Erasmus Mundus
master course in
Data Mining and
Knowledge Management
a european master

Laboratory assignment report for **Optimization**

Chapter 1: Optimization of a neural network

HE Yue

RABEARISON Falitokiniaina

October 2013 - Université Lumière Lyon2

Optimization of a Neural Network

1. Problem Description

Neural network is a type of algorithms that are used to "learn". It learns from existing data, extracts a general model for this dataset. Then the model based on the Neural Network can be used in some classifying or forecasting fields. In this assignment, we use the neural network to do the pattern recognition. The neural network model is actually determined by its parameters. In order to compute a set of "good" parameters we formulate an optimization model in this report and verify the method is fitness.

In conclusion, the objective of this assignment is to find the parameters of the neural network (also named network calibration) by using optimization techniques (in this case, we use the AMPL package to solve the problem).

2. Model Formulation

In this report, we use the simple neural network as shown in the Fig. 2.1. The neural network consists of a set of nodes, which transmit information in the direction indicated by the arcs. In this case, we have seven nodes organized in three layers. The lower layer (the first layer) receives four input values (x1, x2, x3, x4), while the output layer (the third layer) gives us the result of the network, F(x1, x2, x3, x4).

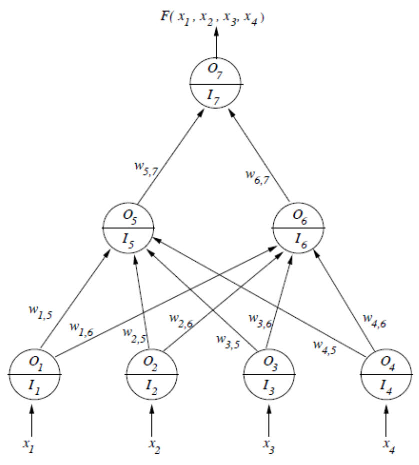


Fig. 2.1. Structure of neural network

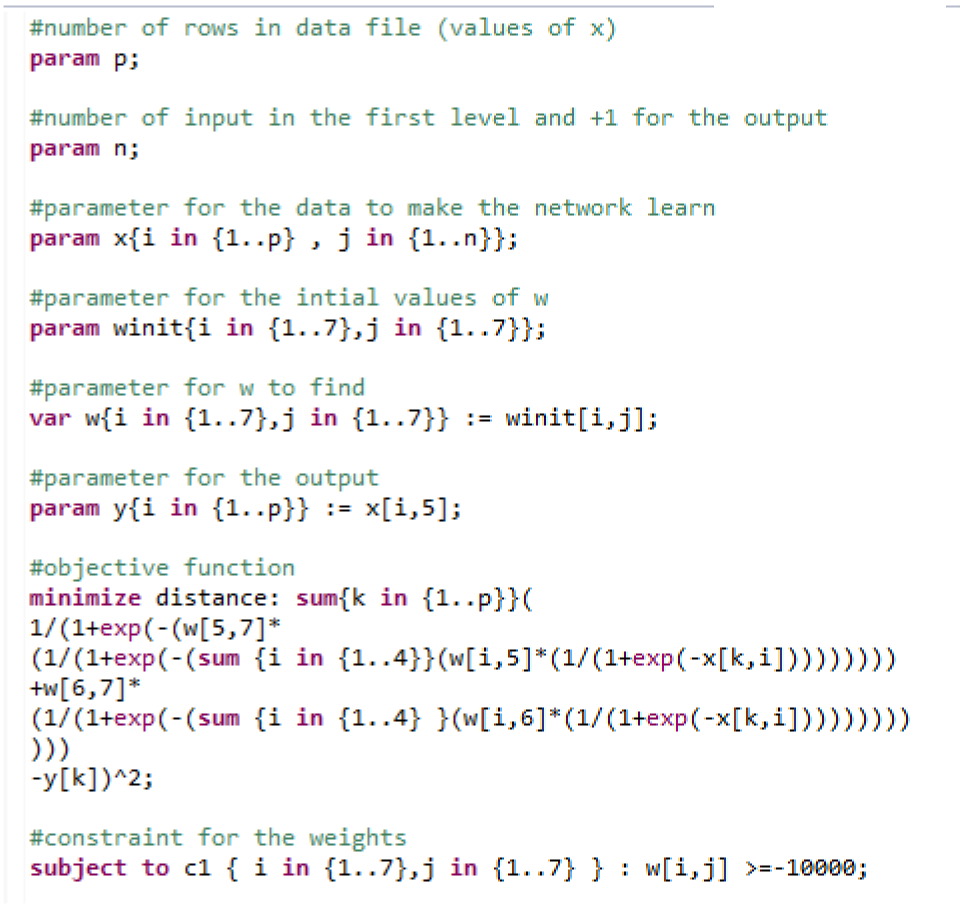
The network works in the following way. Each node i receives an input, Ii, and transforms it into an output, Oi. Here the transformation is done through the sigmoid function $f(x) = \frac{1}{1+e^{-x}}$ (as required in this assignment). So the values of a neuron's output Oi can be calculated as Oi = f(Ii).

In the neural network the INPUTS of following layer is obtained adding up the OUTPUTS of the previous layer after being multiplied by the weights values, $w_{j,i}$. Based on the above theory, we have the result of this network: $F(x_1, x_2, x_3, x_4) = O_7 = f(I_7) = f(w_{5,7} * O_5 + w_{6,7} * O_6) = f(w_{5,7} * f(\sum_{i=1}^{4} w_{i,5} * f(x_i)) + w_{6,7} * f(\sum_{i=1}^{4} w_{i,6} * f(x_i)))$. How to compute the "good" weight is the most import process in the model formulation. Our idea is to minimize the overall distance between predicted values and real outputs.

