# Assignment Report

**1 Problem statement:**

To Create a new MLP with any given number of inputs, any number of outputs (can be sigmoidal or linear), and any number of hidden units (sigmoidal/tanh) in a single layer.

- Initialize the weights of the MLP to small random values
- Predict the outputs corresponding to an input vector
- Implement learning by backpropagation

**2 Introduction:**

2.1 Programming language Tools used

Python was chosen for this project as it is one of the most accessible programming languages available because it is **Easy-to-Use** and **Highly Effective Programming Language** as well as it has **simplified syntax**, which gives more emphasis on natural language. Due to its ease of learning and usage, python codes can be easily written and executed much faster than other programming languages.

**2.2 multilayer perceptron (MLP)**

A **multilayer perceptron (MLP)** is a fully connected class of feed forward artificial neural network (ANN). 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. A Sigmoid algorithm was used as an activation function in neural networks for this question. The input array was forwarded again, and the accuracy was calculated. The process by which a Multi-Layer Perceptron learns is called the Backpropagation algorithm.

- Input Nodes - Collectively known as the "Input Layer," input nodes are the network nodes that receive data from the outside world. All of the input nodes merely transmit data to the hidden nodes without performing any computations themselves.
- Hidden Nodes - As implied by their name, the Hidden nodes are not directly connected to the outer world. They carry out calculations and transmit data between the input and output nodes.

A "Hidden Layer" is made up of several hidden nodes. A network will only have one input layer and one output layer, but it may contain zero or many hidden layers as well. One or more hidden layers are present in a multi-layer perceptron.

