18310726 Julia Filipczak

COMP30230 Connectionist Computing Programming Assignment Report

Coding Choices

Python was chosen for this project since it's intuitive, concise and makes building of models like this assignment easier. A lot of the suggestions from the assignment specification were incorporated into the implementation. The randomise() function initialises the W1 and W2 to small random values. This is also the place where the arrays for the weight changes are set to all zeroes. The forward() function distributes the input throughout the system and stores the result in O[]. This function also takes in an activation type parameter to differentiate types which will depend on which tests it is used for. The sigmoid activation is used by the XOR function, whereas tanh is used for SIN and letter recognition. The backwards() function calculates the error of the current output based on the target output. This error is then backpropagated through the system. This function also takes in the activation parameter which specifies the activation type like in the forward() function.

Q1 – XOR

1. A number of different learning rates and hidden unit values was used for the purpose of testing in this question. The epochs used ranged from 100 to 1,000,000 and even at those higher values the program was relatively fast to run.

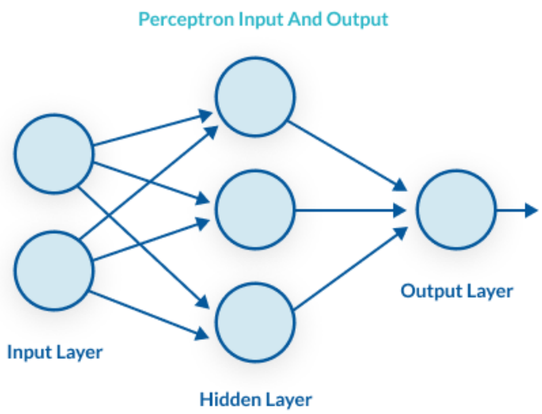

Figure 1: Layers in a perceptron [1]

The inputs and outputs given in the assignment specification were used, namely ((0, 0), 0) ((0, 1), 1) ((1, 0), 1) ((1, 1), 0). For each pair of inputs there should be one output and the amount of neurons in the hidden layer varied across the different tests. The neural network modifies its weights with different learning rates and iterates according to the answers. All the inputs were forwarded through the system during the training. After the completion of backpropagation, the neural network is ready to predict the answers. Sigmoid algorithm was used to predict the outputs for this question. The input array was forwarded again and the accuracy was calculated.

Hidden units: 3

↓ Learning rate, → Epochs

	100	1000	10000	100000	1000000	Accuracy
1	0.4997754588982175	0.09445309778276939	0.019642997028552645	0.005845162219909833	0.0018106743156055514	0.9981893266003508
0.75	0.49979861912644286	0.14727588216996096	0.02285476936925597	0.006773463407666801	0.002095721604730819	0.9979042794563148
0.5	0.5000334125097283	0.4238442893256221	0.029211898085330118	0.00836230392508616	0.002573395520205006	0.997426605785342
0.25	0.5000813390162572	0.47812238753456926	0.0436715264907183	0.011987500208330087	0.003657977054672888	0.99634202480796
0.05	0.49886374403581557	0.4992999402998813	0.24275654695450344	0.02853616115475215	0.00834464369375192	0.9916553606185565

Hidden units: 4

↓ Learning rate, → Epochs

	100	1000	10000	100000	1000000	Accuracy
1	0.49814307242584055	0.08178781967775194	0.015620149381854562	0.004502738287564087	0.0014553443729892386	0.9985446563445518
0.75	0.49361603991927017	0.10081150807382135	0.021302627436411398	0.006409771225820961	0.0019888612594451504	0.9980111397468407
0.5	0.499994027536691	0.4997646593341145	0.02887604002637308	0.007942078868967344	0.002437421777556199	0.9975625794598685
0.25	0.49943182402334957	0.4193481323129695	0.04115867981431498	0.011636551308953346	0.003023494838547779	0.9969765073857346
0.05	0.5000470968815169	0.5000518580070656	0.287723062578562	0.025312761574023034	0.006415357097114462	0.9935846461550312