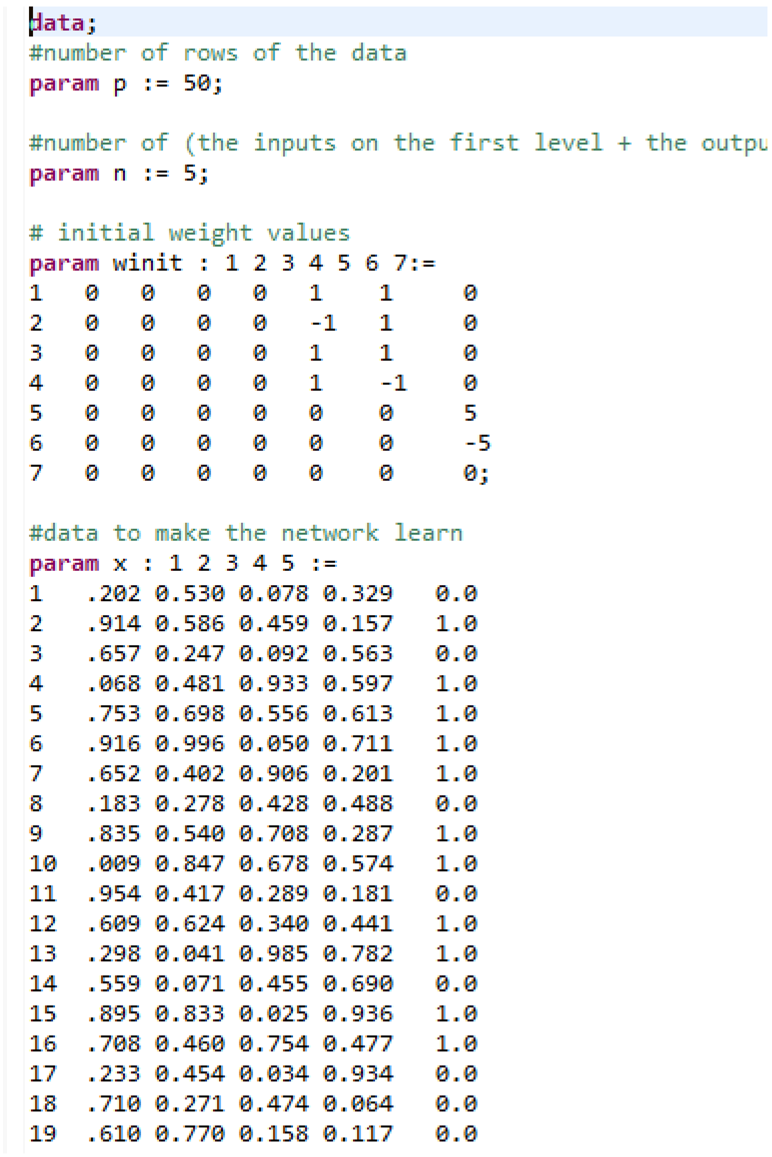
At last, we get the model of this neural network as following, $\min_w \sum_{i=1}^{p} (F(x_{e_i}; w) - x_{o_i})^2$

3. AMPL Experiment and Testing

In order to perform the “learning” of the neural network, we have a 4-input vectors with their corresponding outputs. With the help of GENXNDAT, we obtain a dataset, which size is 50. The first four columns of data are inputs and the last one is the output. We use the AMPL software to solve this optimization assignment of above neural network. We create two files: the model file (assign1.mod) and the data file (assign1_dat2.dat). The codes are as following.

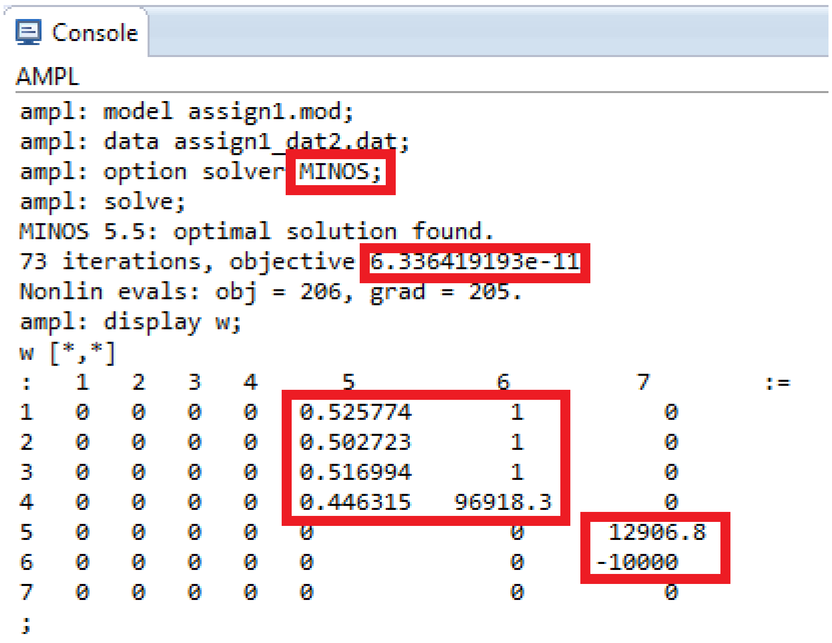


```
#number of rows in data file (values of x)
param p;

#number of input in the first level and +1 for the output
param n;

#parameter for the data to make the network learn
param x{i in {1..p} , j in {1..n}};

#parameter for the intial values of w
param winit{i in {1..7},j in {1..7}};

#parameter for w to find
var w{i in {1..7},j in {1..7}} := winit[i,j];

#parameter for the output
param y{i in {1..p}} := x[i,5];

#objective function
minimize distance: sum{k in {1..p}}(
1/(1+exp(-(w[5,7]*
(1/(1+exp(-(sum {i in {1..4}}(w[i,5]*(1/(1+exp(-x[k,i]))))))))
+w[6,7]*
(1/(1+exp(-(sum {i in {1..4} }(w[i,6]*(1/(1+exp(-x[k,i]))))))))
)))
-y[k])^2;

#constraint for the weights
subject to c1 { i in {1..7},j in {1..7} } : w[i,j] >=-10000;
```



```
data;
#number of rows of the data
param p := 50;

#number of (the inputs on the first level + the outpu
param n := 5;

# initial weight values
param winit : 1 2 3 4 5 6 7:=
1    0    0    0    0    1    1     0
2    0    0    0    0   -1    1     0
3    0    0    0    0    1    1     0
4    0    0    0    0    1   -1     0
5    0    0    0    0    0    0     5
6    0    0    0    0    0    0    -5
7    0    0    0    0    0    0     0;

#data to make the network learn
param x : 1 2 3 4 5 :=
1    .202 0.530 0.078 0.329   0.0
2    .914 0.586 0.459 0.157   1.0
3    .657 0.247 0.092 0.563   0.0
4    .068 0.481 0.933 0.597   1.0
5    .753 0.698 0.556 0.613   1.0
6    .916 0.996 0.050 0.711   1.0
7    .652 0.402 0.906 0.201   1.0
8    .183 0.278 0.428 0.488   0.0
9    .835 0.540 0.708 0.287   1.0
10   .009 0.847 0.678 0.574   1.0
11   .954 0.417 0.289 0.181   0.0
12   .609 0.624 0.340 0.441   1.0
13   .298 0.041 0.985 0.782   1.0
14   .559 0.071 0.455 0.690   0.0
15   .895 0.833 0.025 0.936   1.0
16   .708 0.460 0.754 0.477   1.0
17   .233 0.454 0.034 0.934   0.0
18   .710 0.271 0.474 0.064   0.0
19   .610 0.770 0.158 0.117   0.0
```

