Neural Network Computing - Project

Muhammad Osama Khan (mok232) | Abhishek Sharma (as10686) | Lihui Qin (lq330)

The goal of this project was to create a universal computing machine, using a three-layer feed-forward neural network to approximate the output for any given function f(x), within a desired accuracy (greater than 0).

Our approach towards formulating this universal computing machine comprised of 3 main parts.

In Part 1, the weights and biases after 1 epoch were validated given initial weights and biases for computing a specific function – XOR.

In Part 2, the hyperparameters – number of neurons of the hidden layer ($\boldsymbol{N1}$), the learning rate ($\boldsymbol{\alpha}$), width for random initialization of weights and biases ($\boldsymbol{\zeta}$), and the scalar argument for the transfer function ($\boldsymbol{x_0}$) – were varied while using the quadratic cost function.

Whereas in Part 3, the same hyperparameters – $\boldsymbol{N1, \alpha, \zeta \; and \; x_0}$ – were varied while using the cross-entropy cost function.

The results were then analyzed at each step and were found to be in close conformity with the theoretical expectations. The remaining portion of this report covers each of the parts and the findings in detail.

Part 1: XOR Weights Validation

Using the initial weights and biases provided, the following weights and biases were obtained after 1 epoch of training when using a 2-4-1 neural network architecture with the following hyperparameters:

$$\alpha = 0.2$$
$$x_0 = 1$$

Weights1 = [[0.193475, 0.316754, -0.144748, 0.363745],
[0.306867, 0.188452, -0.033015, -0.488590]]

Weights2 = [[0.475348], [0.276428], [-0.383950], [0.348013]] ^{T}

Biases1 = [[-0.322434, 0.265042, 0.273305, -0.32503622]]

Biases2 = [[-0.080274]]

These weights and biases were verified with the GA before continuing with the rest of the experiments.

Note: Throughout all the experiments, tolerance was kept fixed at 0.05 and max number of epochs was set to 700 after empirically trying several different combinations.

Part 2a: Varying α, ζ and x_0 (Quadratic Cost Function)

In this part of the project, the quadratic cost function was used and the following hyperparameter combinations were tested:

$$\alpha = [0.1, 0.2, 0.3]$$
$$\zeta = [0.5, 1.0, 1.5]$$
$$x_0 = [0.5, 1.0, 1.5]$$

Hence, a total of 27 hyperparameter combinations were tried. Note that for this part, the NN architecture was kept fixed at 2-4-1 in order to observe the effect of the different learning rates, weights and bias initializations and slope of the transfer function. The results are summarized below:

Table 1: Effect of the hyperparameters α, ζ and x_0 on convergence (quadratic cost function)

#	α	ζ	x_0	Final Epoch Error	Convergence	# Training Epochs
1	**0.1**	0.5	0.5	0.0496	Yes	172
2	**0.1**	0.5	1	0.0499	Yes	661
3	**0.1**	0.5	1.5	4.0497	No	700
4	**0.1**	1	0.5	0.0495	Yes	120
5	**0.1**	1	1	0.0499	Yes	535
6	**0.1**	1	1.5	1.3302	No	700
7	**0.1**	1.5	0.5	0.0499	Yes	109
8	**0.1**	1.5	1	0.0499	Yes	478
9	**0.1**	1.5	1.5	0.0852	No	700
10	**0.2**	0.5	0.5	0.0496	Yes	70
11	**0.2**	0.5	1	4.205	No	700
12	**0.2**	0.5	1.5	4.0934	No	700
13	**0.2**	1	0.5	0.0493	Yes	48
14	**0.2**	1	1	0.05	Yes	316
15	**0.2**	1	1.5	0.0515	No	700
16	**0.2**	1.5	0.5	0.0498	Yes	45
17	**0.2**	1.5	1	0.0499	Yes	428
18	**0.2**	1.5	1.5	0.05	Yes	517
19	**0.3**	0.5	0.5	0.0487	Yes	38
20	**0.3**	0.5	1	4.3105	No	700
21	**0.3**	0.5	1.5	4.1365	No	700
22	**0.3**	1	0.5	0.0492	Yes	28
23	**0.3**	1	1	0.0496	Yes	219
24	**0.3**	1	1.5	0.0499	Yes	491
25	**0.3**	1.5	0.5	0.0482	Yes	32
26	**0.3**	1.5	1	0.0499	Yes	129
27	**0.3**	1.5	1.5	0.0499	Yes	431

We see that 19 out of the 27 hyperparameter combinations tested lead to convergence. With max number of epochs set to 700, the learning rate α does not seem to affect how many hyperparameter combinations converge (6-7 convergence results in each set of 9 rows with α=0.1, α=0.2 and α=0.3).

