THE MULTI-LAYER PERCEPTRON IN PRACTICE

SESSION 14

AIM

To familiarise students with the basic concept of Multi Layer Perceptron Learning

Practice

INSTRUCTIONAL OBJECTIVES

This unit is designed to:

1. Demonstrate Multi Layer Perceptron overview
2. List out the functions of MLP
3. Describe the loss functions

LEARNING OUTCOMES

At the end of this unit, you should be able to:

1. Define the functions MLP Training Algorithm
2. Summarise the development MLPNN
3. Describe the various activation functions with loss values.

TOPICS TO BE COVERED

1. Multi Layer Perceptron
2. MLP Architecture
3. MLP Learning Algorithm using XOR Gate
4. MLP in Practice

WHAT IS MULTI LAYER PERCEPTRON

- A **multilayer perceptron** (**MLP**) is a fully connected class of feedforward artificial neural network (ANN). Multilayer perceptrons are sometimes colloquially referred to as "vanilla" neural networks, especially when they have a single hidden layer.
- An MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function.
- MLP utilizes a chain rule based supervised learning technique called backpropagation or reverse mode of automatic differentiation for training.
- Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable.

- The term "multilayer perceptron" does not refer to a single perceptron that has multiple layers.
- MLP perceptrons can employ arbitrary activation functions.
- A true perceptron performs *binary* classification, an MLP neuron is free to either perform classification or regression, depending upon its activation function.

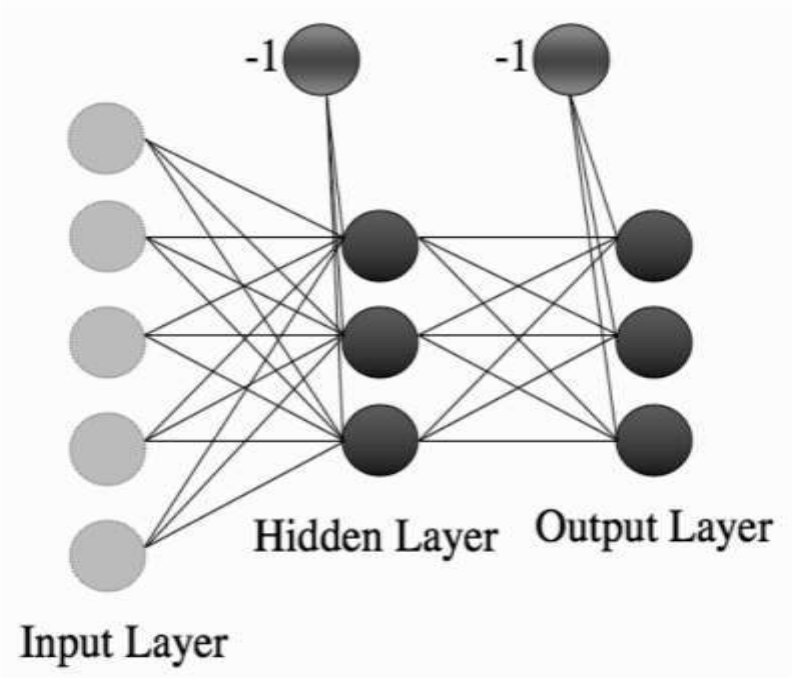


FIGURE 4.1 The Multi-layer Perceptron network, consisting of multiple layers of connected neurons.

Activation function

If a multilayer perceptron has a linear activation function in all neurons, that is, a linear function that maps the weighted inputs to the output of each neuron, then linear algebra shows that any number of layers can be reduced to a two-layer input-output model. In MLPs some neurons use a nonlinear activation function that was developed to model the frequency of action potentials, or firing, of biological neurons.

The two historically common activation functions are both sigmoids, and are described by

.

$$y(v_i) = \tanh(v_i) \text{ and } y(v_i) = (1 + e^{-v_i})^{-1}$$

The first is a hyberbolic tangent that ranges from -1 to 1, while the other is the logistic function, which is similar in shape but ranges from 0 to 1.

MLP LEARNING ALGORITHM

- we can check that a prepared network can solve the two-dimensional XOR problem, something that we have seen is not possible for a linear model like the Perceptron.
- A suitable network is shown in Figure, treating it as two different Perceptrons,
- first computing the activations of the neurons in the middle layer (labelled as C and D in Figure 4.2) and then using those activations as the inputs to the single neuron at the output.
- As an example, I'll work out what happens when you put in (1, 0) as an input; the job of checking the rest is up to you.