Here is a screenshot of the result. We put in red frame the name of the solver which is MINOS, the result of objective function and the obtained weights w.

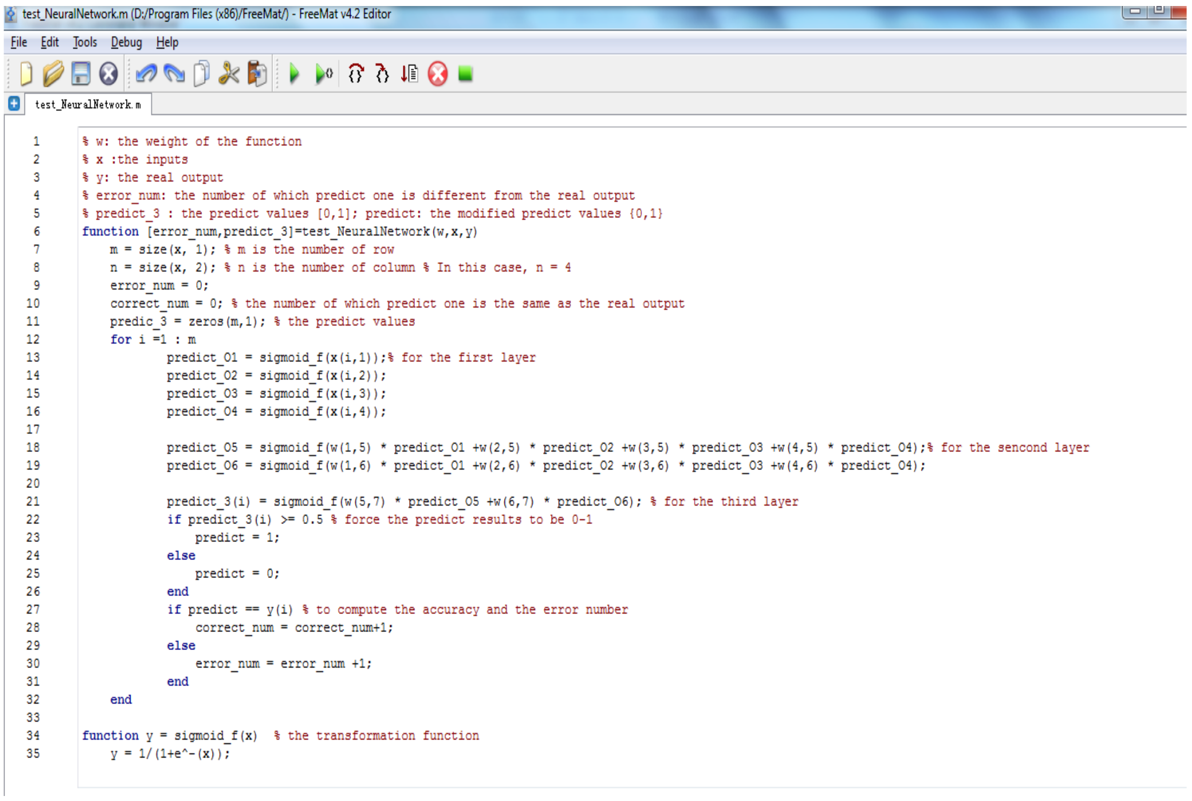


Console

AMPL

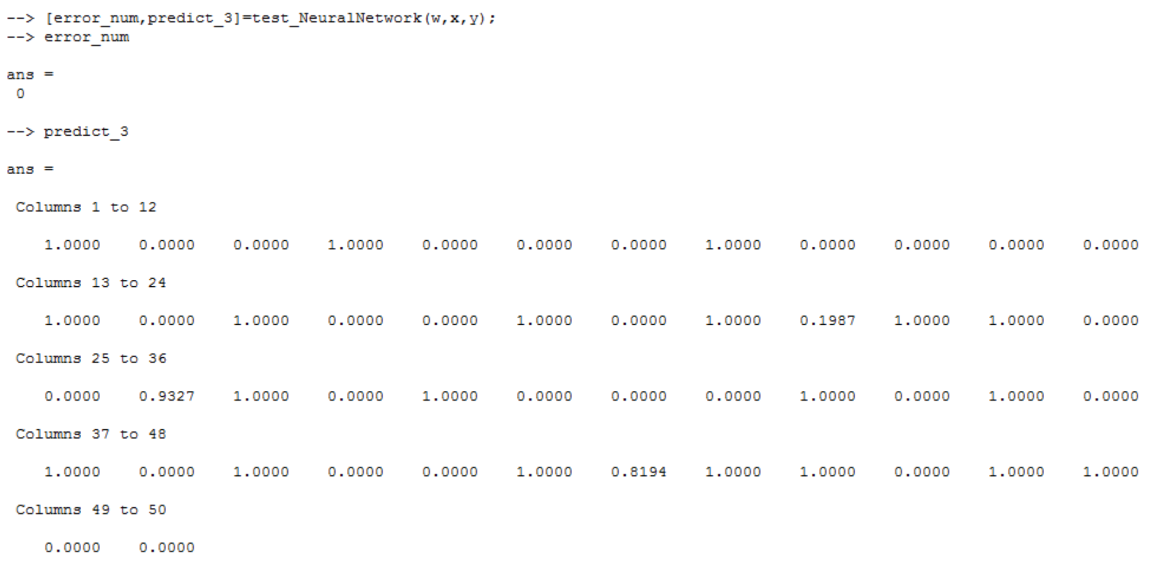
```
ampl: model assign1.mod;
ampl: data assign1_dat2.dat;
ampl: option solver MINOS;
ampl: solve;
MINOS 5.5: optimal solution found.
73 iterations, objective 6.336419193e-11
Nonlin evals: obj = 206, grad = 205.
ampl: display w;
w [*,*]
:   1   2   3   4      5          6          7         :=
1   0   0   0   0   0.525774      1          0
2   0   0   0   0   0.502723      1          0
3   0   0   0   0   0.516994      1          0
4   0   0   0   0   0.446315   96918.3       0
5   0   0   0   0   0             0      12906.8
6   0   0   0   0   0             0     -10000
7   0   0   0   0   0             0          0
;
```

Once the “learning” has been performed, it is necessary to verify that the neural network works properly. So we generate another 50-size of dataset to test whether the predict values are the same as the real ones. In the test part, we wrote the code in Freemat.

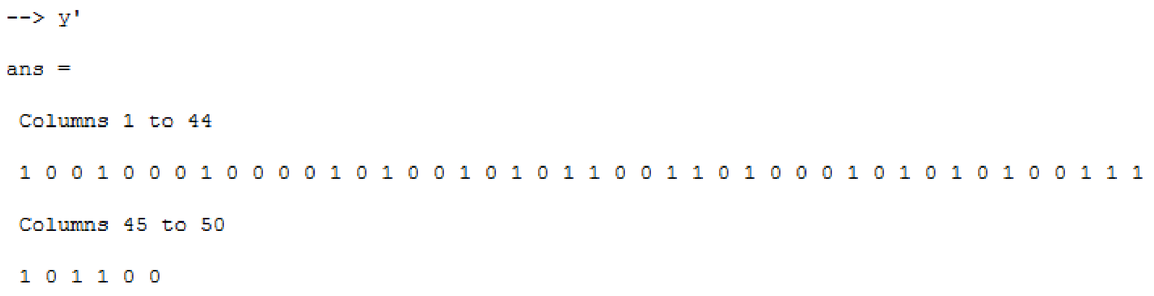

test_NeuralNetwork.m (D:/Program Files (x86)/FreeMat/) - FreeMat v4.2 Editor
File Edit Tools Debug Help
test_NeuralNetwork.m

```
% w: the weight of the function
% x :the inputs
% y: the real output
% error_num: the number of which predict one is different from the real output
% predict_3 : the predict values [0,1]; predict: the modified predict values {0,1}
function [error_num,predict_3]=test_NeuralNetwork(w,x,y)
    m = size(x, 1); % m is the number of row
    n = size(x, 2); % n is the number of column % In this case, n = 4
    error_num = 0;
    correct_num = 0; % the number of which predict one is the same as the real output
