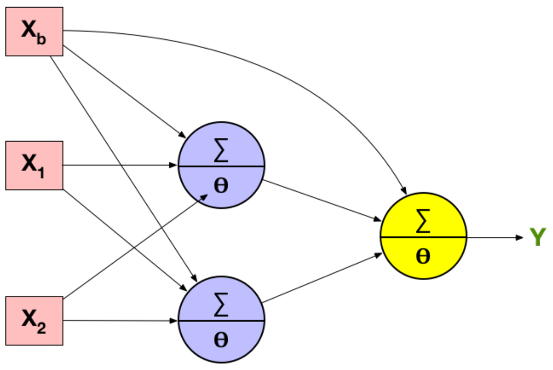
Department of Computer Science & Applied Physics

# *Learning XOR with A Multilayer Perceptron*

## Overview

A single-layer perceptron is only capable of learning a linear-separable function. While logical OR and AND are linear-separable, the XOR operation requires an additional layer to represent the extra dimension needed for the computation. In this lab we will use the following design to create a Multilayer Perceptron (MLP) capable of learning the XOR rule. The 3-network consists of three input nodes, a hidden layer of two nodes and a single-node output layer.



While the MLP acronym is often used for the topology shown above, it is an inaccurate description of the network as the nodes use a sigmoidal activation function, as opposed to the step function of the Rosenblatt perceptron. Indeed, because of the sigmoidal activation function, the network above is more consistent with the original neurons described by McCulloch and Pitts. Note that it is not possible to simulate a perfect logical gate with a sigmoidal output neuron because the range is [0,1], but it can be approximated. Even a simple neural network (NN) like the one shown above will require a large number of epochs before the actual and expected outputs converge.

## Exercises

- Download **Aicme4j** and add the JAR to the *Module-Path* of a new project.

- Specify the following elements as the training set (data and expected values):

```
double[][] data = {{0, 0}, {1, 0}, {0, 1}, {1, 1}};
double[][] expected = {{0}, {1}, {1}, {0}};
```

- Create a new class called *XORRunner*. Use the *NetworkBuilderFactory* class to get a handle on a builder to create the following network topography:

```
NeuralNetwork net = NetworkBuilderFactory.getInstance()
               .newNetworkBuilder()
               .inputLayer("Input", 2)
               .hiddenLayer("Hidden1", Activation.TANH, LayerSize.SUM)
               .outputLayer("Output", Activation.TANH, 1, false)
               .train(data, expected, 0.01, 0.95, 100000, 0.00001, Loss.SSE)
```

```
                    .save("./xor.data")
                    .build();
```

Note the large number of epochs specified. This is necessary as the TANH activation function represents a continuum of values, as opposed to the discrete step function. If the neural network output converges with the training data before the maximum number of epochs is reached, then the training will terminate early.

- Create the following four data sets required to test if the network is fully trained. These correspond to the truth table permutations for the two operands of the XOR operator.

```
double[] test1 = {0.0, 0.0};
double[] test2 = {1.0, 0.0};
double[] test3 = {0.0, 1.0};
double[] test4 = {1.0, 1.0};

out.println("00=>" + net.process(test1, Output.NUMERIC_ROUNDED));
out.println("10=>" + net.process(test2, Output.NUMERIC_ROUNDED))
out.println("01=>" + net.process(test3, Output.NUMERIC_ROUNDED));
out.println("11=>" + net.process(test4, Output.NUMERIC_ROUNDED));
```

- Execute the test data a number of times to see if the network is stable. A stable neural network consistently returns a result that is consistent with the training data.

- Progressively increase the learning rate to {0.05, 0.1, 0.2, 0.3, 0.4, 0.5} and examine the effect that this has on the output.

- Progressively reduce the number of epochs to {500,000, 100,000, 50,000, 10,000, 1000, 500, 100} and examine the effect that this has on the output.

- Change the activation function to a hyperbolic tangent function and examine the effect that this has on the output.

- What activation functions work best with this type of data?

- Should the data be standardised or normalised?

- The save() method persists the neural network configuration and weights to disk. Use the following method to load a trained model>

  ```
  NeuralNetwork net = Aicme4jUtils.load("./xor.data");
  ```

  What is the benefit of saving a neural network?

- Read through the JavaDocs for Aicme4j and familiarise yourself with the API. Examine the source code and make sure you understand the basic mechanics of a neural network.