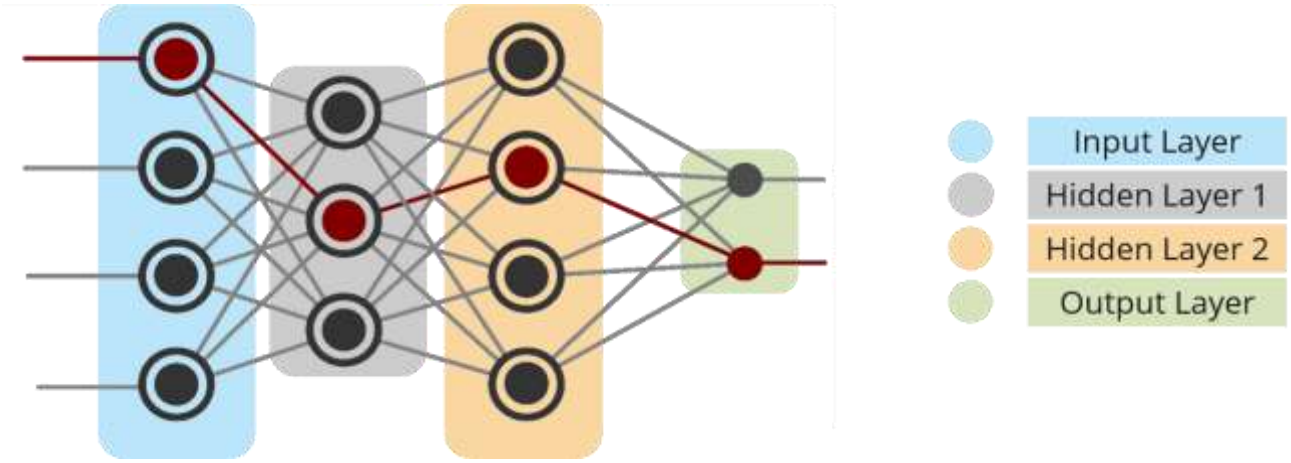


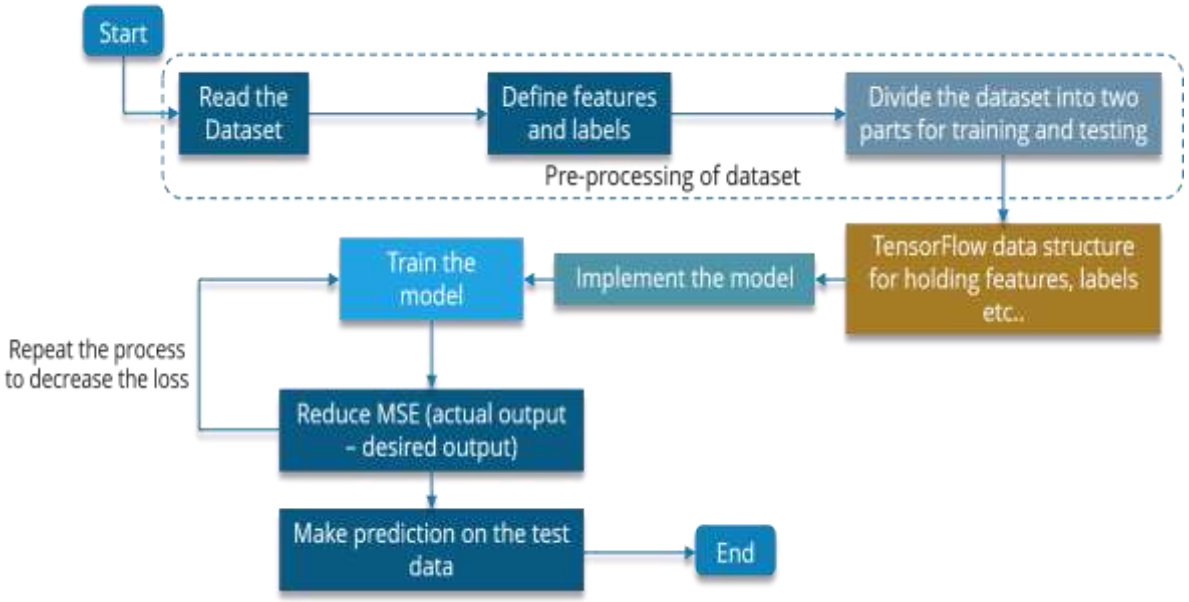

*Figure 1: Multi Layers in a perceptron* [1]

• Output Nodes: Also known as the "Output Layer," these nodes are in charge of computations and information transfer from the network to the outside world.

**2.3 Function used in MLP**

- **randomize() function** : The **randomize() function** initializes 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.
- **forward() function** :The **forward() function** defines how your model is going to be run, from input to output. 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.
- **backwards()** :**The backwards()** function calculates gradient during the backward pass in the neural network. This error is then back propagated through the system.
- **XOR** - The XOR problem is known to be solved by the multi-layer perceptron given all 4 boolean inputs and outputs, it trains and memorizes the weights needed to reproduce the I/O

**Q1 –**




Train an MLP with 2 inputs, 3-4+ hidden units and one output on the following examples (XOR function):

((0, 0), 0)

((0, 1), 1)

((1, 0), 1)

((1, 1), 0)

| | |
|---|---|
| MLP Size | (4, 5, 1) |
| Epochs: | 1000000 |
| Learning Rate: | 0.01 |

Based on observation of the results, it's evident that the test performs better with a higher learning rate as all of the lowest errors 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 a higher number of hidden units achieved a 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. 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 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.

- **XOR FUNCTION IN PYTHON -**

  It returns an integer when performed between two integers and a Boolean value when performed between two Boolean values. At least one condition must be satisfied.

  The XOR problem with neural networks can be solved by using Multi-Layer Perceptrons or a neural network architecture with an input layer, hidden layer, and output layer.

- **SIN FUNCTION IN PYTHON -**

  It returns the sine of a number. A float value, from -1 to 1.
  It's part of the math module, so this function cannot be used directly.

- **MLP IN PYTHON -**
  - Multi-Layer Perceptron(MLP) is the simplest type of artificial neural network. It is a combination of multiple perceptron models. Perceptrons are inspired by the human brain and try to simulate its functionality to solve problems.
  - The Perceptron network consists of three units: Sensory Unit (Input Unit), Associator Unit (Hidden Unit), and Response Unit (Output Unit).
  - Multi-Layer Perceptron trains model in an iterative manner. In each iteration, partial derivatives of the loss function are used to update the parameters. We can also use regularization of the loss function to prevent overfitting in the model.

