Øving 4 - Artificial Neural Networks

Sigve Skaugvoll

## A) Coding the **Learning rule for a perceptron.**

Following the algorithm in handed out article on page 172.

I chose to use the programming language Python 3.5.

The code can be located in file perceptronLearningRule.py along with the actual running of the artificial neural network, in the main function or in the file main.py

You can either run perceptronLearningRule.py or main.py (main.py) has a nicer “terminal print”.

## B) Running the **Learning rule for a perceptron, on specified parameters.**

The specified parameters are;

* Thetha = 0.2
* Weights = [0.3,-0.1]
* Learning rate = 0.1

The weights adjust / changes by either learning rate (𝛂) or 0, because,

the feature values (Xi(p)) are 1 or 0,

the error function e(p) produces

Yd - Yp

1 - 0 = 1,

1 - 1 = 0,

0 - 1 = - 1

or 0 - 0 = 0

Plotting this in the function for calculating new weight value:

**Weight algorithm is :** self.weights[i] = self.weights[i] + self.deltaRule(p, i)

**Delta rule is :** self.learning\_rate \* self.training\_set[p][index] \* self.calculateError(p)

Xi (e(p))

dr(𝛂,1) → 𝛂 \* 1 \* 0 = 0 # Yd = 0 and Yp = 0 → e(p) = 0

dr(𝛂,1) → 𝛂 \* 1 \* 1 = 𝛂 # Yd = 1 and Yp = 0 → e(p) = 1

dr(𝛂,1) → 𝛂 \* 1 \* 0 = 0 # Yd = 1 and Yp = 1 → e(p) = 0

dr(𝛂,1) → 𝛂 \* 1 \* -1 = -𝛂 # Yd = 0 and Yp = 1 → e(p) = -1

Thus weight changes accordingly to: weight[ p + 1] = weight[p] +/- 𝛂

#### Running on AND values:

##### Random initialization values:

/Library/Frameworks/Python.framework/Versions/3.5/bin/python3.5 "/Users/sigveskaugvoll/Documents/Skole/2017H/TDT4217 - KOGARK/øvinger/øving4/main.py"

I'm Perceptron

MIN weight are -0.5

MAX weight are 0.5

Theta are 0.3737228107137689

# of weights 2

Y is 0

Weights are [-0.09185459250548134, 0.33292114726442956]

Training.W0 Before : -0.09185459250548134

W0 After : -0.09185459250548134

W1 Before : 0.33292114726442956

W1 After : 0.33292114726442956

.W0 Before : -0.09185459250548134

W0 After : -0.09185459250548134

W1 Before : 0.33292114726442956

W1 After : 0.33292114726442956

.W0 Before : -0.09185459250548134

W0 After : -0.09185459250548134

W1 Before : 0.33292114726442956

W1 After : 0.33292114726442956

.W0 Before : -0.09185459250548134

W0 After : 0.00814540749451867

W1 Before : 0.33292114726442956

W1 After : 0.43292114726442954

.W0 Before : 0.00814540749451867

W0 After : 0.00814540749451867

W1 Before : 0.43292114726442954

W1 After : 0.43292114726442954

.W0 Before : 0.00814540749451867

W0 After : 0.00814540749451867

W1 Before : 0.43292114726442954

W1 After : 0.33292114726442956

.W0 Before : 0.00814540749451867

W0 After : 0.00814540749451867

W1 Before : 0.33292114726442956

W1 After : 0.33292114726442956

.W0 Before : 0.00814540749451867

W0 After : 0.10814540749451867

W1 Before : 0.33292114726442956

W1 After : 0.43292114726442954

.W0 Before : 0.10814540749451867

W0 After : 0.10814540749451867

W1 Before : 0.43292114726442954

W1 After : 0.43292114726442954

.W0 Before : 0.10814540749451867

W0 After : 0.10814540749451867

W1 Before : 0.43292114726442954

W1 After : 0.33292114726442956

.W0 Before : 0.10814540749451867

W0 After : 0.10814540749451867

W1 Before : 0.33292114726442956

W1 After : 0.33292114726442956

.W0 Before : 0.10814540749451867

W0 After : 0.10814540749451867

W1 Before : 0.33292114726442956

W1 After : 0.33292114726442956

.W0 Before : 0.10814540749451867

W0 After : 0.10814540749451867

W1 Before : 0.33292114726442956

W1 After : 0.33292114726442956

.W0 Before : 0.10814540749451867

W0 After : 0.10814540749451867

W1 Before : 0.33292114726442956

W1 After : 0.33292114726442956

.W0 Before : 0.10814540749451867

W0 After : 0.10814540749451867

W1 Before : 0.33292114726442956

W1 After : 0.33292114726442956

.W0 Before : 0.10814540749451867

W0 After : 0.10814540749451867

W1 Before : 0.33292114726442956

W1 After : 0.33292114726442956

Training complete in 1.64seconds and 3 epochs!

