CSE 412: Machine Learning Lab

Lab Activity 7: XOR Classification Using Multilayer Perceptron

Dr Muhammad Abul Hasan*

May 6, 2024

1 Objective

XOR classification using a Multilayer Perceptron (MLP) is a classic example that highlights the power of neural networks in solving non-linearly separable problems. Here are three key points related to the objects of learning in XOR classification using an MLP:

- **Non-linearity in XOR Problem**: The XOR (exclusive OR) problem involves classifying inputs into two classes (0 or 1) based on a combination of features. The challenge with XOR arises because it is not linearly separable—meaning a single straight line cannot accurately divide the data points of the two classes. This non-linearity requires a more complex model like a Multilayer Perceptron (MLP) to capture intricate relationships and patterns in the data.
- **Multilayer Perceptron Architecture**: The MLP architecture consists of an input layer, one or more hidden layers, and an output layer. For XOR classification, a simple architecture with one hidden layer containing at least two neurons is sufficient. The non-linear activation function, such as the sigmoid or hyperbolic tangent (tanh), is crucial in the hidden layer to introduce non-linearity and enable the network to learn complex mappings. The output layer typically uses a sigmoid activation function to produce values between 0 and 1, representing the predicted class probabilities.
- **Backpropagation for Training**: Training an MLP for XOR classification involves the use of the backpropagation algorithm. During training, the network adjusts its weights based on the error between the predicted output and the actual target. The backpropagation algorithm calculates the gradient of the error with respect to the weights and updates them using gradient descent. The iterative process of forward propagation (calculating predictions) and backward propagation (updating weights) continues until the model converges to a state where the error is minimized. This training process allows the MLP to learn the non-linear decision boundaries required for accurate XOR classification. Understanding these key points provides insights into the challenges posed by non-linearly separable problems like XOR and the effective use of neural networks, specifically Multilayer Perceptrons, in addressing such challenges.

2 Background

The XOR problem is a classic example of a problem that can be solved by a multilayer perceptron but not by a single-layer perceptron. The multilayer perceptron uses multiple layers and non-linear activation functions to learn non-linear decision boundaries and solve problems that are not linearly separable. The

*Associate Professor, Deptartment of CSE, GUB muhammad.hasan@cse.green.edu.bd

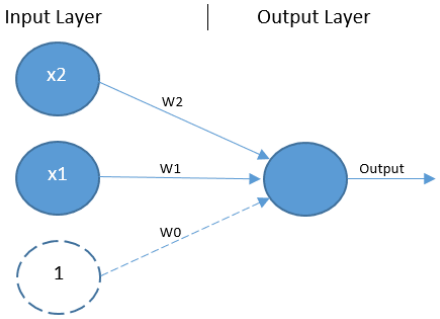

Figure 1: Perceptron Architecture.

backpropagation algorithm is used to train the multilayer perceptron by iteratively adjusting the weights of the neural network to minimize the error between the predicted output and the actual output.

3 The XOR Problem

The XOR problem serves as a paradigmatic illustration of a challenge that necessitates the capabilities of a multilayer perceptron, in contrast to a single-layer perceptron. Utilizing multiple layers and non-linear activation functions, the multilayer perceptron excels at discerning non-linear decision boundaries, making it adept at addressing problems that lack linear separability. To train the multilayer perceptron, the backpropagation algorithm is employed, iteratively fine-tuning the neural network's weights to minimize the disparity between predicted and actual outputs.

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

XOR is a classification problem and one for which the expected outputs are known in advance. It is therefore appropriate to use a supervised learning approach.

On the surface, XOR appears to be a very simple problem, however, it was considered a big problem for neural network architectures of the 1960s, known as perceptions.

4 Perceptrons

Like all artificial neural networks, the perceptron is composed of a network of *units*, which are analogous to biological neurons. A unit can receive input from other units. In doing so, it takes the sum of all values received and decides whether it is going to forward a signal to other units to which it is connected. This is called activation. The activation function uses some means or other to reduce the sum of input values to a 1 or a 0 (or a value very close to a 1 or 0) in order to represent activation or lack thereof. Another form of unit, known as a bias unit, always activates, typically sending a hard-coded 1 to all units to which it is connected.

Perceptrons include a single layer of input units — including one bias unit — and a single output unit (see Figure 1). Here a bias unit is depicted by a dashed circle, while other units are shown as blue circles. There are two non-bias input units representing the two binary input values for XOR. Any number of input units can be included.

The perceptron is a type of feed-forward network, which means the process of generating an output — known as forward propagation — flows in one direction from the input layer to the output layer.

There are no connections between units in the input layer. Instead, all units in the input layer are connected directly to the output unit.

A simplified explanation of the forward propagation process is that the input values X1 and X2, along with the bias value of 1, are multiplied by their respective weights W0..W2, and parsed to the output unit. The output unit takes the sum of those values and employs an activation function — typically the Heavside step function — to convert the resulting value to a 0 or 1, thus classifying the input values as 0 or 1.

It is the setting of the weight variables that give the network's author control over the process of converting input values to output values. It is the weights that determine where the classification line, the line that separates data points into classification groups, is drawn. If all data points on one side of a classification line are assigned the class of 0, all others are classified as 1.

A limitation of this architecture is that it is only capable of separating data points with a single line. This is unfortunate because the XOR inputs are not linearly separable. This is particularly visible if you plot the XOR input values to a graph. As shown in Figure 2, the output of OR can be linearly separable, however, there is no way to separate the XOR 1 and 0 predictions with a single classification line.

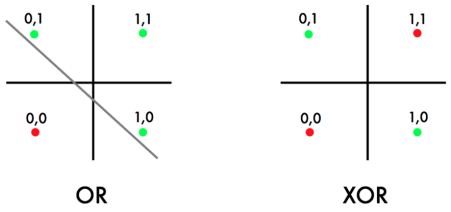

Figure 2: Outputs of OR and XOR logical units.