Hidden units: 5

↓ Learning rate, → Epochs

	100	1000	10000	100000	1000000	Accuracy
1	0.4987364305048847	0.0741030233014965	0.016258747248731765	0.004684975275529719	**0.0011405319640957708**	**0.9988594686999286**
0.75	0.49790529728971217	0.10768820223837405	0.020866518178789335	0.005302428926888445	0.0016473655928749722	0.9983526352264166
0.5	0.49864600927246694	0.17334245383737698	0.02500510272078042	0.006178601429593457	0.0019164460795849446	0.9980835548671808
0.25	0.5002212717671062	0.4440846889235712	0.04073097842451367	0.009480754218094672	0.002746949535247349	0.9972530518421053
0.05	0.4988469975531365	0.4997972016570869	0.22525844545034604	0.02671691686784915	0.006206541480711315	0.9937934623800837

Hidden units: 6

↓ Learning rate, → Epochs

	100	1000	10000	100000	1000000	Accuracy
1	0.4999004607777355	0.09690564026872003	0.017677056989886385	0.0044571056180262685	0.0013139816203728582	0.9986860190475421
0.75	0.4987925949926231	0.10021475556181374	0.016893301756713174	0.005010015873156355	0.0015506257796105698	0.9984493750067097
0.5	0.49938725963477526	0.2829575671677137	0.025780179011971056	0.006735960905644764	0.0018367586159591855	0.9981632423132738
0.25	0.49885643103642463	0.40863495523905025	0.0395971207826565	0.010571356742520003	0.0028197839175286557	0.997180217652527
0.05	0.5005113851374285	0.500298563970837	0.3539544018653854	0.025612027917160968	0.006593271570188056	0.9934067321863354

Hidden units: 7

↓ Learning rate, → Epochs

	100	1000	10000	100000	1000000	Accuracy
1	0.4999889468013331	0.09574941530641068	0.016563223483808995	0.004449972235578817	0.0013739672712529817	0.9986260334188622
0.75	0.4997335556982462	0.131934842266219	0.01911684218531571	0.005013780979687775	0.0014277457673013436	0.9985722549686116
0.5	0.4984861022209544	0.1802200381173395	0.0206360616555659	0.0059577210951262975	0.0017926037250862589	0.9982073972579067
0.25	0.49905080309614114	0.40606129302890104	0.03079485670983549	0.0083909510790299985	0.0025971806775525485	0.9974028206337613
0.05	0.5000587395382362	0.4989309110932938	0.17841086158629782	0.024654119108681913	0.006757179853139298	0.993242824160439

Based on observation of the results, it's evident that the test performs better with a higher learning rate as all of lowest errors occurred in each test occurred at the learning rate of 1. The lowest error occurred at 5 hidden units and 1000000 epochs and learning rate of 1. I was surprised to see that tests with higher number of hidden units achieved worse error rate. Before carrying out those tests I was convinced that increasing the number of hidden units would increase the accuracy of prediction, although at the expense of performance. Following this test I was led to believe that too many hidden units can also decrease the accuracy. After some research I have found out that this is indeed true. Firstly, having too many units in the hidden layer can result in overfitting. This occurs when neural network has so much information processing capacity that the limited amount of information contained in the training set is not enough to train all of the neurons in the hidden layers. Another problem which can occur even when the training data is sufficient, is that a large number of neurons in the hidden layers can increase the time it takes to train the network. The amount of training time can then increase to the point that it is impossible to adequately train the neural network.[2] Among the different tests which were carried out a trend can be noticed where in all cases the lowest error occurs at the maximum number of epochs (in this case 1000000).

Another observation which can be made is that the limited amount of data meant that the same data had to be used for both training and testing. This caused the model to have poor generalization ability and made it prone to overfitting as mentioned above.

2. Upon testing the lowest error model with the test data provided the accuracy achieved was 99.87%. Given that this is a very high percentage it can be concluded that MLP can indeed learn XOR. It has also been verified that at the end

of training, the MLP correctly predicts all the examples. The cases where the target value was 0 are at 0.00.. after training whereas when the target was 1 are at 0.99.., which when rounded up gives the correct values. The exact figures can be seen in the test output files.