    predic_3 = zeros(m,1); % the predict values
    for i =1 : m
            predict_01 = sigmoid_f(x(i,1));% for the first layer
            predict_02 = sigmoid_f(x(i,2));
            predict_03 = sigmoid_f(x(i,3));
            predict_04 = sigmoid_f(x(i,4));

            predict_05 = sigmoid_f(w(1,5) * predict_01 +w(2,5) * predict_02 +w(3,5) * predict_03 +w(4,5) * predict_04);% for the sencond layer
            predict_06 = sigmoid_f(w(1,6) * predict_01 +w(2,6) * predict_02 +w(3,6) * predict_03 +w(4,6) * predict_04);

            predict_3(i) = sigmoid_f(w(5,7) * predict_05 +w(6,7) * predict_06); % for the third layer
            if predict_3(i) >= 0.5 % force the predict results to be 0-1
                predict = 1;
            else
                predict = 0;
            end
            if predict == y(i) % to compute the accuracy and the error number
                correct_num = correct_num+1;
            else
                error_num = error_num +1;
            end
    end

function y = sigmoid_f(x)  % the transformation function
    y = 1/(1+e^-(x));
```

The result of the testing is “all of the data can be predicted correctly”.



```
--> [error_num,predict_3]=test_NeuralNetwork(w,x,y);
--> error_num

ans =
 0

--> predict_3

ans =

 Columns 1 to 12

    1.0000    0.0000    0.0000    1.0000    0.0000    0.0000    0.0000    1.0000    0.0000    0.0000    0.0000    0.0000

 Columns 13 to 24

    1.0000    0.0000    1.0000    0.0000    0.0000    1.0000    0.0000    1.0000    0.1987    1.0000    1.0000    0.0000

 Columns 25 to 36

    0.0000    0.9327    1.0000    0.0000    1.0000    0.0000    0.0000    0.0000    1.0000    0.0000    1.0000    0.0000

 Columns 37 to 48

    1.0000    0.0000    1.0000    0.0000    0.0000    1.0000    0.8194    1.0000    1.0000    0.0000    1.0000    1.0000

 Columns 49 to 50

    0.0000    0.0000
```