However, the learning rate α does affect the *rate of convergence* if the experiments do converge. For instance, consider the hyperparameter combinations indicated by the rows highlighted in yellow. With the same settings for ζ *and* $\boldsymbol{x_0}$, increasing the learning rate decreases the number of training epochs required for convergence. Note, however, that increasing the learning rate leads to greater risk of skipping minima during the gradient descent process, which may lead to divergence. Interestingly, we observe that we even for smaller learning rates (e.g. 0.1) some of the iterations do not converge. This is probably a result of being stuck in local optima. This explains why it is preferable to vary the learning rate over the course of the experiment. It would be preferable to have a higher learning rate initially in order to reach near the global optima quicker (saving computational resources) and then use a lower learning rate for fine-tuning in order not to skip over the global optima.

As a further note, even though ζ *and* $\boldsymbol{x_0}$ have the same values in the experiments considered above, they might not necessarily yield the same results every time since the weights and biases are randomly initialized.

Furthermore, it is observed that using relatively low initialization values of the weights and biases (e.g. ζ = 0.5) decreases the chance of convergence. This can be seen from the rows highlighted in green where decreasing ζ increases the number of epochs needed for convergence. Here, it must be emphasized that in general it is not a good idea to initialize the weights and biases to very low values or very high values since that means that the net input into a neuron is either very low or very high. Applying the bipolar sigmoid transfer function then leads to activation values very close to -1 or 1 respectively. This then leads to extremely small gradients which slows down learning (vanishing gradient problem) (refer to Figure 1 below).

Lastly, it was observed that increasing $\boldsymbol{x_0}$ (slope argument of the transfer function) decreased the rate of convergence. This can be seen, for example, from the rows highlighted in orange. The following graphs for the bipolar sigmoid transfer function help explain why this is the case. The red, blue and green curves represent $\boldsymbol{x_0}$=0.5, $\boldsymbol{x_0}$=1 and $\boldsymbol{x_0}$=1.5 respectively. Since the slope of the red curve is greater, this means that lower values of $\boldsymbol{x_0}$ lead to faster rates of convergence as was confirmed empirically by the results above. This is because the slope of the transfer function is greater for lower values of $\boldsymbol{x_0}$ which means that the sensitivity of the final layer is greater. This sensitivity in turn propagates backwards. Since the sensitivity term appears in the weight updated formulas, this means that the weight updates are greater, which helps explain why the examples with lower $\boldsymbol{x_0}$ converge in lesser number of iterations.