Q2 – Sin

3. The aim of this exercise is to compute sin(x1-x2+x3-x4), the tanh function can be used for this purpose. The sin function takes 4 inputs which can be combined to form a vector. The value of each input should be a random value between -1 and 1. To generate the input vectors the Python library called numpy was used. It provides a random.uniform function which gives an array of given size and range. Hence, 4 numbers between -1 and 1 were generated 500 times. For each of these groups of 4 generated inputs the sin(x1-x2+x3-x4) was calculated to generate an output set. Then the system was trained using 400 of these and tested using the remaining 100. Since the dataset is bigger this time I started off with 10 hidden units and went up by 10 for each test.

Hidden units: 10

↓ Learning rate, → Epochs

	100	1000	10000	100000	1000000
0.1	7.192319817531134	9.49113038537905	11.796907092477127	13.094504685521324	83.92300598994647
0.01	0.17630919919411447	0.06148321100482797	0.06282738637772566	0.06168258524422542	0.06371164293423445
0.001	0.15352139035123893	0.0554291198293582	0.02972311784763683	0.020783976417805876	0.02019475631261658
0.0001	0.2748613747103332	0.1538375315462685	0.052481291327491617	0.03345878293330646	0.009294469186627843

Hidden units: 20

↓ Learning rate, → Epochs

	100	1000	10000	100000	1000000
0.1	11.994560424147982	82.7180462165915	14.32555360250589	61.112551219304265	16.480170490921793
0.01	0.11005709759428774	0.08762546111095477	0.09415150570371898	0.09667724678260563	0.09362312214258493
0.001	0.18869238311405995	0.057167683953263236	0.024854228851334533	0.01478170529188243	0.010753057311802266
0.0001	3.1051928971625427	0.1930154689044668	0.055076285647061046	0.025090887173331723	0.007774579442372487

Hidden units: 30

↓ Learning rate, → Epochs

	100	1000	10000	100000	1000000
0.1	23.0079254098837	47.219346699115725	63.977967070508946	7.006852526879918	14.803262921035047
0.01	0.21148737368556997	0.08732621077836868	0.09116421984859738	0.09726941116738307	0.09541089763138644
0.001	0.6821818431849838	0.04089912973675565	0.019781603080317914	0.01447875548312877	0.008284635785147546
0.0001	5.206075299866674	0.20354155855070324	0.06951281224520137	0.02341326829341881	0.008500560881266805

Hidden units: 40

↓ Learning rate, → Epochs

	100	1000	10000	100000	1000000
0.1	87.86316825269274	142.02333535125192	145.63641847489345	146.0680253580371	145.56556867359933
0.01	0.1161197165865212	0.06327071519490321	0.07270361073454974	0.0709176266527068	0.07179253805708101
0.001	1.4332090137981188	0.03923390232961208	0.019039593596453616	0.01481682749298275	**0.006572759704708141**
0.0001	6.935548377497998	0.5044400650983105	0.05877990665347502	0.01710506778558894	0.007505772490996825

The observations made are that testing with more hidden values improved the accuracy. Another observation was that the more epochs used the better the result was and that lower learning rate was preferable. However, as the hidden units were increased 0.001 became more preferable for higher accuracy than the 0.0001 which yielded better results previously. It would have been interesting to see whether the optimal learning rate increases further through further testing.

4. The lowest error of 0.006572759704708141 was obtained with 40 hidden units and at 1000000 epochs with a learning rate of 0.001. The accuracy was quite high with 99.31% and the error has definitely decreased meaning that MLP is able to learn sin() function. This is also shown by the results which compare the predictions before and after training, these are included in the test output files. Since the error rate kept decreasing I'm convinced that further tests with an increased number of hidden units might have led to even better results, however I'm also conscious of the fact that at some point an optimal number of hidden units would have been reached after which increase would no longer yield desirable results. Given that the execution of the test with 40 hidden units took around 3 hours I have decided to not go ahead with a bigger number.

