# INF3490 Oblig nr. 2

Emir Zamwa
UiO-username: emirz

8. October 2018

# Content

**1 Multilayer Perceptron** **1**
1.1 Instructions on running the program . . . . . 1
1.2 Answers to questions from assignment . . . . . 1
1.2.1 Running algorithm with x amount of hidden nodes . . . . . 1
1.3 Briefly what I have done . . . . . 2
1.3.1 Forward . . . . . 2
1.3.2 Train . . . . . 2
1.3.3 Early stopping . . . . . 2
1.3.4 Confusion . . . . . 2

# Pictures

1 X amount of hidden nodes: Results . . . . . 1

# 1 Multilayer Perceptron

## 1.1 Instructions on running the program

To run the program, make sure all python packages are installed on computer. This includes time, random, numpy, copy and pandas. * IF PANDAS CANNOT BE USED, READ COMMENT IN mlp.py ON LINE 204. * Once this is done, go into the code and change the iterations. This is in the train() function. It is initially set to 5. And all the answers is with 5 iterations. After this, to run, write "python3 movement.py" in the terminal. The test.py file was used to set in values as used in the lectures to check if i got correct values out. Was useful to find lot of mistakes and bugs. Tested with and without sigmoid. This file is not supposed to be used.

## 1.2 Answers to questions from assignment

### 1.2.1 Running algorithm with x amount of hidden nodes

```
Running...
Epoch[1] success rate ≈ 52.68%
Epoch[2] success rate ≈ 89.29%
Epoch[3] success rate ≈ 92.86%
Epoch[4] success rate ≈ 94.64%
Didn't find better [1 out of 1]
Time used: 5.183 seconds

CONFUSION MATRIX
Actual = Rows
Predicted = Columns
    a  b   c   d   e  f   g   h
a   6  0   0   0   1  1   0   0
b   0  9   0   0   0  0   0   0
c   0  0  13   0   0  0   0   0
d   0  0   0  12   0  0   0   1
e   0  0   0   1  13  0   0   0
f   0  0   0   0   0  8   1   0
g   0  0   0   2   0  4  16   0
h   0  0   0   2   0  0   0  21
```

(a) 6 hidden nodes

```
Running...
Epoch[1] success rate ≈ 44.64%
Epoch[2] success rate ≈ 77.68%
Epoch[3] success rate ≈ 88.39%
Epoch[4] success rate ≈ 91.07%
Epoch[5] success rate ≈ 94.64%
Epoch[6] success rate ≈ 95.54%
Epoch[7] success rate ≈ 96.43%
Didn't find better [1 out of 1]
Time used: 12.277 seconds

CONFUSION MATRIX
Actual = Rows
Predicted = Columns
    a   b   c   d  e  f   g   h
a  13   0   0   0  0  0   0   0
b   0  20   0   2  0  0   0   0
c   0   0  14   0  0  0   0   0
d   0   0   0  14  0  0   0   1
e   0   0   0   2  9  0   0   0
f   0   0   0   0  0  3   0   0
g   0   0   1   0  0  1  14   0
h   2   0   0   0  0  0   0  15
```

(b) 8 hidden nodes

```
Running...
Epoch[1] success rate ≈ 53.57%
Epoch[2] success rate ≈ 78.57%
Epoch[3] success rate ≈ 91.96%
Epoch[4] success rate ≈ 93.75%
Epoch[5] success rate ≈ 93.75%
Epoch[6] success rate ≈ 95.54%
Didn't find better [1 out of 1]
Time used: 14.484 seconds

CONFUSION MATRIX
Actual = Rows
Predicted = Columns
    a   b   c  d   e   f   g   h
a  11   0   0  0   0   0   0   0
b   0  13   0  1   0   0   0   0
c   0   0  12  1   0   0   0   0
d   0   0   0  9   0   0   0   0
e   0   0   0  4  14   1   0   0
f   0   0   0  0   0  11   4   0
g   0   0   0  0   0   0  14   0
h   0   0   0  1   0   0   0  15
```

(c) 12 hidden nodes

Figure 1: X amount of hidden nodes: Results

In the figures above are the results for running the algorithm with diffrent amounts of hidden nodes. I can't say how many is sufficient for the network. It seems very random, where sometimes one can be better than the other even if they have less hidden nodes. By only looking at the reported confusion matrices, I can't say which classes were most likely to be mistaken for each other.

## 1.3 Briefly what I have done

### 1.3.1 Forward

The forward() function calculates the activations of each node. For all the hidden nodes, as well as the output nodes. Here, the bias is also included. I have chosen to use sigmoid throughout my code, so once the nodes have their values, I make sure they go thorugh the sigmoid function. The forward function returns a copy of the inputs with the bias, and also a copy of the hidden nodes with the bias.

### 1.3.2 Train

The train function trains the system. Here I shuffle the inputs and targets for every input set of the assignment before I calculate on them. Then I go through each of the input sets and train them. I calculate deltas for outputs, then calculate the hidden weights. Then I calculate the deltas for the hidden nodes and in the end the weigthts of the inputs. Sigmoid is also used here. The formulas are taken from the book. The train method runs "iterations" amount of times, and for each iteration, it runs through all the input sets. So if 100 iterations is used, the program will take a while to complete.

### 1.3.3 Early stopping

The earlystopping function is made to stop the algorithm before it tries to overfit. This function runs a loop, that goes for as long as previous success rates are smaller than current success rates. Once previous is better, then it breaks and the program goes on to print out the confusion matrix. I have here also added the oppertunity to give the program more "chances" to get better. The variable stop can be changed to x ¿= 1 to give the loop extra runs to get better.

### 1.3.4 Confusion

Here the confusion matrix is made. I run forward on all of the inputs and then compare them to the global outputs. Then, I add 1 to the corresponding index of the matrix. After this, I use pandas to print out the matrix. Again, if pandas is not allowed, see line 204 in the code for instructions.