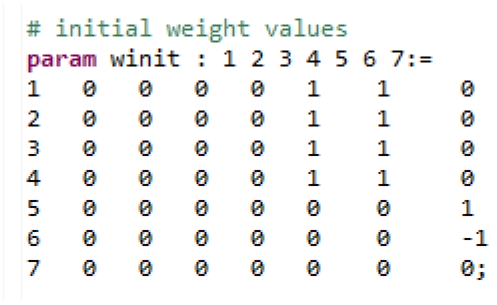
```
--> y'

ans =

 Columns 1 to 44

 1 0 0 1 0 0 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 1 1 0 0 1 1 0 1 0 0 0 1 0 1 0 1 0 1 0 0 1 1 1

 Columns 45 to 50

 1 0 1 1 0 0
```

4. Analysis and Results

The main problem in this assignment is to solve $\min_w \sum_{i=1}^{p}(F(x_{e_i};w)-x_{o_i})^2$, that is to find the corresponding weights $w_{1,5}, w_{1,6}, w_{2,5}, w_{2,6}, w_{3,5}, w_{3,6}, w_{4,5}, w_{4,6}, w_{5,7}, w_{6,7}$ for the neural network. However, we find that initializing the weights in different values and putting the different constraints of weights will lead to various optimization results.

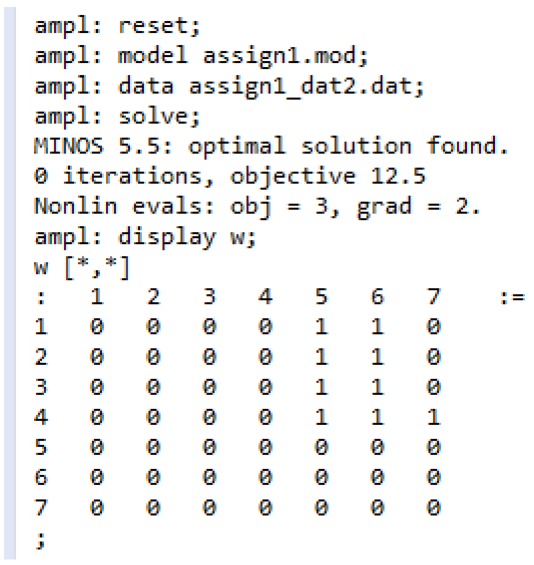
First, we define the lower bound of weight w and the initial w.

- imposing a fictitious lower bound on w (**subject to** c1 { i **in** {1..7},j **in** {1..7} } : w[i,j] >=0;).
- Initializing the weights :



```
# initial weight values
param winit : 1 2 3 4 5 6 7:=
1   0   0   0   0   1    1     0
2   0   0   0   0   1    1     0
3   0   0   0   0   1    1     0
4   0   0   0   0   1    1     0
5   0   0   0   0   0    0     1
6   0   0   0   0   0    0     -1
7   0   0   0   0   0    0     0;
```

Then we go to choosing solver. We find the only one that can solve the nonlinear problem from the solvers which are present with the student version of AMPL is **MINOS** which is the solver by default. (CPLEX is for global optimization, mixed integer linear programming; GUROBI is for linear programming)

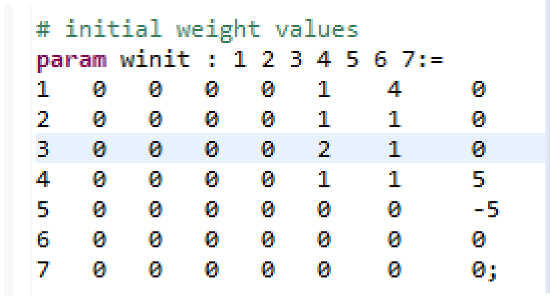


```
ampl: reset;
ampl: model assign1.mod;
ampl: data assign1_dat2.dat;
ampl: solve;
MINOS 5.5: optimal solution found.
0 iterations, objective 12.5
Nonlin evals: obj = 3, grad = 2.
ampl: display w;
w [*,*]
:   1   2   3   4   5   6   7    :=
1   0   0   0   0   1   1   0
2   0   0   0   0   1   1   0
3   0   0   0   0   1   1   0
4   0   0   0   0   1   1   1
5   0   0   0   0   0   0   0
6   0   0   0   0   0   0   0
7   0   0   0   0   0   0   0
;
```

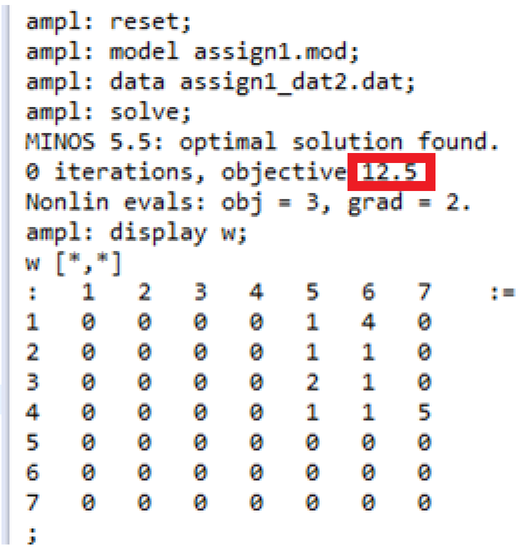
Unfortunately we got as result, an objective reslut very high, (objective = 12.5). Whereas here, the local minimal that we are looking for is a minimum close to 0, so that we can admit that there were few errors when solving the nonlinear least square problem. That last did not happen, so we can say that **the learning has not been satisfactory**.

To fix it, we go to **choose randomly** and **change the initial weights** that we gave.

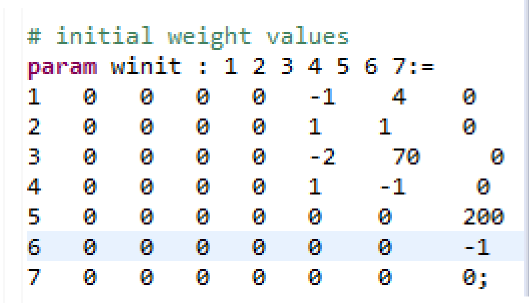
- With our constraint concerning w (w>=0), whatever we put as initial weights, the objective function stays at 12.5 or goes more and more high far from 0.