Q3 Letter recognition

This exercise aims to apply the MLP on the UCI Letter Recognition Data Set. The dataset was split into a training part containing 4/5 of the items (16000) and a testing part containing the remaining 1/5 (4000) of the items. Each item in the dataset consists of 16 attributes which describe the letter therefore the model has 16 inputs. Since the dataset is concerned with letter prediction the number of outputs should match the number of letters in the alphabet. The 16 attributes have been normalized to ensure that the accuracy of the prediction is not affected. The resulting output is a matrix of 26 elements where each represents the likelihood of each of the letters occurring. Based on those values the largest one is chosen to predict the letter.

Hidden units: 10, Learning rate = 5e-06

Epochs	Error
5000	1.4537171889230796
10000	0.2664705417281237
15000	0.07570564383227822
20000	0.08011705065122383
25000	0.08086266185076946
30000	0.08074920138591478
35000	0.08036471258239228
40000	0.07974971073560483
45000	0.0791730403276846
50000	0.07872783311179397
55000	0.07838535514124413
60000	0.07812559329421451
65000	0.07800780226422246
70000	0.07809280842242747
75000	0.07813330481687598
80000	0.07773210422942316
85000	0.07717085802327724
90000	0.07672835853553837
95000	0.07641459305245625
100000	0.07617446435666624

Hidden units: 20, Learning rate = 5e-06

Epochs	Error
5000	9.917096030790248
10000	9.912365211183685
15000	9.907332327683255
20000	9.90192314172349
25000	9.896028721258256
30000	9.88947937825585
35000	9.881988297772281
40000	9.873009368404098
45000	9.861280794899873
50000	9.84249409971579
55000	9.59960381817654
60000	9.596412408875736
65000	9.59071121197064
70000	9.581052250119482
75000	9.560692971154081
80000	9.479880909924164
85000	9.341847651216227
90000	9.37473246679543
95000	9.399486277389821
100000	9.402932328741915

Hidden units: 30, Learning rate = 5e-06

Epochs	Error
5000	14.810010500084301
10000	14.809981451140889
15000	14.809952362832648
20000	14.809923235050356
25000	14.809894067684576
30000	14.809864860625371
35000	14.809835613762083
40000	14.809806326983805
45000	14.809777000179214
50000	14.809747633236444
55000	14.80971822604324
60000	14.809688778486795
65000	14.80965929045369
70000	14.80962976183031
75000	14.809600192502407
80000	14.809570582355285
85000	14.809540931273693
90000	14.809511239141944
95000	14.809481505843769
100000	14.809451731262582

Hidden units:10 (higher epoch test), Learning rate = 5e-06

Epochs	Error
50000	0.07872783311179397
100000	0.07617446435666624
150000	0.0747266886053331
200000	0.07477618645807428
250000	0.07332680968382348
300000	0.07224531857515955
350000	0.07183871644378866
400000	0.07164906673413897
450000	0.07126644610625858
500000	0.07089686949948208
550000	0.07057381307690223
600000	0.07027438563995345
650000	0.06999458787021243
700000	0.06973345771199833
750000	0.06948705051517366
800000	0.06924886502036462
850000	0.06901307929787096
900000	0.06878288393507274
950000	0.0685670438203772
1000000	**0.06837104268957342**

This test doesn't achieve very high accuracy compared to the previous two exercises. As can be seen in the results the first test which was carried out yielded only 31.8% accuracy. Due to the size of the dataset I was limited to smaller epoch numbers. As a result I decided to note the error at different epoch numbers during my execution. A clear trend is visible that the more the epochs the lower the error. However, since I only used 10 hidden units for my test, I was also interested to see how increasing the number of hidden units would affect the predictions. I ran two tests one with 20 and the second one with 30 hidden units. As expected, the tests took a significant amount of time to execute but unfortunately the results were not very promising. The accuracy decreased significantly in both cases with 4.2% in the first and 3.4% in the second. Also, a bizarre trend developed in both where the model would essentially only learn to predict one letter out of the whole alphabet and would predict a 100% of the examples featuring those letters correctly. This is the reason for the low accuracy percentage as it only accounts for those correct predictions of that particular letter. The accuracy percentage of the 20 hidden units test is higher then the one of the 30 hidden units one, however this seems to be a coincidence due to the number of the times the single letter which was learned occurred within the test sample. I'm led to believe that the reason for these results is due to overfitting.

