

# Java Project: Part 1

CMPS 450.2: Fall 2001

Dr. Maida

September 18, 2001

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Supervised learning . . . . .	2
1.2	How does the network work? . . . . .	2
1.2.1	Feedforward operation . . . . .	2
	How a neuron works . . . . .	2
	A network of neurons . . . . .	3
1.2.2	Weight-adjustment mode . . . . .	3
1.3	Training procedure . . . . .	5
<b>2</b>	<b>Design specifications</b>	<b>6</b>
2.1	The main class . . . . .	6
2.2	The pattern class . . . . .	6
2.3	Neuron classes . . . . .	6
2.4	Network weight initialization . . . . .	7

## 1 Introduction

The Java project for CMPS 450G is to write an artificial neural network program which can be trained to recognize simple two-input boolean patterns such as AND, OR, and XOR. You will implement a neural network consisting of three neurons arranged into two layers. The architecture is shown in Figure 1. The solid circles represent neurons. There are two neurons in the hidden layer and one neuron in the output layer. The arrows connecting the neurons show the feedforward flow of information from the input to the output. The arrows also represent adjustable weights in the network. By convention, the input layer is not counted as a layer because it does not have neurons with trainable weights. The input layer holds the input features for the pattern that the network is currently processing.

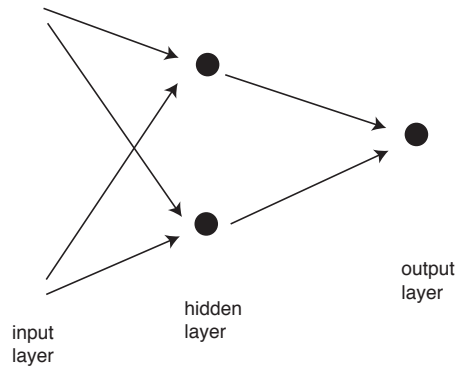


Figure 1: An artificial neuron with two inputs. The errors represent information flow and adjustable weights. The weights and biases are adjustable. Each neuron has a bias which is not shown.

---

## 1.1 Supervised learning

This network will be trained by a method known as supervised learning. The program will present a pattern to the network and then look at the network's output. The output is given by the output value of the neuron in the output layer. This neuron can have output values between 0 and 1, where 0 means "no" or "false" and 1 means "yes" or "true". Numbers in between represent a graded response. For example, consider the task of training the network to recognize the boolean pattern AND. The pattern consists of two input features and the objective is to teach the network to output "yes" whenever both of its inputs are 1 and to output "no" otherwise. The program will tell the network whenever it makes a mistake. When the network makes a mistake, it will adjust its weights to reduce the severity of that mistake in the future. After many trials, the amount of error that the network makes will become very low.

## 1.2 How does the network work?

The network has two modes of operation. The first mode is *feedforward mode* and the second mode is *weight-adjustment mode*.

### 1.2.1 Feedforward operation

In feedforward mode, the network is presented with a pattern, and the network computes its output response. To understand how feedforward mode works, we need to look at the structure of the neurons that are in the network.

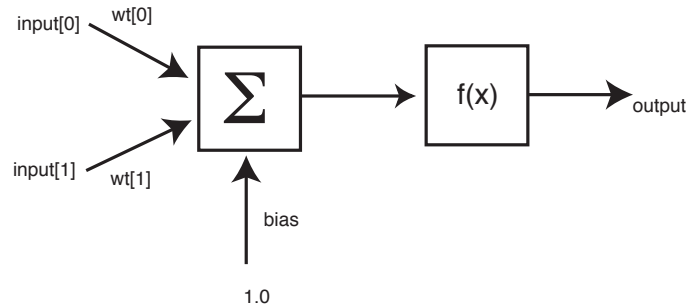


Figure 2: An artificial neuron with two inputs. The weights and bias are adjustable.

**How a neuron works** All of the neurons have the same structure. Figure 2 shows a neuron in the network. The neuron's output is given by the formula below.

$$f(wt[0] * input[0] + wt[1] * input[1] + bias)$$

The formula says to take the inputs of the neuron, multiply them by their associated weights and then add a bias. Take this result and run it through the function  $f$ . This function is defined below. The function produces values in the range (0.0, 1.0).

$$f(x) = \frac{1}{1 + e^{-4x}}$$

The function is called the *sigmoidal output function* and since it returns values between 0 and 1 for any input  $x$ , we can be sure that the neuron always outputs a value between 0 and 1.

**A network of neurons** Now that we know how a neuron works, we can come back to the question of how the network computes its output. Each neuron in the network looks at its input and then produces an output. The output layer gets its input from the hidden layer, so we have to make sure that the hidden layer neurons have produced their values before the output unit computes its values. Looking back at the network architecture, there are two neurons in the hidden layer. Each of these neurons will compute an output as described above. Once this is done, the neuron in the output layer of the network can treat these neurons as input to compute its own outputs. This is shown in the pseudo-code below.

```
double feedForward (pattern)
    load pattern into the hidden unit inputs;
    compute output for each hidden unit;
    compute output for the output unit;
    return output value for the output unit;
```

### 1.2.2 Weight-adjustment mode

Weight-adjustment occurs when the network makes a mistake. A network makes a mistake when it produces an incorrect response to a given input pattern. To make this more concrete, let us look at the structure of a set of input patterns which encode the boolean concept AND. It consists of four patterns. Each pattern consists of two input features and a desired output.

pattern number	first feature	second feature	desired output
1.	0	0	0
2.	0	1	0
3.	1	0	0
4.	1	1	1

To train the network, we present it with an input pattern and then look at the network's output. This is called the *actual* or *observed* output. We then compare the network's observed output with the desired output for that particular pattern. The desired output is either 0 or 1. Recall, that the network gives an output in the range between 0 and 1. To judge the extent that the network made an error, we look at the quantity  $E_2$ .  $E_2$  denotes the error for the output unit (layer 2), and is defined below.

$$E_2 = \text{desired} - \text{observed}$$

If this quantity is zero, then the network did not make an error and we do not have to adjust the weights. However, this quantity never actually reaches zero; since the sigmoidal output function never reaches zero or one, the network cannot produce output that is exactly zero or one.