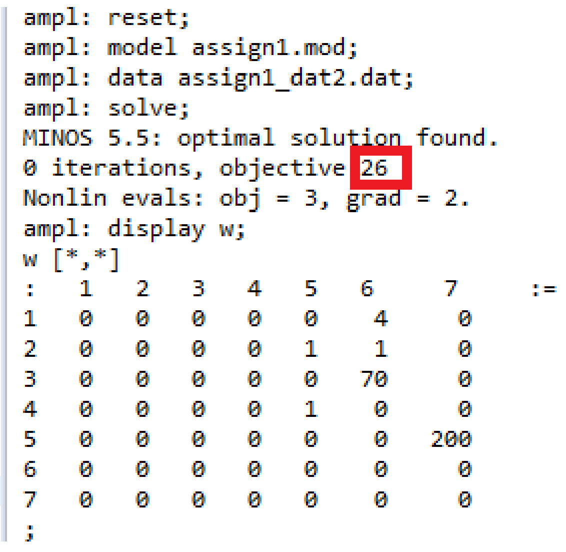
```
# initial weight values
param winit : 1 2 3 4 5 6 7:=
1   0   0   0   0   1    4     0
2   0   0   0   0   1    1     0
3   0   0   0   0   2    1     0
4   0   0   0   0   1    1     5
5   0   0   0   0   0    0     -5
6   0   0   0   0   0    0     0
7   0   0   0   0   0    0     0;
```



```
ampl: reset;
ampl: model assign1.mod;
ampl: data assign1_dat2.dat;
ampl: solve;
MINOS 5.5: optimal solution found.
0 iterations, objective 12.5
Nonlin evals: obj = 3, grad = 2.
ampl: display w;
w [*,*]
:   1   2   3   4   5   6   7    :=
1   0   0   0   0   1   4   0
2   0   0   0   0   1   1   0
3   0   0   0   0   2   1   0
4   0   0   0   0   1   1   5
5   0   0   0   0   0   0   0
6   0   0   0   0   0   0   0
7   0   0   0   0   0   0   0
;
```



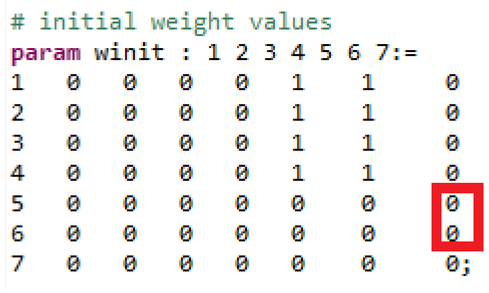
```
# initial weight values
param winit : 1 2 3 4 5 6 7:=
1   0   0   0   0   -1   4    0
2   0   0   0   0   1    1    0
3   0   0   0   0   -2    70    0
4   0   0   0   0   1    -1    0
5   0   0   0   0   0    0    200
6   0   0   0   0   0    0    -1
7   0   0   0   0   0    0    0;
```



```
ampl: reset;
ampl: model assign1.mod;
ampl: data assign1_dat2.dat;
ampl: solve;
MINOS 5.5: optimal solution found.
0 iterations, objective 26
Nonlin evals: obj = 3, grad = 2.
ampl: display w;
w [*,*]
:   1   2   3   4   5   6     7     :=
1   0   0   0   0   0    4     0
2   0   0   0   0   1    1     0
3   0   0   0   0   0   70     0
4   0   0   0   0   1    0     0
5   0   0   0   0   0    0   200
6   0   0   0   0   0    0     0
7   0   0   0   0   0    0     0
;
```

These are because the constraint does not allow the function to go lower.

- Initializing the weights with 0 is nonsense, as shown in the following figure.



```
# initial weight values
param winit : 1 2 3 4 5 6 7:=
1   0   0   0   0   1    1    0
2   0   0   0   0   1    1    0
3   0   0   0   0   1    1    0
4   0   0   0   0   1    1    0
5   0   0   0   0   0    0    0
6   0   0   0   0   0    0    0
7   0   0   0   0   0    0    0;
```

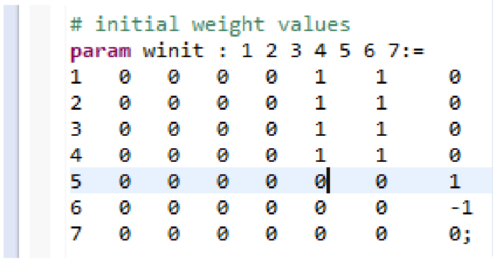
That is because it is like putting always the information that should pass between two nodes, to 0. Especially for $w_{5,6}, w_{5,7}$ we will always get as output of the neural network $F(x_1, x_2, x_3, x_4) = \frac{1}{1+e^{-(0)}} = 0.5$.

And as observed above, low bounding w with 0 always gives the function more chance to assign that value 0 to a w.

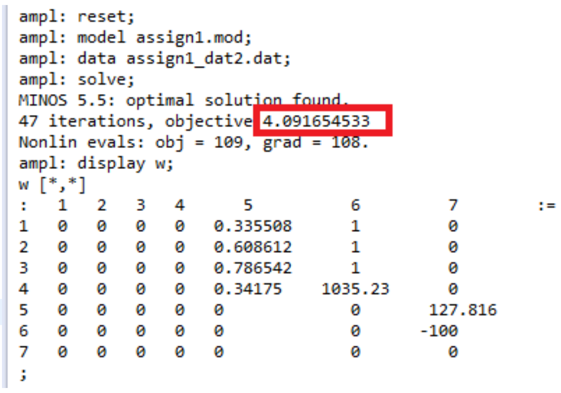
- Let us allow w go much more far negatively as the constraint was just done for not to allow it to go to minus infinite.