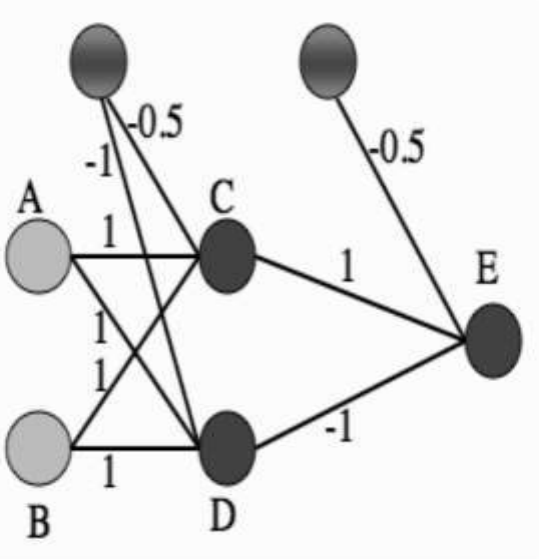


FIGURE 4.2 A Multi-layer Perceptron network showing a set of weights that solve the XOR problem.

MLP Learning Algorithm using XOR Gate

- Input (1, 0) corresponds to node A being 1 and B being 0. The input to neuron C is therefore
  $-1 \times 0.5 + 1 \times 1 + 0 \times 1 = -0.5 + 1 = 0.5$.

This is above the threshold of 0, and so neuron C fires, giving output 1.

- For neuron D the input is

$-1 \times 1 + 1 \times 1 + 0 \times 1 = -1 + 1 = 0$, and so it does not fire, giving output 0.

- Therefore the input to neuron E is
  $-1 \times 0.5 + 1 \times 1 + 0 \times -1 = 0.5$, so neuron E fires.
- Checking the result of the inputs should persuade you that neuron E fires when inputs A and B are different to each other, but does not fire when they are the same, which is exactly the XOR function.

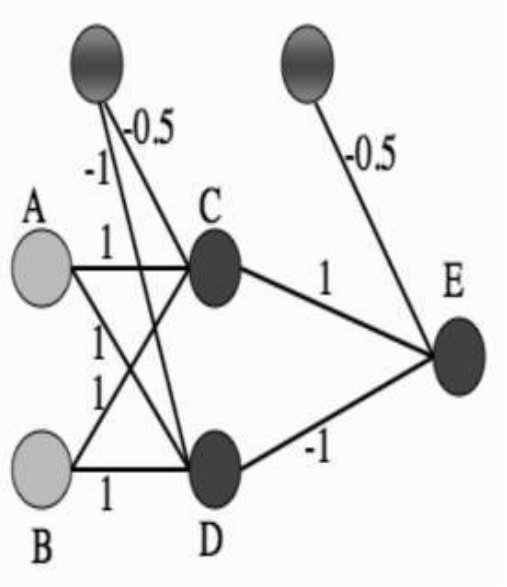


FIGURE 4.2 A Multi-layer Perceptron network showing a set of weights that solve the XOR problem.

MLP Implementation using Numpy

```
import numpy as np
import mlp

anddata = np.array([[0,0,0],[0,1,0],[1,0,0],[1,1,1]])
xordata = np.array([[0,0,0],[0,1,1],[1,0,1],[1,1,0]])

p = mlp.mlp(anddata[:,0:2],anddata[:,2:3],2)
p.mlptrain(anddata[:,0:2],anddata[:,2:3],0.25,1001)
p.confmat(anddata[:,0:2],anddata[:,2:3])

q = mlp.mlp(xordata[:,0:2],xordata[:,2:3],2)
q.mlptrain(xordata[:,0:2],xordata[:,2:3],0.25,5001)
q.confmat(xordata[:,0:2],xordata[:,2:3])
```

```
Iteration:  0  Error:  0.367917569871
Iteration:  1000  Error:  0.0204860723612
Confusion matrix is:
[[ 3.  0.]
 [ 0.  1.]]
Percentage Correct:  100.0
Iteration:  0  Error:  0.515798627074
Iteration:  1000  Error:  0.499568173798
```

INITIALIZE THE WEIGHTS

- The MLP algorithm suggests that the weights are initialised to small random numbers, both positive and negative.
- If the initial weight values are close to 1 or -1 then the inputs to the sigmoid are also likely to be close to ±1 and so the output of the neuron is either 0 or 1.
- if we view the values of these inputs as having uniform variance, then the typical input to the neuron will be $w\sqrt{n}$, where w is the initialization value of the weights. So a common trick is to set the weights in the range $-1/\sqrt{n} < w < 1/\sqrt{n}$, where n is the number of nodes in the input layer to those weights.

THE MULTI-LAYER PERCEPTRON IN PRACTICE

In this section, we are going to look more at choices that can be made about the network in order to use it for solving real problems to four different types of problem: regression, classification, time-series prediction, and data compression.

