tanh was the ideal activation function because it outputs values in the same range as a sine function within (-1 to 1) and the tan function shares similar symmetrical properties around zero. The outputs can be seen in table 2 and in the plots 5 ,6,7, 8 and 9.

The SINE problem results reveal significantly different training dynamics compared to the XOR problem, particularly in terms of optimal learning rates and convergence patterns. While the XOR problem benefited from higher learning rates, the SINE problem demonstrates superior performance with moderate learning rates, specifically 0.01 The highest learning rate of 0.3, which performed well in the XOR problem, consistently showed the poorest performance in the SINE problem. This difference can be attributed to the fundamental nature of the problem as a SINE function is continuous and nonlinear makes it a more complex problem

TABLE 2: SINE Problem: Error Rates at Different Epochs for Various Hidden Sizes and Learning Rates

Hidden Size	Learning Rate	1000	2000	5000	10000
5	0.001000	0.079979	0.091271	0.079244	0.069648
5	0.010000	0.072487	0.062437	0.055265	0.054641
5	0.100000	0.067249	0.067119	0.067116	0.067116
5	0.300000	0.089481	0.089500	0.089500	0.089500
8	0.001000	0.067867	0.069953	0.092047	0.071598
8	0.010000	0.076634	0.064051	0.057100	0.052811
8	0.100000	0.069299	0.065807	0.058982	0.059024
8	0.300000	0.098856	0.096082	0.095682	0.095655
12	0.001000	0.064558	0.064257	0.065082	0.084117
12	0.010000	0.080618	0.058715	0.051930	0.050963
12	0.100000	0.068479	0.068261	0.068621	0.068643
12	0.300000	0.091508	0.091679	0.091709	0.091709
20	0.001000	0.064765	0.064631	0.066163	0.076429
20	0.010000	0.073458	0.054675	0.048936	0.047652
20	0.100000	0.055532	0.053759	0.053694	0.053687
20	0.300000	0.091260	0.091554	0.091558	0.091483
50	0.001000	0.065036	0.065034	0.065924	0.089140
50	0.010000	0.063067	0.065455	0.053054	0.049243
50	0.100000	0.055556	0.053466	0.053931	0.054575
50	0.300000	0.095516	0.097151	0.097185	0.097185

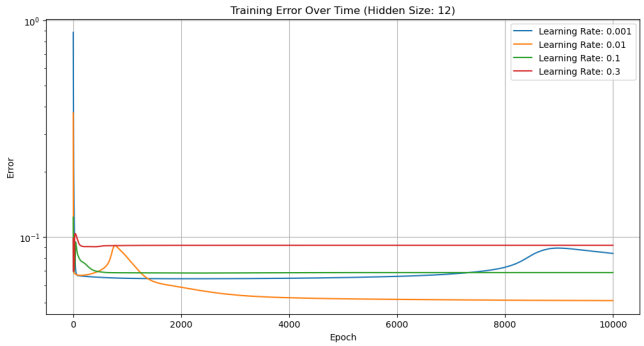

FIGURE 7: SINE Problem: Training Error Over Time with 12 Hidden Neurons

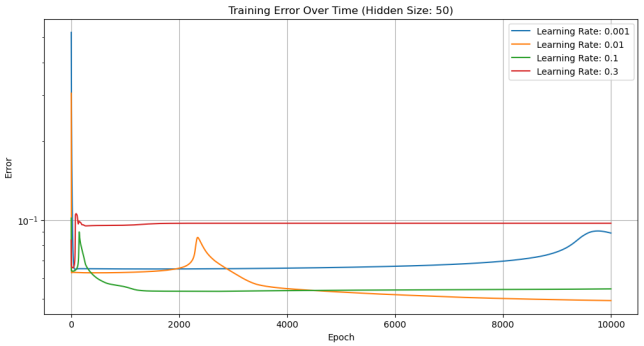

FIGURE 9: SINE Problem: Training Error Over Time with 50 Hidden Neurons

FIGURE 8: SINE Problem: Training Error Over Time with 20 Hidden Neurons

that requires more careful navigation during training. The high learning rate appears to cause the network to converge prematurely to suboptimal solutions, likely in local minima due to taking steps that are too large.

Whats interesting is in all cases there is a rise in error again before it reduces again. This is likely due to network capturing initially a good pattern but for it to improve beyond that and cover the full sine wave it needs to adjust its weights which temporarily result in an increase in error. This phenomenon can be seen in all occurrence of training.

We can see from the tables and plots that an increase in hidden size doesn't really result in significant performance improvements and also leads to a longer time for the model to converge. However a hidden size of 20 gave the best results so the added complexity does provide some benefit.