Inspired by my findings and the clear trend between the higher epoch number and accuracy of prediction, I decided to undertake one final yet very ambitious test. I set my max epoch number as 1 million and kept the hidden units as 10 which would allow me to directly compare the results with the first test I run by keeping the number of hidden units constant. Needless to mention, the execution of this test took a very long amount of time, in fact it spanned over a period of two days. However, I was able to confirm my claim that the increased number of epochs did indeed increase the accuracy of the prediction of letters. The lowest error of 0.06837104268957342 occurred at 100000 epoch. Compared to the first test, the accuracy achieved here went up to 38% (6.2% increase). A sample of the predicted letters has been included in the output file. I would have been interested to see whether increasing the number of hidden units by a smaller amount would have had any positive effect however given how long the other tests took to execute I was quite limited by time. It would have also been interesting to look at even higher number of epoch, however I believe that in that case I would have been limited not only by time but the also the capabilities of the machine I used to run these tests. Finally, another factor which could have been varied across the tests is the learning rate yet once again I did not get a chance to explore its effect in my tests.

Conclusion

Overall I'm quite satisfied with the results I obtained as they were more accurate than I have hoped for. In the first two exercises my model was able to learn really well, with both the xor and sin functions it was able to achieve over 99% accuracy. Although the letter recognition did not achieve the same level of accuracy, given that the character images in the dataset were based on 20 different fonts, I consider the 38% accuracy achieved as significant. Furthermore, I believe that with more experiments and computational power I would be able to improve that result even further.

The XOR data was quite limited which led to overfitting which was quite visible in my results. The approach applied to the sin and letter recognition exercise where the dataset was much bigger and split into training set and a test set is much more desirable.

It was an interesting exercise to try to find the right combination of parameters. For instance smaller learning rates require more training epochs since smaller changes will be made to weight with each update whereas larger learning rates will apply bigger changes with the updates so less epochs should be needed. A value that is too small will result in a long learning process. This leads to another kind of discussion which is could we use models which have adaptive learning rates.

While accuracy does depend on factors like the number of hidden units/layers, number of epochs etc, it also depends on other things like the quality of the model as well the quality and quantity of the data which is used for training. An interesting exercise for the future would be to try to also incorporate those factors more by comparing various datasets and their performance and changing how the model is implemented.

In general what I have noticed across these exercises is that increasing the number of epochs is quite helpful in increasing the accuracy of prediction, which seems intuitive since it gives the model more time to reduce the error. It does however significantly affect the execution time therefore it might not be the most desirable approach. It is possible to increase the number of hidden units, however as it was already demonstrated this only works to a certain extent. Another approach could be to incorporate multiple hidden layers although this increases the complexity of implementation, moreover it is believed that a vast majority of problems do not require multiple hidden layers.

One of the main outcomes of this assignment for me is that I really got an appreciation for how models like this are trained and tested. The process of doing so involves carefully balancing the various parameters, analysing the outcome and adapting those over and over many times. We might be able to come up with a set of guidelines on how to chose such values, however it's unlikely that these will be optimal for all kinds of problems. This leads me back to Minky's idea [3] of how to approach Artificial Intelligence, that it cannot be modelled with a single technique and instead many different ones should be combined. This assignment really gave me a first-hand experience of this.

References

[1] “Solving the XOR problem using MLP | by Priyansh Kedia | MLearning.ai | Medium”

[2] The Number of Hidden Layers | Heaton Research (no date). Available at: https://www.heatonresearch.com/2017/06/01/hidden-layers.html.

[3] Minsky, M.L., 1991. Logical versus analogical or symbolic versus connectionist or neat versus scruffy. AI magazine, 12(2), pp.34-34.