Once the output error,  $E_2$  is known, each of the incoming weights for the output neuron are adjusted according to the formula below.

$$\Delta w_{1,2} = r o_1 s_2 E_2$$

The symbol  $\Delta w_{1,2}$  denotes the amount of weight adjustment on the weight that connects a neuron in Layer 1 to the neuron in Layer 2. The formula says that the amount of weight adjustment is the product of four quantities. These are a rate parameter,  $r$ , the output of the neuron in the hidden layer (layer 1),  $o_1$ , the slope of the output of the neuron in the output layer,  $s_2$ , and the output error  $E_2$ . For this project use a value of 0.125 for the rate parameter  $r$ . If you know the value of the output unit, then its slope is easy to compute. This is because the derivative of the sigmoidal function,  $f(x)$ , is equal to  $4f'(x)(1 - f'(x))$ . Therefore,  $s_2 = 4o_2(1 - o_2)$ , where  $o_2$  is the output value of the output unit.

This tells us how to adjust the weights in Layer 2. We also need to adjust the bias weight. If you look at Figure 2, you will see that a bias is a weight whose input value is always clamped to 1. Therefore, we can use the above weight adjustment formula with  $o_1$  set to 1.

We also need to adjust the weights in Layer 1. The procedure is identical provided that we know the error for the neuron whose incoming weights we want to adjust. The error,  $E_1$ , for a neuron in Layer 1 is given in the formula below.

$$E_1 = w_{1,2} s_2 E_2$$

Notice that the value of  $E_1$  depends on the value of  $E_2$ . This means that we have to compute the error term for the output neuron before we compute the error terms for neurons in the hidden layer. Because of this requirement, the neural network training procedure is called *backpropagation of errors*.

The last back propagation formula is given below. This formula specifies how to adjust the incoming weights to each of the hidden neurons.

$$\Delta w_{0,1} = r i s_1 E_1$$

The symbol  $i$  denotes the value of the input pattern to the weight being adjusted. The symbol  $s_1$  denotes the slope of the hidden unit. The formula can be used to adjust the bias by assuming that the value of  $i$  is one.

### 1.3 Training procedure

Training proceeds in units of time called epochs. To train the network for one epoch, you proceed as follows. First, you randomly select one of the input patterns and present it to the network in feedforward mode. You then use the output error to adjust all of the weights in the network. You repeat this process until the network has seen all of the training patterns. This constitutes one epoch of training.

Typically, it is necessary to train the network for hundreds or even thousands of epochs to get satisfactory performance. To judge how well the network is learning, we need a measure of its performance. To do this, we will record the sum-of-squared error for each epoch. To do this, you need to remember the output units, output error  $E_2$  for each of the four patterns. Then SSE for an epoch is given in the formula below.

$$SSE = \sum_{p=0}^3 (E_{2,p})^2$$

The symbol  $E_{2,p}$  means  $E_2$  for pattern  $p$ . You should train the network until  $SSE$  becomes very close to zero.

## 2 Design specifications

Your project is to build the neural network shown in Figure 1 and train it to learn the boolean concepts AND, OR, and XOR.

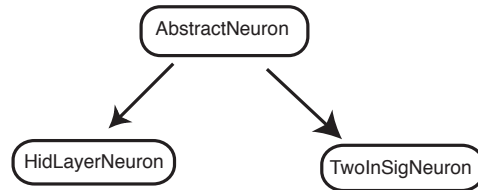


Figure 3: The neuron class hierarchy.

---

## 2.1 The main class

The main class in the network should be called `TrainingEngine`. Its code is given below.

```
class TrainingEngine {
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        System.out.println("args[0]: "+n);
        TwoLayerNet net = new TwoLayerNet();
        net.trainNepochs(n);}}}
```

After compiling, you run the program by typing “`java TrainingEngine n`”, where  $n$  is the number of epochs that you want to train the network for.

## 2.2 The pattern class

You will need a class called `Pattern` to represent an instance of a training pattern. It will have three fields: `feature1`, `feature2`, and `desiredOutput`.

## 2.3 Neuron classes

The primary data object in the program will be the neuron as shown in Figure 2. The neuron class hierarchy is shown in Figure 3. There will be two kinds of neurons, corresponding to the classes `HidLayerNeuron` and `TwoInSigNeuron`. A hidden-layer neuron differs from a two-input sigmoidal neuron in the kind of inputs it takes. The hidden-layer neurons takes pattern feature values as input. The two-input sigmoidal neuron takes hidden-layer neurons as input. These classes will have properties in common and be linked together by the class `AbstractNeuron`. The fields in class `AbstractNeuron` are:

field name	data type
rate	static double
bias	double
output	double
error	double
wts	double[]

The class AbstractNeuron should have at least the methods in the table below.

return type	name and args	purpose
double	getActivation()	returns output
double	getError()	returns error
void	resetError()	sets error to 0.0
double[]	getWeights()	returns the array of weights
double	getBias()	returns the bias
double	getDerivative()	returns $4.0 * (\text{output} * (1.0 - \text{output}))$
void	setBias(double val)	sets bias to val
void	setErr(double val)	sets error to val
void	incErr(double val)	increments the value of error by value

## 2.4 Network weight initialization

Before you start training the network, you should initialize the weights to random values uniformly distributed between -0.5 and 0.5.