3) Model 3 (Letter Recognition)

In my third model we're dealing with a more complex pattern recognition problem where the input is a 16-dimensional vector representing letter features, and we need to classify it into one of 26 possible letters. My network needs to identify values for the vector of features provided. For my hidden layer I chose ReLU because it allows for selective activation eg when my inputs are negative the neurons output zero essentially turning off and creating a form of feature selection where only the most relevant neurons for identifying specific letter features remain activated. I set up the output layer to have 26 neurons, one for each letter of the alphabet. I used one-hot encoding create a vector of 26 probabilities to represent the output. I used a softmax activation for this layer because it takes my network's raw scores and turns them into

FIGURE 10: Letter Recognition: Training Error Over Time with 10 Hidden Neurons

FIGURE 11: Letter Recognition: Training Error Over Time with 30 Hidden Neurons

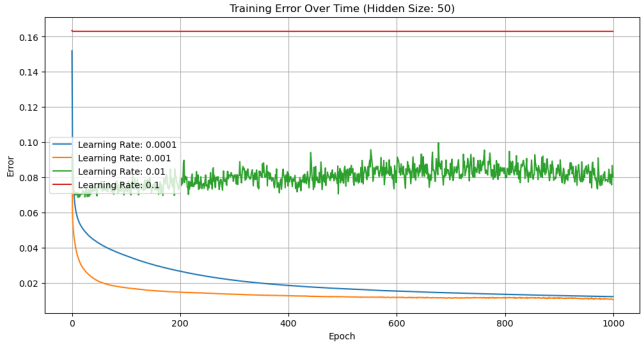

FIGURE 12: Letter Recognition: Training Error Over Time with 50 Hidden Neurons

probabilities that add up to 1. This way, each output tells us how confident the network is that the input represents that particular letter, and we can easily pick the most likely letter by looking at which output has the highest probability.

The digit recognition results demonstrate a clear relationship between network size, learning rate, and model performance, with some particularly interesting patterns . Most notably, the combination of larger networks and smaller learning rates consistently achieved the best performance, with the most impressive results coming from a network with 100 hidden neurons and a learning rate of 0.001, achieving a remarkably low final error.

Looking at learning rate patterns, we can observe a contrast in performance across different rates. The highest learning rate of 0.1 performed universally poorly, resulting in immediate convergence to a suboptimal error of around 0.16 across all network sizes. This identical error across configurations suggests the high learning rate caused the network to become stuck in the same local minimum regardless of network capacity, indicating the steps taken during training were too large for effective learning. Similarly poor results can be seen for 0.01 however it converge more so but then gets stuck within a range and oscillates between that range and also fails to converge.

FIGURE 13: Letter Recognition: Training Error Over Time with 100 Hidden Neurons

The smaller learning rates (0.001 and 0.0001) demonstrated substantially better performance across all network sizes, with their advantage becoming more pronounced in larger networks. This is particularly evident in the 100 neuron configurations, where both rates showed consistent improvement over time, with 0.001 achieving the best overall performance over 1000 epochs. However as it gets to the end of the epochs we can see the smaller learning rate of 0.0001 beginning to catch it. Suggesting likely if we run my model long enough the smaller learning rate will surpass.

In general as the network got larger so did my models performance suggesting the added neurons allowed us to capture more complex patterns.

III. MODEL RESULTS

1) Model 1 (XOR Problem)

The XOR model demonstrates excellent performance, achieving outputs that closely approximate the ideal binary values. For input combinations [0,0] and [1,1], the model predicts values very close to 0 (0.0399 and 0.0275 respectively), while for [0,1] and [1,0] it predicts values near 1 (0.9646 and 0.9665). These results show the neural network has successfully learned the nonlinear XOR relationship, effectively separating the input space into the correct regions.

2) Model 2 (Sine Function)

The sine function model shows poor results in its predictive capabilities. With only 11.45% of predictions falling within 0.1 of the true values, and lower accuracies at finer thresholds (0.18% within 0.001 and 0.02% within 0.0001). The model

TABLE 3: Letter Recognition: Error Rates at Different Epochs for Various Hidden Sizes and Learning Rates

Hidden Size	Learning Rate	100	200	500	1000
10	0.000100	0.064744	0.055789	0.052621	0.050666
10	0.001000	0.058355	0.057548	0.057413	0.057126
10	0.010000	0.135031	0.135430	0.136310	0.143029
10	0.100000	0.163024	0.163024	0.163024	0.163024
30	0.000100	0.043110	0.036737	0.028251	0.025653
30	0.001000	0.028285	0.024949	0.022963	0.021880
30	0.010000	0.103355	0.103261	0.103933	0.109155
30	0.100000	0.163024	0.163024	0.163024	0.163024
50	0.000100	0.035495	0.026626	0.016620	0.012114
50	0.001000	0.017197	0.014694	0.012133	0.010533
50	0.010000	0.074295	0.078940	0.078228	0.080252
50	0.100000	0.163024	0.163024	0.163024	0.163024
100	0.000100	0.028190	0.019444	0.011460	0.007825
100	0.001000	0.007459	0.004653	0.001475	0.000305
100	0.010000	0.065967	0.073079	0.078127	0.065003
100	0.100000	0.163024	0.163024	0.163024	0.163024