With

subject to c1 { i in {1..7},j in {1..7} } : w[i,j] >= -100; we get the following result:



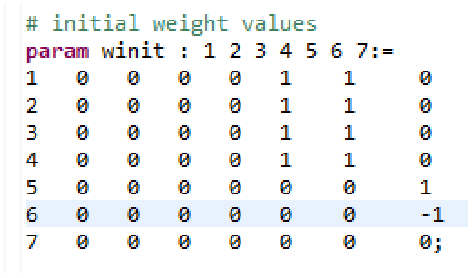
```
# initial weight values
param winit : 1 2 3 4 5 6 7:=
1    0    0    0    0    1    1    0
2    0    0    0    0    1    1    0
3    0    0    0    0    1    1    0
4    0    0    0    0    1    1    0
5    0    0    0    0    0    0    1
6    0    0    0    0    0    0   -1
7    0    0    0    0    0    0    0;
```



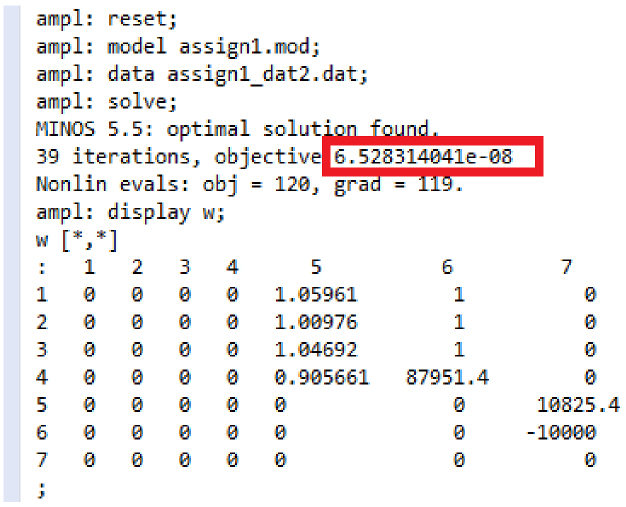
```
ampl: reset;
ampl: model assign1.mod;
ampl: data assign1_dat2.dat;
ampl: solve;
MINOS 5.5: optimal solution found.
47 iterations, objective 4.091654533
Nonlin evals: obj = 109, grad = 108.
ampl: display w;
w [*,*]
:   1   2   3   4      5          6          7        :=
1   0   0   0   0   0.335508      1          0
2   0   0   0   0   0.608612      1          0
3   0   0   0   0   0.786542      1          0
4   0   0   0   0   0.34175    1035.23       0
5   0   0   0   0   0             0        127.816
6   0   0   0   0   0             0       -100
7   0   0   0   0   0             0          0
;
```

If we go much more in minus, such the following,

subject to c1 { i in {1..7},j in {1..7} } : w[i,j] >= -10000; then we get the following results:



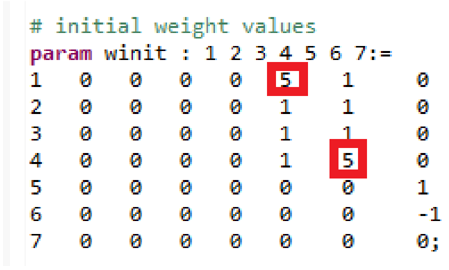
```
# initial weight values
param winit : 1 2 3 4 5 6 7:=
1    0    0    0    0    1    1    0
2    0    0    0    0    1    1    0
3    0    0    0    0    1    1    0
4    0    0    0    0    1    1    0
5    0    0    0    0    0    0    1
6    0    0    0    0    0    0   -1
7    0    0    0    0    0    0    0;
```



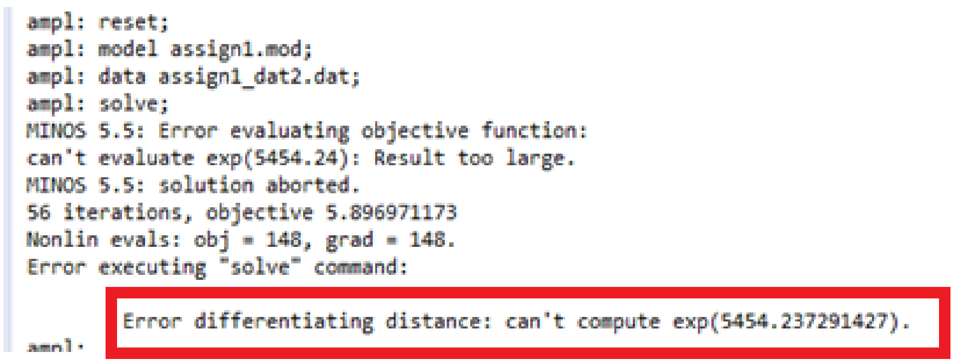
```
ampl: reset;
ampl: model assign1.mod;
ampl: data assign1_dat2.dat;
ampl: solve;
MINOS 5.5: optimal solution found.
39 iterations, objective 6.528314041e-08
Nonlin evals: obj = 120, grad = 119.
ampl: display w;
w [*,*]
:   1   2   3   4      5          6          7
1   0   0   0   0   1.05961       1          0
2   0   0   0   0   1.00976       1          0
3   0   0   0   0   1.04692       1          0
4   0   0   0   0   0.905661   87951.4       0
5   0   0   0   0   0             0      10825.4
6   0   0   0   0   0             0     -10000
7   0   0   0   0   0             0          0
;
```

We get an objective near to 0, it is ok but we should still perform some changes to the initial weights if we can get better results. Let us change the values of initial w.

- To do so, if we give a bit heavy weights (negatively or positively) to the ones going from layer 1 to layer 2, , it will always give incomputable value of exp as shown in figure ….