1. **Amount of Training Data**

For the MLP with one hidden layer there are (L + 1) × M + (M + 1) × N weights, where L, M, N are the number of nodes in the input, hidden, and output layers, respectively. The extra +1s come from the bias nodes, which also have adjustable weights.

2. Number of Hidden Layers

Two hidden layers are sufficient to compute for different inputs, and so if the function that we want to learn (approximate) is continuous, the network can compute it. It can therefore approximate any decision boundary, not just the linear one that the Perceptron computed.

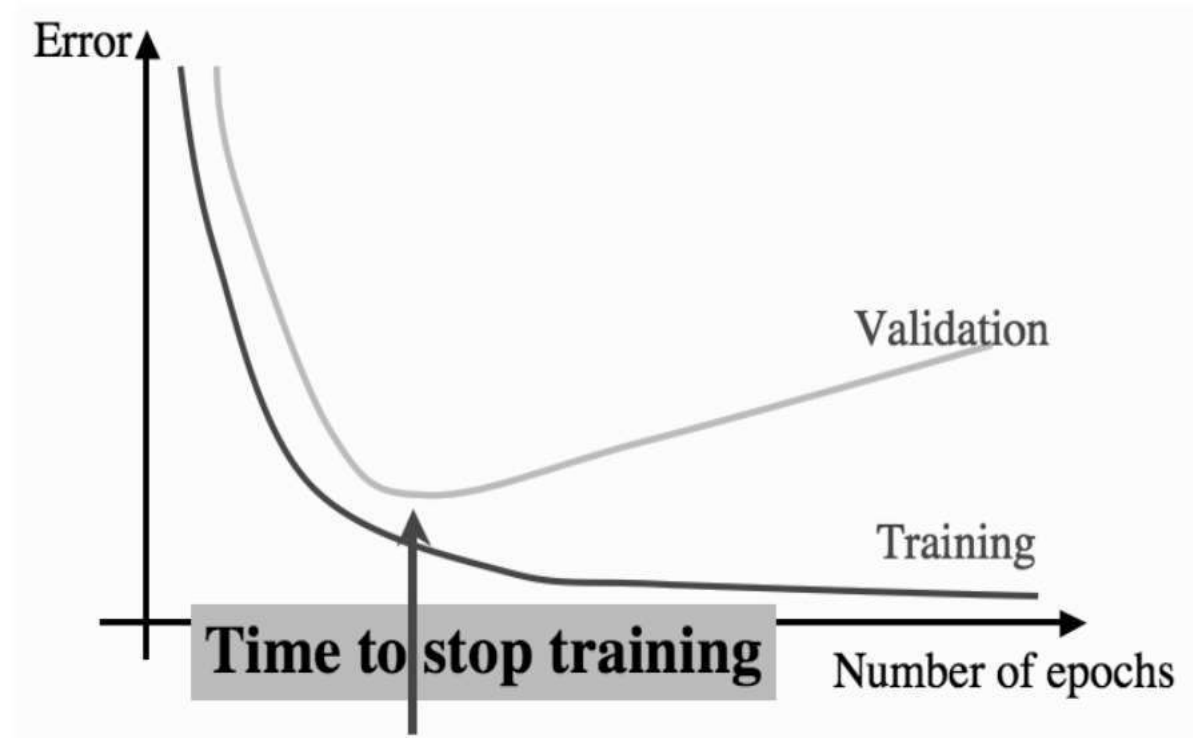
THE MULTI-LAYER PERCEPTRON IN PRACTICE

3. When to Stop Learning

To monitor the generalisation ability of the network at its current stage of learning. If we plot the sum-of-squares error during training, it typically reduces fairly quickly during the first few training iterations.

At some stage the error on the validation set will start increasing again, because the network has stopped learning about the function that generated the data, and started to learn about the noise that is in the data itself (shown in Figure).

At this stage we stop the training. This technique is called early stopping.



WEB REFERENCES

[1] https://www.guru99.com/backpropogation-neural-network.html

[2] https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

[3] http://neuralnetworksanddeeplearning.com/chap2.html

Self-Assessment Questions

1. Why is the XOR problem exceptionally interesting to neural network researchers?

(a) Because it can be expressed in a way that allows you to use a neural network Loss function

(b) Because it is complex binary operation that cannot be solved using neural networks Positive Function

(c) Because it can be solved by a single layer perceptron

(d) Because it is the simplest linearly inseparable problem that exists.

2. A perceptron adds up all the weighted inputs it receives, and if it exceeds a certain value, it outputs a 1, otherwise it just outputs a 0.

a) **True**

b) False

THANK YOU

OUR TEAM