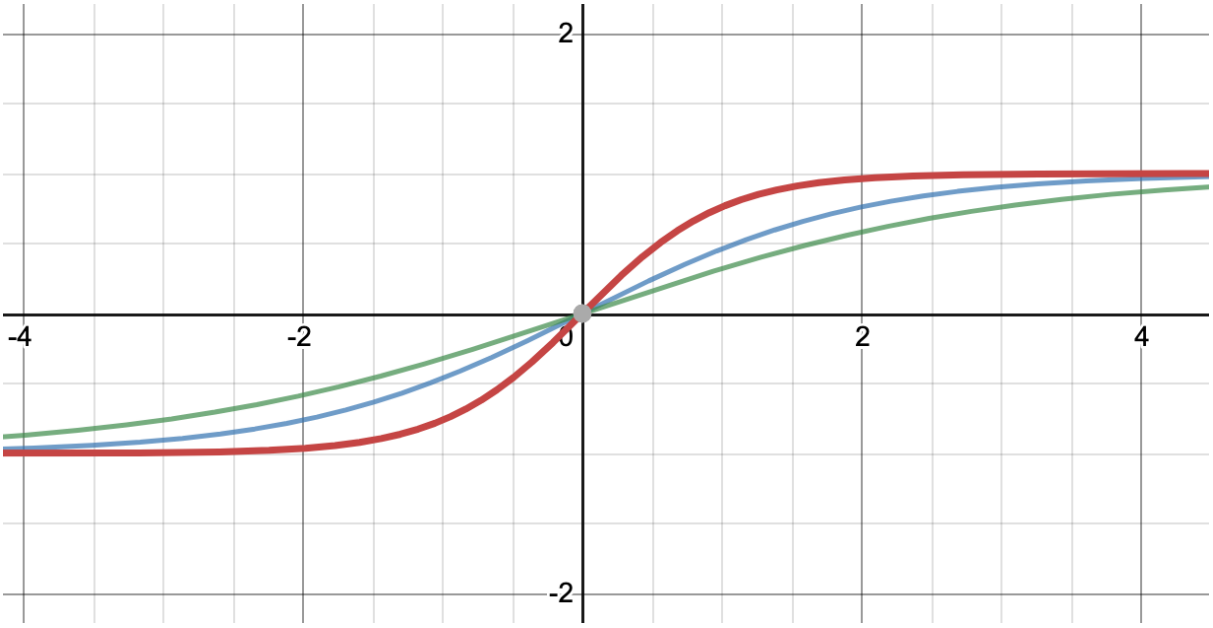

Figure 1: Bipolar sigmoid transfer functions with different values of x_0. The red, blue and green curves represent x_0=0.5, x_0=1 and x_0=1.5 respectively.

Part 2b: Varying $N1$ (hidden layer neurons) (Quadratic Cost Function)

In this part of the project, the NN architecture, **N1**, was changed while keeping α, ζ *and* x_0 fixed at 0.2, 1 and 1 respectively. NN architecture of the form 2-N1-1 was used where the following values for N1 were tried (each 100 times):

$$N1 = [2, 4, 6, 8, 10]$$

Table 2: Effect of varying number of hidden neurons on convergence (quadratic cost function).

N1	**# Iterations Not Converged**	**Convergence Epoch Statistics**			
		Min	Max	Mean	Median
2	16	264	678	351.0714	341.5
4	1	166	385	258.3333	261
6	1	160	482	223.1919	221
8	1	142	278	189.8384	184
10	1	135	329	179.4545	175

As can be seen from the results in Table 2, increasing the number of hidden layers increases the number of iterations that converge since it increases the representation power of the NN according to the universality theorem. Furthermore, the median number of iterations required for convergence also decreases. However, after N1 = 4, increasing N1 does not seem to yield similar magnitude gains in performance as almost all iterations converge when N1=4.