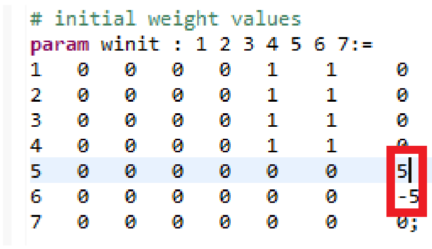


```
# initial weight values
param winit : 1 2 3 4 5 6 7:=
1    0    0    0    0    5    1    0
2    0    0    0    0    1    1    0
3    0    0    0    0    1    1    0
4    0    0    0    0    1    5    0
5    0    0    0    0    0    0    1
6    0    0    0    0    0    0   -1
7    0    0    0    0    0    0    0;
```

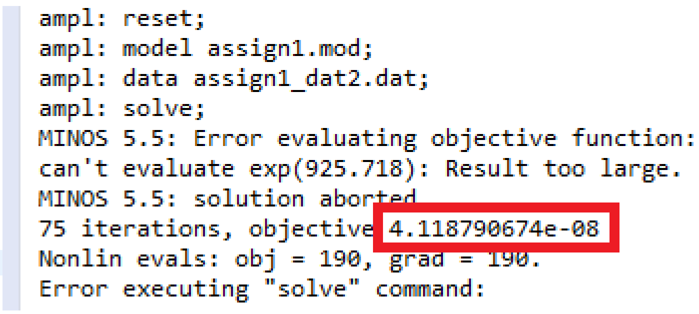


```
ampl: reset;
ampl: model assign1.mod;
ampl: data assign1_dat2.dat;
ampl: solve;
MINOS 5.5: Error evaluating objective function:
can't evaluate exp(5454.24): Result too large.
MINOS 5.5: solution aborted.
56 iterations, objective 5.896971173
Nonlin evals: obj = 148, grad = 148.
Error executing "solve" command:

    Error differentiating distance: can't compute exp(5454.237291427).
ampl:
```

- We switch on trying to change the weights between the last layer and the layer 2 of the network. During the execution, we the range of values computable is a bit large; doing that, we observe that we get better with the following :

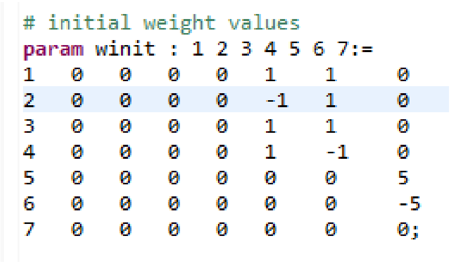


```
# initial weight values
param winit : 1 2 3 4 5 6 7:=
1   0   0   0   0   1    1    0
2   0   0   0   0   1    1    0
3   0   0   0   0   1    1    0
4   0   0   0   0   1    1    0
5   0   0   0   0   0    0    5
6   0   0   0   0   0    0    -5
7   0   0   0   0   0    0    0;
```

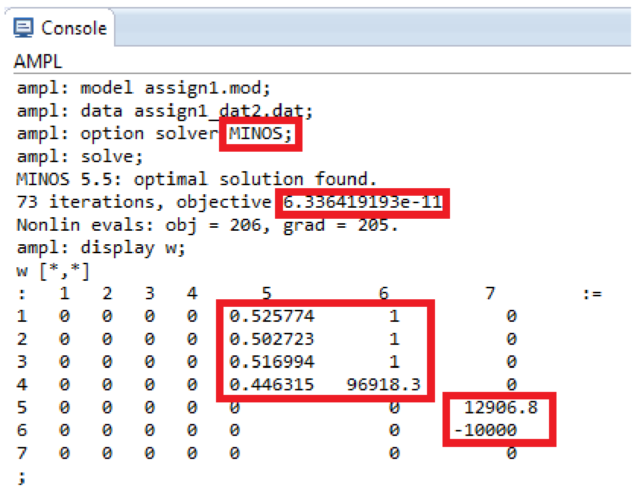


```
ampl: reset;
ampl: model assign1.mod;
ampl: data assign1_dat2.dat;
ampl: solve;
MINOS 5.5: Error evaluating objective function:
can't evaluate exp(925.718): Result too large.
MINOS 5.5: solution aborted.
75 iterations, objective 4.118790674e-08
Nonlin evals: obj = 190, grad = 190.
Error executing "solve" command:
```

- Then with that result, we come back to change some weights of the layer 1 to layer 2, just to recheck, and we get much more better result with:



```
# initial weight values
param winit : 1 2 3 4 5 6 7:=
1   0   0   0   0   1    1    0
2   0   0   0   0   -1   1    0
3   0   0   0   0   1    1    0
4   0   0   0   0   1    -1   0
5   0   0   0   0   0    0    5
6   0   0   0   0   0    0    -5
7   0   0   0   0   0    0    0;
```



```
ampl: model assign1.mod;
ampl: data assign1_dat2.dat;
ampl: option solver MINOS;
ampl: solve;
MINOS 5.5: optimal solution found.
73 iterations, objective 6.336419193e-11
Nonlin evals: obj = 206, grad = 205.
ampl: display w;
w [*,*]
:   1   2   3   4      5          6          7        :=
1   0   0   0   0   0.525774      1          0
2   0   0   0   0   0.502723      1          0
3   0   0   0   0   0.516994      1          0
4   0   0   0   0   0.446315   96918.3       0
5   0   0   0   0   0             0      12906.8
6   0   0   0   0   0             0     -10000
7   0   0   0   0   0             0          0
;
```

Finally we keep it as our last best result and go to tests.

In the conclusion, the lower bound of weight we set, the less cost we get. And the optimization results are related with the initial weights.

5. Conclusion and Comments

In this assignment, we formulate an optimization model to solve the neural network and use this method to do the pattern recognition. The testing results show that our method is acceptable.

About this assignment we also have more comments besides the Analysis Part.

Firstly, we wanted to try to solve the problem with other solvers like CONOPT, filter, Ipopt, KNITRO, LANCELOT, LOQO, LRAMBO, MOSEK, SNOPT, all of them are nonlinear solvers online as we just have the student version of AMPL (http://neos.mcs.anl.gov/neos/solvers/). But as problem, we can just upload the model, data and run files on the AMPL server from each respective web page; and normally should receive by mail the results. But still waiting for the results from the server, we did not get any of them. (Anyway, with the result, we can say that MINOS was enough for this problem.)

Secondly, we find it is prone to lose the parenthesis during coding the cost function in AMPL, we should be much more careful to deal with that function.