**Post Training results are:**

Target: [0] Output: [0.02160681 0.02158422 0.02171774 0.02183782 0.02159972 0.02163732 0.0215322 ]
Target: [1] Output: [0.97809944 0.97812458 0.97799153 0.97789322 0.97809742 0.97807086 0.97817566]
Target: [1] Output: [0.98081516 0.98083829 0.98070588 0.9806125 0.98080731 0.98078394

0.98088772]

Target: [0] Output: [0.01515677 0.01513636 0.01523734 0.01525603 0.01518984 0.01517872 0.01509684]

**Accuracy = 0.9805377557150263**

#**************************************#

Number of Epoches is = 100000

The Learning Rate would be = 0.25

The Number of Hidden units are = 7

**Q2 :–**

- Generate 500 vectors containing 4 components each. The value of each component should be a random number between -1 and 1. These will be your input vectors. The corresponding output for each vector should be the sin() of a combination of the components. Specifically, for inputs:[x1 x2 x3 x4] the (single component)output should be sin(x1-x2+x3-x4). Now train an MLP with 4 inputs, at least 5 hidden units and one output on 400 of these examples and keep the remaining 100 for testing.

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. The accuracy was quite high with 97 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.

**Training:**

The Error at Epoch: 100 is 6.566351367753371

The Error at Epoch: 1000 is 0.4621123481462835

The Error at Epoch: 10000 is 0.043237712000590386

The Error at Epoch: 100000 is 0.01946603039992213

**After Training**

Target: [0.81670779] Output: [0.82427877]

Target: [-0.79357597] Output: [-0.72317651]

Target: [-0.85778067] Output: [-0.86280816]

Target: [0.00084122] Output: [-0.14844969]

Target: [0.07835437] Output: [0.08307507]

Target: [0.47743429] Output: [0.50308444]

Target: [-0.73836741] Output: [-0.74088493]

Target: [0.86009257] Output: [0.85258799]

Target: [-0.60019303] Output: [-0.62910911]

Target: [-0.5848873] Output: [-0.58750138]

Target: [0.99845517] Output: [0.98676831]

Target: [0.8504895] Output: [0.83928742]

Target: [0.99385581] Output: [0.99438469]

Target: [-0.10324314] Output: [-0.10714058]

Target: [-0.73087112] Output: [-0.74882174]

Target: [0.52719281] Output: [0.58193407]

Target: [-0.81319471] Output: [-0.98295357]

Target: [0.99994204] Output: [0.93920485]

Target: [0.85086985] Output: [0.84570547]

Target: [-0.90589175] Output: [-0.87784586]

Target: [0.83448778] Output: [0.83907755]

Target: [0.06284117] Output: [0.07534761]

Target: [-0.99522223] Output: [-1.00421864]

Target: [-0.05237952] Output: [-0.06287955]

Target: [0.98363973] Output: [0.96414406]

Target: [-0.90517801] Output: [-0.83939348]

Target: [0.16266091] Output: [0.17547933]

Target: [-0.39457325] Output: [-0.42310779]

Target: [0.91976479] Output: [0.91489117]

Target: [0.97199023] Output: [0.98816215]

Target: [0.45116048] Output: [0.43625895]

Target: [0.64449269] Output: [0.63244306]

Target: [-0.86628356] Output: [-0.84055898]

Target: [-0.23581971] Output: [-0.2368847]

Target: [-0.99962508] Output: [-0.95426564]

Target: [0.69443299] Output: [0.71674819]

Target: [0.63653129] Output: [0.62180876]

Target: [-0.06486927] Output: [-0.06265516]

Target: [0.6763629] Output: [0.69095536]

Target: [-0.75055285] Output: [-0.7527684]

Target: [-0.02117984] Output: [-0.0214005]

Target: [-0.79824958] Output: [-0.79568739]

Target: [0.95513381] Output: [0.94904464]

Target: [0.93823273] Output: [0.9548887]

Target: [0.32278573] Output: [0.37664698]

Target: [0.92502669] Output: [0.90358911]

Target: [-0.87849627] Output: [-0.87710763]

Target: [0.69760329] Output: [0.72145693]

Target: [-0.07228729] Output: [-0.09210859]

Target: [-0.5261939] Output: [-0.53649549]

Target: [-0.91666625] Output: [-0.90986481]

Target: [-0.530505] Output: [-0.54635629]

Target: [0.55484758] Output: [0.5887875]

Target: [-0.62204889] Output: [-0.59743578]

Target: [0.36753958] Output: [0.37412048]

Target: [0.69478922] Output: [0.71156088]