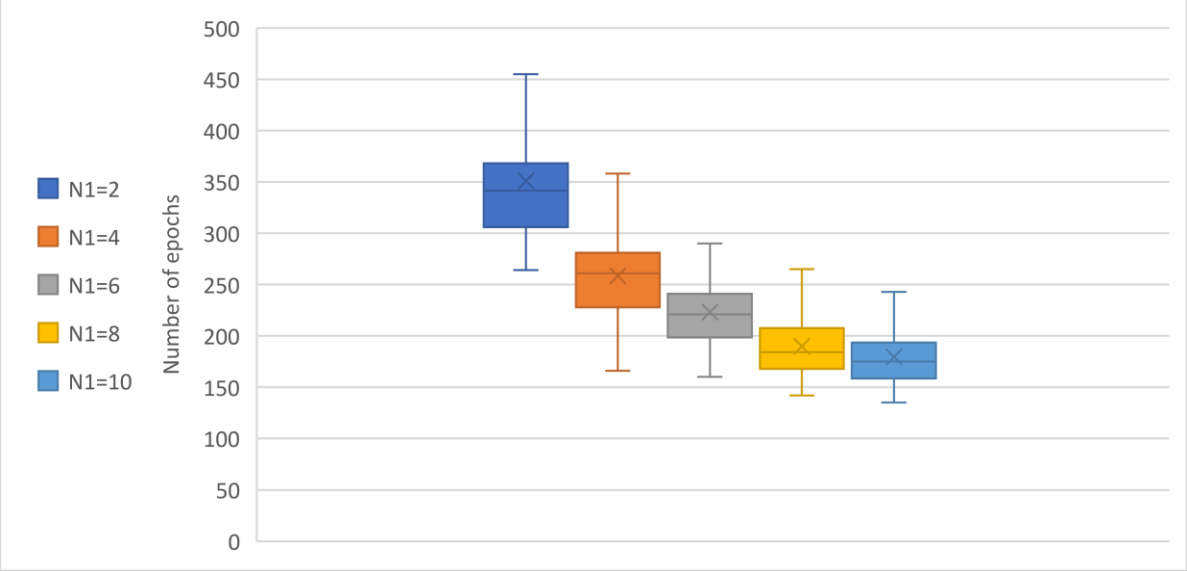

Figure 2: A box-and-whisker plot using the empirical data for various values of N1 – quadratic cost function.

In fact, increasing the number of hidden neurons N1 might have the undesired effect of overfitting to the training data. In order to avoid overfitting, we would need to use some form of regularization. However, in this example, since we don't have any test set, the results can only be reported on the training set and deductions regarding overfitting can't be made.

When using only 1 hidden neuron (N1=1), none of the 100 iterations converged. This is because the XOR problem is not linearly separable and the representation power of only 1 hidden neuron is not enough to approximate the XOR function. According to the universality theorem, using more neurons in the hidden layer increases the function approximation of the NN, as was observed for the cases where N1>=2.

Part 3a: Varying α, ζ and x_0 (Cross Entropy Cost Function)

In this part of the project, the cross-entropy cost function was used and the following hyperparameter combinations were tested:

$$\alpha = [0.1, 0.2, 0.3]$$
$$\zeta = [0.5, 1.0, 1.5]$$
$$x_0 = [0.5, 1.0, 1.5]$$

Using the cross-entropy cost function has the effect that only the formula used for the sensitivity of the last layer changes as can be seen from the source code.

A total of 27 hyperparameter combinations were tried. Note that for this part, the NN architecture was kept fixed at 2-4-1 in order to observe the effect of the different learning rates, weights and bias initializations and slope of the transfer function. The results are summarized in below.

Table 3: Effect of the hyperparameters α, ζ and x_0 on convergence (cross entropy cost function).

#	α	ζ	x_0	Final Epoch Error	Convergence	# Training Epochs
1	**0.1**	0.5	0.5	0.0478	Yes	75
2	**0.1**	0.5	1	4.205	No	700
3	**0.1**	0.5	1.5	4.1366	No	700
4	**0.1**	1	0.5	0.0486	Yes	51
5	**0.1**	1	1	0.0493	Yes	153
6	**0.1**	1	1.5	0.0493	Yes	275
7	**0.1**	1.5	0.5	0.0483	Yes	25
8	**0.1**	1.5	1	0.0497	Yes	114
9	**0.1**	1.5	1.5	0.0497	Yes	181
10	**0.2**	0.5	0.5	0.0495	Yes	52
11	**0.2**	0.5	1	4.4182	No	700
12	**0.2**	0.5	1.5	4.2751	No	700
13	**0.2**	1	0.5	0.0472	Yes	26
14	**0.2**	1	1	0.0481	Yes	93
15	**0.2**	1	1.5	0.0484	Yes	187
16	**0.2**	1.5	0.5	0.0427	Yes	12
17	**0.2**	1.5	1	0.0484	Yes	34
18	**0.2**	1.5	1.5	0.0497	Yes	177
19	**0.3**	0.5	0.5	5.3024	No	700
20	**0.3**	0.5	1	4.638	No	700
21	**0.3**	0.5	1.5	4.4182	No	700
22	**0.3**	1	0.5	0.047	Yes	9
23	**0.3**	1	1	0.05	Yes	58
24	**0.3**	1	1.5	0.0478	Yes	114
25	**0.3**	1.5	0.5	0.0381	Yes	9
26	**0.3**	1.5	1	0.0476	Yes	42
27	**0.3**	1.5	1.5	0.0492	Yes	96

With the cross-entropy cost function, 20 out of the 27 hyperparameter combinations tested lead to convergence compared to the quadratic cost function case where 19 hyperparameter combinations converged.

