Soft Computing Laboratory

Task number 1

Multilayer Perceptron

Florian TISSIER, number 901415
Geoffrey RAMDE, number 901414

1 The main goal

This rapport documents the implementation of a Multilayer Perceptron (MP). The MP is similar to neural networks such as brains. It can:

- Learn by getting more information
- Solve problems by changing the weights of its neurons.

Neural networks are used to solve problems that can't be solved with a traditional algorithm. We wish to have our MP learn simple patterns.

2 Theoretical background

Our MP is composed of 3 layers:

- Input layer with 4 neurons
- Hidden layer with 2 neurons
- Output layer with 4 neurons

The MP is entirely written in C++.

2.1 Training Pattern

We can use for example 4 trainings patterns and try to train our multilayer perceptron. The trainings patterns can be for example:

([1,0,0,0],[1,0,0,0]),

([0,1,0,0],[0,1,0,0]),

([0,0,1,0],[0,0,1,0]),

([0,0,0,1],[0,0,0,1])

The first list corresponds to the values that passed in input and the second one is the list we expected as output. So for example we put [1,0,0,0] in input and after the learning process, the perceptron should be able to recognize this pattern and we should get it back in output. And so on for the 3 other training patterns.

2.2 Activation Function

We compute the ouput value by using an activation function for each neuron. We implement 2 differents activation functions:

Identity function: $f(x) = x$
Sigmoid function: $f(x) = 1/(1 + e^{-\beta x})$

2.3 Backpropagation Method

The backpropagation method is used to train our MP. It consists of 7 steps:

1. random weights are chosen for each neuron into the network, $w \in [-0:5; 0:5]$
2. we choose a random training pattern and propagate it into the network until every output values are computed
3. errors values are calculated for each neuron in the last layer then we for the hidden layers
4. change of weights for each neuron
5. we test if we trained every training pattern. If not we return to step 1
6. we test is the network have been well trained, we check if we get expected results. If not we returned to step 1
7. the method is over

3 Experiments

3.1 Experiment 1 : description

In the experiment 1 the network was trained with the following patterns:

([1,0,0,0],[1,0,0,0]),

([0,1,0,0],[0,1,0,0]),

3.2 Experiment 2 : description

In the experiment 2 the network was trained with an other set of patterns:

([1,0,0,0],[1,0,0,0]),

([0,1,0,0],[0,1,0,0]),

([0,0,1,0],[0,0,1,0]),

([0,0,0,1],[0,0,0,1])

3.2.1 Experiment 2 without bias neuron

We tried to implement the MP without a bias neuron but the results were inconclusive. Only 2 patterns can be trained.

3.2.2 Experiment 2 with bias neuron

By adding a Bias Neuron to the first and second layer, we add an input to the second and the third layer, in that way the network is able to compute results. We also need to add an additional weight to each neuron in the output and the hidden layer.

4 Results

4.1 Experiment 1

Our results:

Expected	Neuron 1	Neuron 2	Neuron 3	Neuron 4
[1,0,0,0]	0.9915	0.0085	0.0079	0.0079
[0,1,0,0]	0.0086	0.9913	0.0076	0.0076

Rounded values:

Expected	Neuron 1	Neuron 2	Neuron 3	Neuron 4
[1,0,0,0]	1	0	0	0
[0,1,0,0]	0	1	0	0

The MP is able to learn 2 different patterns without any trouble.

4.2 Experiment 2

4.2.1 Experiment 2 without bias neuron

Our results:

Expected	Neuron 1	Neuron 2	Neuron 3	Neuron 4
[1,0,0,0]	0.3777	0.079	0.079	0.3778
[0,1,0,0]	0.0801	0.9640	1.55E-12	0.801
[0,0,1,0]	0.0800	1.46E-12	0.9639	0.0800
[0,0,0,1]	0.3777	0.796	0.0797	0.3777

Rounded values:

Expected	Neuron 1	Neuron 2	Neuron 3	Neuron 4
[1,0,0,0]	0.4	0	0	0.4
[0,1,0,0]	0	1	0	0
[0,0,1,0]	0	0	1	0
[0,0,0,1]	0.4	0	0	0.4

The MP is unable to compute the correct answers. Only 2 patterns are correct. 2 neurons in the hidden layer is insufficient.

4.2.2 Output of the hidden layer

Values:

Input	Neuron 1	Neuron 2
[1,0,0,0]	0.1028	0.1031
[0,1,0,0]	0.0101	0.9974
[0,0,1,0]	0.9974	0.0109
[0,0,0,1]	0.1029	0.1031

Rounded values:

Expected	Neuron 1	Neuron 2
[1,0,0,0]	0.1	0.1
[0,1,0,0]	1	1
[0,0,1,0]	1	0
[0,0,0,1]	0.1	0.1

Pattern output of the hiden layer:

[1,0,0,0]: [0,0]

[0,1,0,0]: [1,1]

[0,0,1,0]: [1,0]

[0,0,0,1]: [0,0]

4.2.3 Experiment 2 with bias neuron

Our results:

Expected	Neuron 1	Neuron 2	Neuron 3	Neuron 4
[1,0,0,0]	0.9832	3.2E-6	0.0183	0.016
[0,1,0,0]	2.70E-6	0.9834	0.0181	0.0116
[0,0,1,0]	0.0116	0.0116	0.9783	1.93E-6
[0,0,0,1]	0.0146	0.0144	1.00E-5	0.9815

Rounded values:

Expected	Neuron 1	Neuron 2	Neuron 3	Neuron 4
[1,0,0,0]	1	0	0	0
[0,1,0,0]	0	1	0	0
[0,0,1,0]	0	0	1	0
[0,0,0,1]	0	0	0	1

The MP is now working correctly and is ablle to learn 4 patterns.

4.2.4 Output of the hidden layer

Values:

Expected	Neuron 1	Neuron 2
[1,0,0,0]	0.0091	0.0098
[0,1,0,0]	0.0087	0.9732
[0,0,1,0]	0.9901	0.9907
[0,0,0,1]	0.9753	0.0082

Rounded values:

Expected	Neuron 1	Neuron 2
[1,0,0,0]	0	0
[0,1,0,0]	0	1
[0,0,1,0]	1	1
[0,0,0,1]	1	0

5 The role of the BIAS Neuron

Why do we need a BIAS neuron to make the MLP works?
We can see that the outputs of the hidden layer are binary values. These values will be used to calculate the weighted sum for the sigmoid function.
But we will encounter a problem because of the [0,0] values returned by the hidden layer.
For example, if we take the first pattern [1,0,0,0], the first neuron of the output layer should return 1. This is only possible if the weights for this neuron are really big.
But when we will want to train the other patterns, this time the first output should be 0 not 1, so to make it possible, the weights of this neuron should be small.
We arrive at a contradiction: at the same time, the weights should be big and small, which is obviously impossible and the MLP can not decide.

This is where the BIAS neuron is useful. It will allow us to shift the sigmoid function on the x-axis to the left or to the right depending on the BIAS weight of each neuron.
Thus, we will have small weights which will allow us to obtain 0 as the output of the first neuron of the output layer for all pattern except the first one, and thanks to the weight of the BIAS neuron, which will shift the sigmoid function, with this same small weights we will be able to obtain 1 as the first output for the first pattern.

6 Summary and conclusions

We have succeeded in the implementation of a Multilayer Perceptron with C++ and we have understood how it is working, with and without BIAS neuron.
Despite our lack of experience in the subject, the concise explainations of the lectures allowed us to do our work without any big difficulty.