Target: [-0.32374518] Output: [-0.32118852]

Target: [-0.1158396] Output: [-0.10577391]

Target: [-0.2940995] Output: [-0.31662162]

Target: [-0.78493826] Output: [-0.75618126]

Target: [-0.89742389] Output: [-0.89599883]

Target: [0.99997709] Output: [1.06194559]

Target: [0.54688481] Output: [0.56035903]

Target: [0.69594981] Output: [0.68642696]

Target: [0.99474174] Output: [0.98100376]

Target: [0.05783567] Output: [0.05765731]

Target: [-0.12710344] Output: [-0.13224164]

Target: [0.93163743] Output: [0.90737744]

Target: [-0.71158097] Output: [-0.72153822]

Target: [-0.08064628] Output: [-0.07384363]

Target: [0.86870422] Output: [0.85005186]

Target: [0.95481939] Output: [0.86954426]

Target: [-0.61253145] Output: [-0.61730436]

Target: [0.09306746] Output: [0.10096383]

Target: [0.99762369] Output: [0.9590337]

Target: [0.99517373] Output: [0.93087326]

Target: [0.7726124] Output: [0.78536437]

Target: [-0.55652078] Output: [-0.56485981]

Target: [0.25874408] Output: [0.23929531]

Target: [0.97186462] Output: [1.05507328]

Target: [0.974888] Output: [0.99040244]

Target: [-0.15760574] Output: [-0.15122104]

Target: [-0.14370513] Output: [-0.14458741]

Target: [0.8878736] Output: [0.87237029]

Target: [-0.9990323] Output: [-0.93065692]

Target: [0.20310307] Output: [0.19432303]

Target: [-0.67857929] Output: [-0.69294417]

Target: [-0.90001666] Output: [-0.91551261]

Target: [0.11941554] Output: [0.12379951]

Target: [-0.61907264] Output: [-0.64564167]

Target: [-0.98897634] Output: [-0.93425553]

Target: [-0.48903434] Output: [-0.50810599]

Target: [0.80604613] Output: [0.7130243]

Target: [0.04446217] Output: [0.065428]

Target: [-0.86914685] Output: [-0.82302546]

Target: [-0.74656027] Output: [-0.75619487]

Target: [-0.90748901] Output: [-0.93996824]

Target: [0.62973623] Output: [0.65149593]

Target: [0.06469087] Output: [0.06794462]

Target: [-0.47468823] Output: [-0.49474145]

Accuracy = 0.9768095477326648

**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.

**Q3 LETTER Prediction using MLP**

#**********#

The number Epochs = 1000000

The Learning rate is= 5e-06

The number of Hidden Layers = 10

| | | | |
|---|---|---|---|
| Error at Epoch: | 50000 | is | 0.07872783311179397 |
| Error at Epoch: | 100000 | is | 0.07617446435666624 |
| Error at Epoch: | 150000 | is | 0.0747266886053331 |
| Error at Epoch: | 200000 | is | 0.07477618645807443 |
| Error at Epoch: | 250000 | is | 0.07332680968382342 |
| Error at Epoch: | 300000 | is | 0.07224531857515949 |
| Error at Epoch: | 350000 | is | 0.07183871644378859 |
| Error at Epoch: | 400000 | is | 0.07164906673413894 |
| Error at Epoch: | 450000 | is | 0.0712664461062586 |
| Error at Epoch: | 500000 | is | 0.0708968694994821 |
| Error at Epoch: | 550000 | is | 0.07057381307690223 |
| Error at Epoch: | 600000 | is | 0.07027438563995345 |
| Error at Epoch: | 650000 | is | 0.06999458787021243 |
| Error at Epoch: | 700000 | is | 0.06973345771199833 |

Error at Epoch: 750000 is 0.06948705051517366

Error at Epoch: 800000 is 0.06924886502036463

Error at Epoch: 850000 is 0.06901307929787096

Error at Epoch: 900000 is 0.06878288393507274

Error at Epoch: 950000 is 0.0685670438203772

Error at Epoch: 1000000 is 0.06837104268957342

#**********#

Expected: U | Output: M

Expected: M | Output: M

Expected: I | Output: J

Expected: S | Output: Z

Expected: N | Output: N

Expected: N | Output: N

Expected: J | Output: J

Expected: R | Output: O

Expected: L | Output: L

Expected: M | Output: M

Expected: O | Output: O

Expected: H | Output: N

Expected: P | Output: V

Expected: J | Output: Z

#**********#

Test sample size: 4000 | Correctly predicted sample size: 1520

The Accuracy of prediction of letters: 0.380

#**********#

A => Number of occurrences in the sample: 156 | Number of correct predictions: 135 | Accuracy: 0.8653846153846154