The cross-entropy cost function has the property that the derivative of the cost function is very large when the predicted value is far from the ground truth, which makes learning fast. Hence, we observe that some of the iterations converge very rapidly (e.g. the rows highlighted in dark gray) in comparison to the same iterations with the quadratic cost function.

Like the quadratic cost function case, we see that increasing the learning rate $\boldsymbol{\alpha}$ increases the rate of convergence or it may lead to divergence as in the case highlighted in blue. Increasing $\boldsymbol{\zeta}$ from 0.5 – 1.5 decreases the number of epochs needed for convergence as was also observed for the quadratic cost function before. Similarly, it was also observed that lower values of $\boldsymbol{x_0}$ lead to faster rates of convergence.

Part 3b: Varying $N1$ (hidden layer neurons) (Cross Entropy Cost Function)

In this part of the project, the NN architecture was changed while keeping $\alpha, \zeta \; and \; x_0$ fixed at 0.2, 1 and 1 respectively. NN architecture of the form 2-N1-1 was used where the following values for N1 were tried (each 100 times):

$$N1 = [2, 4, 6, 8, 10]$$

Table 4: Effect of varying number of hidden neurons on convergence (cross entropy cost function).

N1	**# Iterations Not Converged**	**Convergence Epoch Statistics**			
		Min	Max	Mean	Median
2	39	59	186	97.42623	92
4	11	48	172	79.78652	75
6	6	39	140	67.93617	63
8	0	35	162	56.73	52.5
10	0	34	92	49.83	49

Again, like Part 2b, increasing the number of hidden layers increases the number of iterations that converged since it increases the representation power of the NN according to the universality theorem (Figure 3).

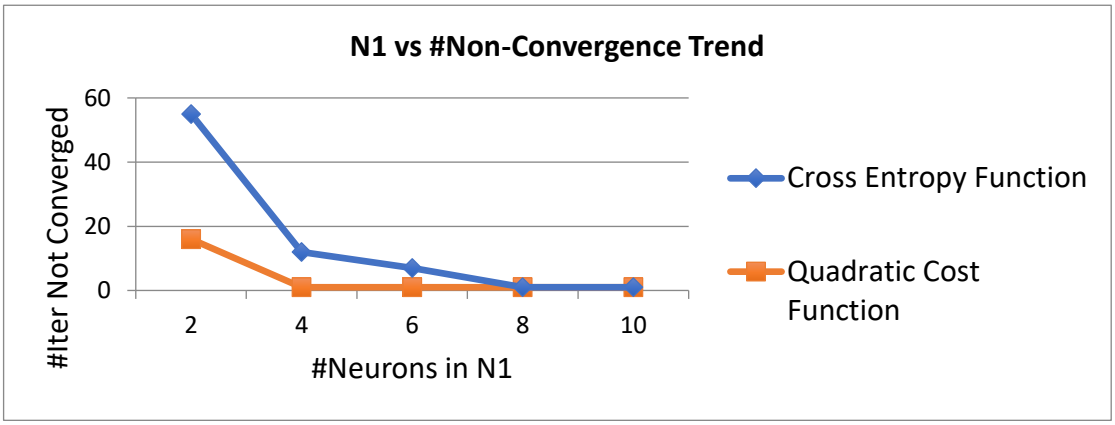

Figure 3: A stacked-line plot showing the trend of increased neurons in hidden layer (N1) decreased number of non-converged iterations

Furthermore, we note that when using the cross-entropy cost function, the rate of convergence increases. This can be seen by comparing the median number of epochs required for convergence when using the quadratic cost function vs cross entropy cost function (Tables 2 and 4). This is because the quadratic cost function can suffer from learning slowdown. In contrast, the cross-entropy cost function has the property that the derivative of the cost function is very large when the predicted value is far from the ground truth, which makes learning fast. Hence, we observe that the rate of convergence increases when using the cross-entropy cost function.