TABLE 4: XOR Function Accuracy Results

Input	Predicted Output
[0 0]	0.0399
[0 1]	0.9646
[1 0]	0.9665
[1 1]	0.0275

TABLE 5: Sine Function Accuracy Results

Threshold	Accuracy
Within 0.1	11.45%
Within 0.001	0.18%
Within 0.0001	0.02%

struggles to capture the underlying sin(x1-x2+x3-x4) pattern. This suggest although the performance seemed good on my training data it has generalized pretty poorly on my test set. These poor generalization metrics suggest its either overfitting to the training data or theres insufficient model capacity to properly represent the periodic relationships between the four input variables. I will need to go back and revise ways to potentially improve my test sets results.

3) Model 3 (Letter Recognition)

The letter recognition model achieves strong classification performance with a 94.08z% accuracy rate. The confusion matrix demonstrates clear that the model can clearly seperate multiple character classes. My model has generalised really well in this case indicating the model has successfully learned meaningful features for the character recognition.

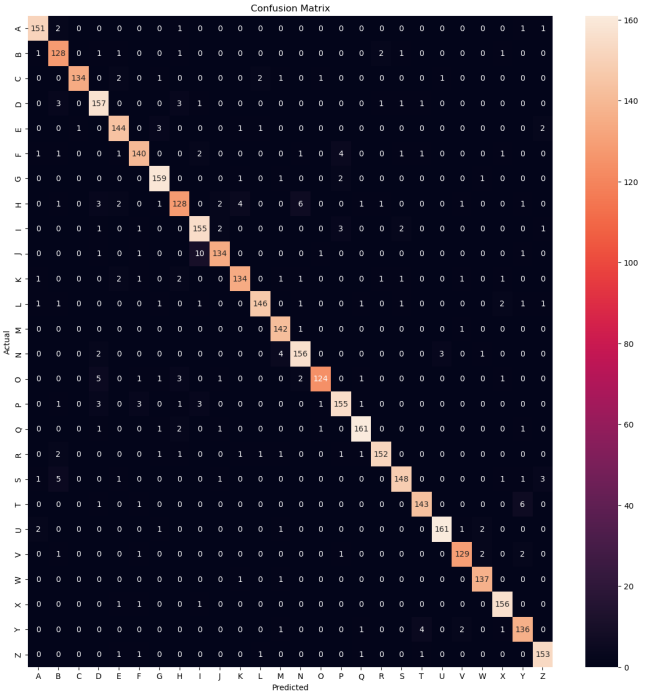

FIGURE 14: Confusion Matrix. The model achieved a final accuracy of 94.08%.

IV. CONCLUSION

Throughout this assignment I learned a lot about the interplay between networks architecture a significant one being the learning rates and weight initialization. I found the initial setting of the weights to be a pretty significant factor in the performance of my models. An example where I found setting larger weights(between 1 and -1) to be beneficial was the XOR problem however by doing this I introduce a number of problems into the model. As different runs resulted in sometimes significantly different results. Where as the small weights initialization really struggled to converge unless I went with a really large learning rate. Which is also not great practice. This lead me down the hole of researching good weight initialization techniques for different activations hence why I settled on the Xavier and He techniques for the models. Both of which ended up performing very well across all three models.

Another thing I found interesting was the impact the size of my network had on the models. In general if a task was complex it benefited from large network size and typically with that a small learning rate. However the idea of the law of diminishing returns was very evident in a number of our plots where we can see as the network size grew the percent

increase between plots really began to shrink a lot. However in general I found bigger to almost always be better if not just as good as smaller network size.

One thing I was surprised by was the poor performance of the SINE model on the test data. I experimented with a number of different activations, network sizes and learning rates and couldn't find any significant improvement. In fact quite often the model performed just as well on the test data when trained on a smaller network size and for much else epochs. This was despite the fact it had a significantly worse training error score. I can guess my model is suffering from over-fitting in that case and maybe a different approach needs to be taken to better generalize for the test set.

One thing I found interesting was when the model had a clear direction in which to go a larger learning rate benefited it hugely. For example the XOR problem benefited hugely as its a simple problem which a very clear direction therefore a high learning rate only sought to increase the speed of convergence. The importance of adaptive learning rates that are used in practice make a lot more sense to me now as when there is a clear direction in a problem having a learning rate that starts of larger and shrinks is super beneficial when it comes to reaching convergence faster.

Overall I learned a lot about multilayer perceptrons and there were a lot of interesting surprises in the assignment with a lot of trial and error.