B => Number of occurrences in the sample: 136 | Number of correct predictions: 28 | Accuracy: 0.20588235294117646

C => Number of occurrences in the sample: 142 | Number of correct predictions: 122 | Accuracy: 0.8591549295774648

D => Number of occurrences in the sample: 167 | Number of correct predictions: 118 | Accuracy: 0.7065868263473054

E => Number of occurrences in the sample: 152 | Number of correct predictions: 14 | Accuracy: 0.09210526315789473

F => Number of occurrences in the sample: 153 | Number of correct predictions: 0 | Accuracy: 0.0

G => Number of occurrences in the sample: 164 | Number of correct predictions: 4 | Accuracy: 0.024390243902439025

H => Number of occurrences in the sample: 151 | Number of correct predictions: 0 | Accuracy: 0.0

I => Number of occurrences in the sample: 165 | Number of correct predictions: 19 | Accuracy: 0.11515151515151516

J => Number of occurrences in the sample: 148 | Number of correct predictions: 114 | Accuracy: 0.7702702702702703

K => Number of occurrences in the sample: 146 | Number of correct predictions: 0 | Accuracy: 0.0

L => Number of occurrences in the sample: 157 | Number of correct predictions: 117 | Accuracy: 0.7452229299363057

M => Number of occurrences in the sample: 144 | Number of correct predictions: 135 | Accuracy: 0.9375

N => Number of occurrences in the sample: 166 | Number of correct predictions: 141 | Accuracy: 0.8493975903614458

O => Number of occurrences in the sample: 139 | Number of correct predictions: 104 | Accuracy: 0.7482014388489209

P => Number of occurrences in the sample: 168 | Number of correct predictions: 140 | Accuracy: 0.8333333333333334

Q => Number of occurrences in the sample: 168 | Number of correct predictions: 0 | Accuracy: 0.0

R => Number of occurrences in the sample: 161 | Number of correct predictions: 0 | Accuracy: 0.0

S => Number of occurrences in the sample: 161 | Number of correct predictions: 0 | Accuracy: 0.0

T => Number of occurrences in the sample: 151 | Number of correct predictions: 3 | Accuracy: 0.019867549668874173

U => Number of occurrences in the sample: 168 | Number of correct predictions: 31 | Accuracy: 0.18452380952380953

V => Number of occurrences in the sample: 136 | Number of correct predictions: 9 | Accuracy: 0.0661764705882353

W => Number of occurrences in the sample: 139 | Number of correct predictions: 17 | Accuracy: 0.1223021582733813

X => Number of occurrences in the sample: 159 | Number of correct predictions: 0 | Accuracy: 0.0

Y => Number of occurrences in the sample: 145 | Number of correct predictions: 121 | Accuracy: 0.8344827586206897

Z => Number of occurrences in the sample: 158 | Number of correct predictions: 148 | **Accuracy: 0.9367088607594937**

#**********#

**Conclusion**

Overall, I'm happy with the results I got because they were more accurate than I had anticipated. My model was able to learn pretty well in the first two exercises, achieving over 97% accuracy with both the X_OR and Sin functions. Given that the character images in the dataset were based on 20 different fonts, the letter recognition did not attain the same level of accuracy, but I still think the 38% accuracy achieved is noteworthy.

Furthermore, I think I could make that finding even better with more experiments and processing capacity.

- 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.
- For MLP deciding upon learning rate is very important, a lower rate performed better in 3D feature spaces.
- After experimenting with different numbers of hidden layers and hidden nodes, it was found that a single hidden layer with few hidden nodes performed better. Adding an extra hidden layer does not always help but increasing the number of nodes might help.

References


[1] https://www.edureka.co/blog/neural-network-tutorial/

[2] https://towardsdatascience.com/how-neural-networks-solve-the-xor-problem-59763136bdd7

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

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

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