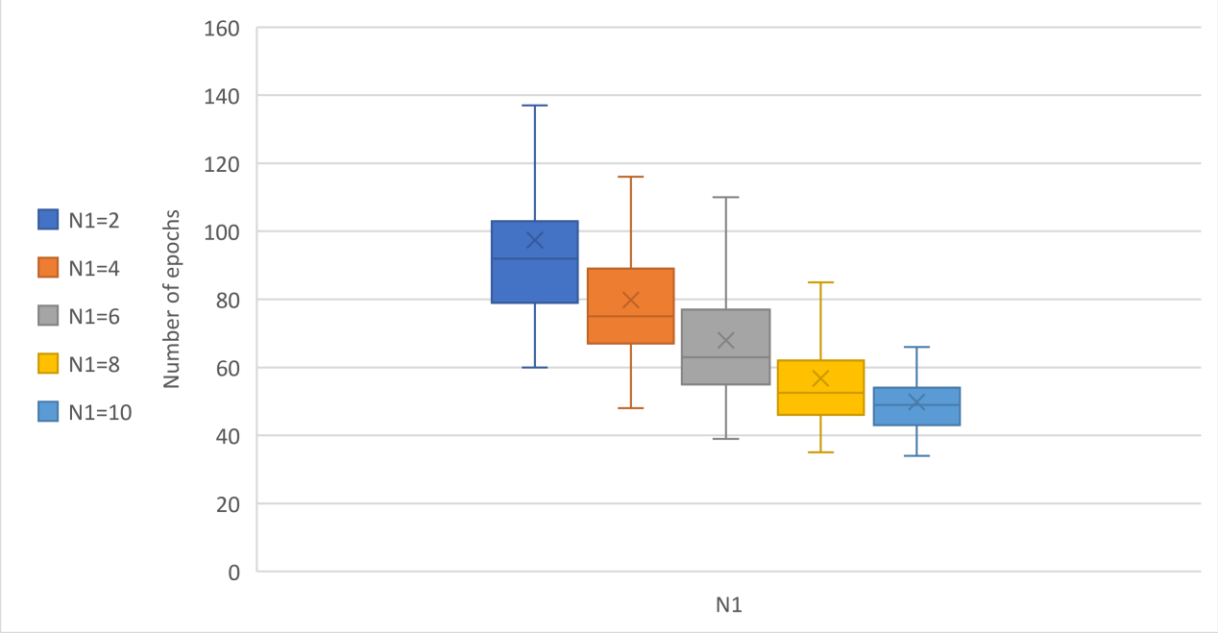

Figure 4: A box-and-whisker plot using the empirical data for various values of N1 – cross entropy cost function.

When using only 1 hidden neuron (N1=1), none of the 100 iterations converged. This is because the XOR problem is not linearly separable and the representation power of only 1 hidden neuron is not enough to approximate the XOR function. According to the universality theorem, using more neurons in the hidden layer increases the function approximation of the NN, as was observed for the cases where N1>=2.

Remarks regarding iterations that did not converge

Most of the reasons for non-convergence have been discussed in detail above along with experimental results. In summary, for most of the runs that did not converge, the main problems were that either the number of hidden layers was too low (e.g. N1=1), the learning rate α was too high (which caused divergence), the scale parameter x_0 of the bipolar sigmoid transfer function was too high (smaller gradients) or the weights and biases were initialized to very low values (e.g. ζ=0.5) (vanishing gradient problem). Additionally, for iterations that converged most of times (e.g. 98/100), the non-convergent iterations could be attributed to the random initialization of weights and biases which might have caused the problem to be stuck in a local minimum.

Part 4. Weights and biases for N1 = 4, α = 0.2, ζ = 1.0, and x0 = 1.0 after 1 epoch

Weights1 = [[0.19383967 0.30895515 -0.14727152 0.36911844]
[0.29881712 0.18811518 -0.02889164 -0.48928638]]

Biases1 = [-0.30754551 0.24693804 0.25732658 -0.30887916]

Weights2 = [0.44724602 -0.24360234 -0.35686098 0.32501904]

Biases2 = [-0.01923564]]