5 Multilayer Perceptrons

The solution to this problem is to expand beyond the single-layer architecture by adding an additional layer of units without any direct access to the outside world, known as a hidden layer. This kind of architecture — shown in Figure 3 — is another feed-forward network known as a multilayer perceptron (MLP).

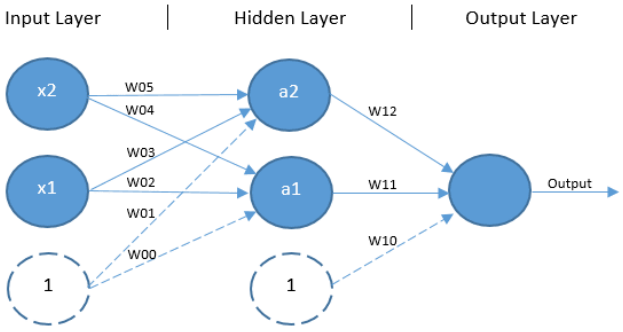

Figure 3: An example of Multilayer Perceptron architecture.

It is worth noting that an MLP can have any number of units in its input, hidden, and output layers. There can also be any number of hidden layers. The architecture used here is designed specifically for the XOR problem.

Similar to the classic perceptron, forward propagation begins with the input values and bias unit from the input layer being multiplied by their respective weights, however, in this case, there is a weight for each combination of input (including the input layer's bias unit) and hidden unit (excluding the hidden

layer's bias unit). The products of the input layer values and their respective weights are parsed as input to the non-bias units in the hidden layer. Each non-bias hidden unit invokes an activation function — usually the classic sigmoid function in the case of the XOR problem — to squash the sum of their input values down to a value that falls between 0 and 1 (usually a value very close to either 0 or 1). The outputs of each hidden layer unit, including the bias unit, are then multiplied by another set of respective weights and parsed to an output unit. The output unit also parses the sum of its input values through an activation function — again, the sigmoid function is appropriate here — to return an output value falling between 0 and 1. This is the predicted output.

This architecture, while more complex than that of the classic perceptron network, is capable of achieving non-linear separation. Thus, with the right set of weight values, it can provide the necessary separation to accurately classify the XOR inputs (see Figure 4).

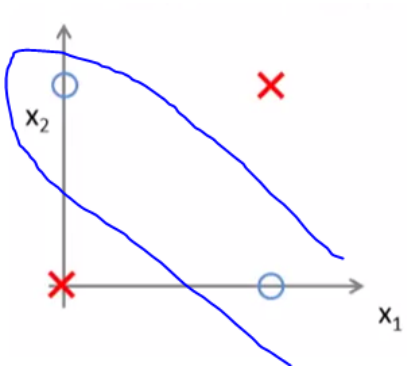

Figure 4: Requirement of non-linear separator for XOR problem.

6 Backpropagation

The question is, of course, how one might come up with a set of weight values that ensure the network produces the expected output. In practice, trying to find an acceptable set of weights for an MLP network manually would be an incredibly laborious task. In fact, it is NP-complete (Blum and Rivest, 1992). However, it is, fortunately, possible to learn a good set of weight values automatically through a process known as backpropagation. This was first demonstrated to work well for the XOR problem by Rumelhart et al. (1985).

The backpropagation algorithm begins by comparing the actual value output by the forward propagation process to the expected value and then moves backward through the network, slightly adjusting each of the weights in a direction that reduces the size of the error by a small degree. Both forward and back propagation are re-run thousands of times on each input combination until the network can accurately predict the expected output of the possible inputs using forward propagation.

For the XOR problem, 100% of possible data examples are available to use in the training process. We can therefore expect the trained network to be 100% accurate in its predictions and there is no need to be concerned with issues such as bias and variance in the resulting model.

7 Training a Neural Network to Compute 'XOR' in scikit-learn

First we import relevant libraries for implementing neural networks.

```
1 import numpy as np
2 import sklearn.neural_network
```

There are only four cases. Certainly, it seems like overkill to train a neural network to compute such a simple function, but since it is simple, it should be fairly easy to do.

```
xs = np.array([
    0, 0,
    0, 1,
    1, 0,
    1, 1
]).reshape(4, 2)

ys = np.array([0, 1, 1, 0]).reshape(4,)
```

We can handcode a neural network for 'XOR' with 3 neurons in the first hidden layer. However, to make our life easy, we would like to see if scikit-learn can find us that neural network that works to perform the classification task.

```
model = sklearn.neural_network.MLPClassifier(
    activation='logistic', max_iter=10000,
    learning_rate_init = 0.0001, hidden_layer_sizes=(3))
model.fit(xs, ys) # Training with the train data and labels
```

Output:

```
MLPClassifier(activation='logistic', hidden_layer_sizes=3, max_iter=10000)
```

However, it does not seem like scikit-learn's MLPClassifier can converge to a correct solution.

```
print('score:', model.score(xs, ys))
print('predictions:', model.predict(xs))
print('expected:', np.array([0, 1, 1, 0]))
```

Output:

```
score: 0.5
predictions: [0 0 0 0]
expected: [0 1 1 0]
```

What could be the possible reasons? The sklearn Multilayer Perceptron manual can be found in the following link.

`https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html`

You are required to conduct experiments to find out the possible reasons.

8 Lab Exercises

Please code yourself and write a report based on your findings:

1. Experiment with different activation functions and report your results.
2. Experiment with different learning rates and report your results.
3. Experiment with a different number of iterations and report your results.
4. Experiment with different numbers of layers and the number of neurons to find your solutions and report your results.