Now testing

[0, 0] should be 0, and are : 0

[0, 1] should be 0, and are : 0

[1, 0] should be 0, and are : 0

[1, 1] should be 1, and are : 1

Success rate: 1.0%

I'm Perceptron

MIN weight are -0.5

MAX weight are 0.5

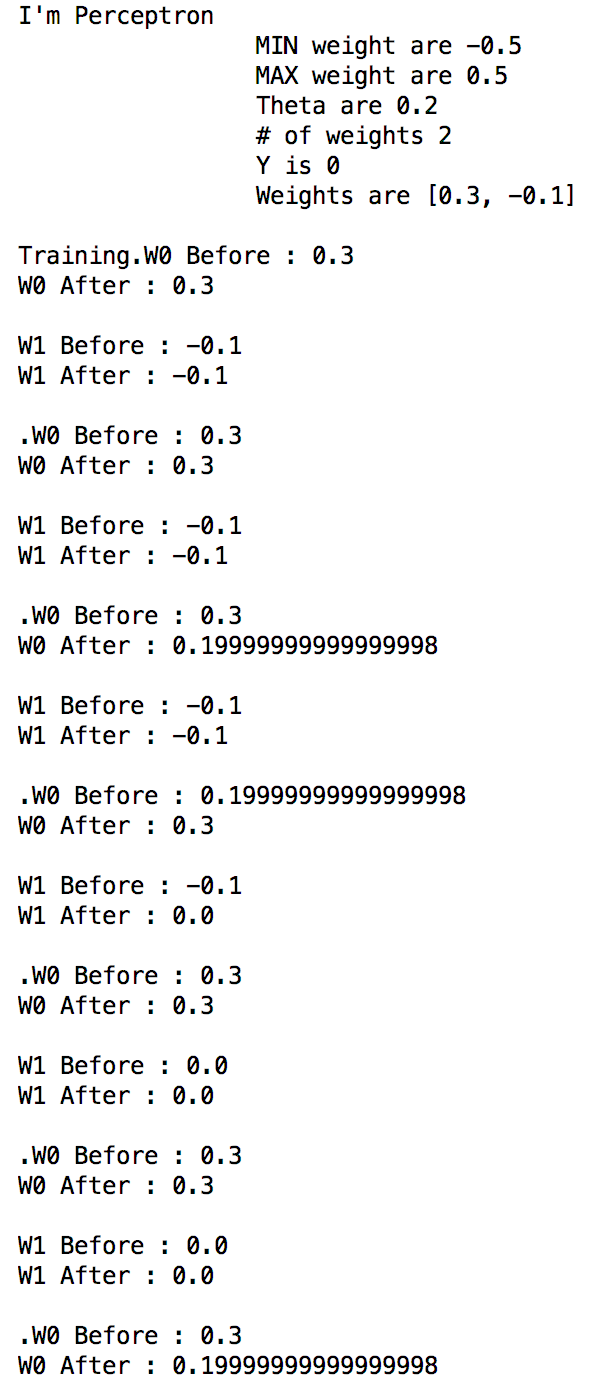
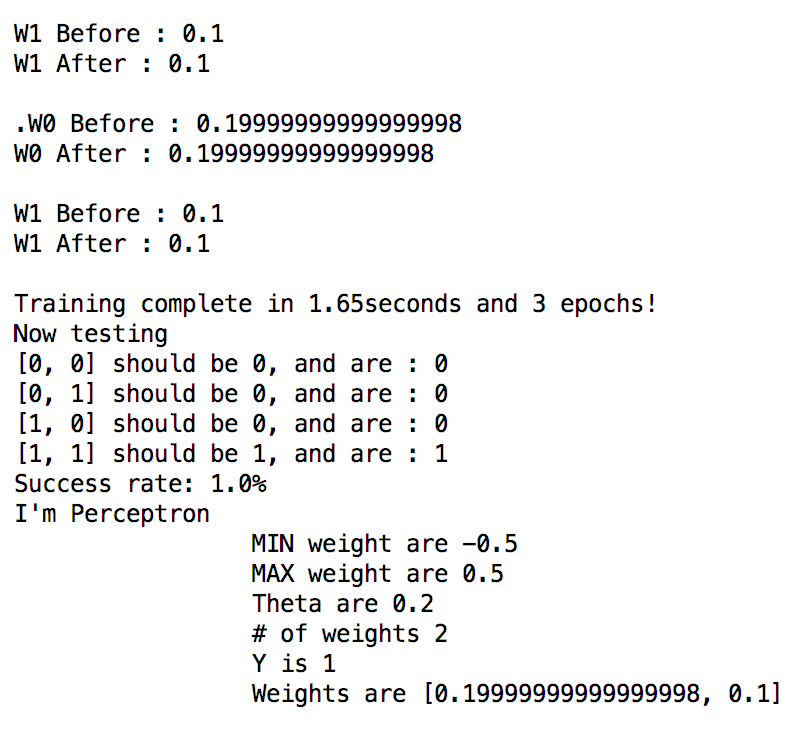
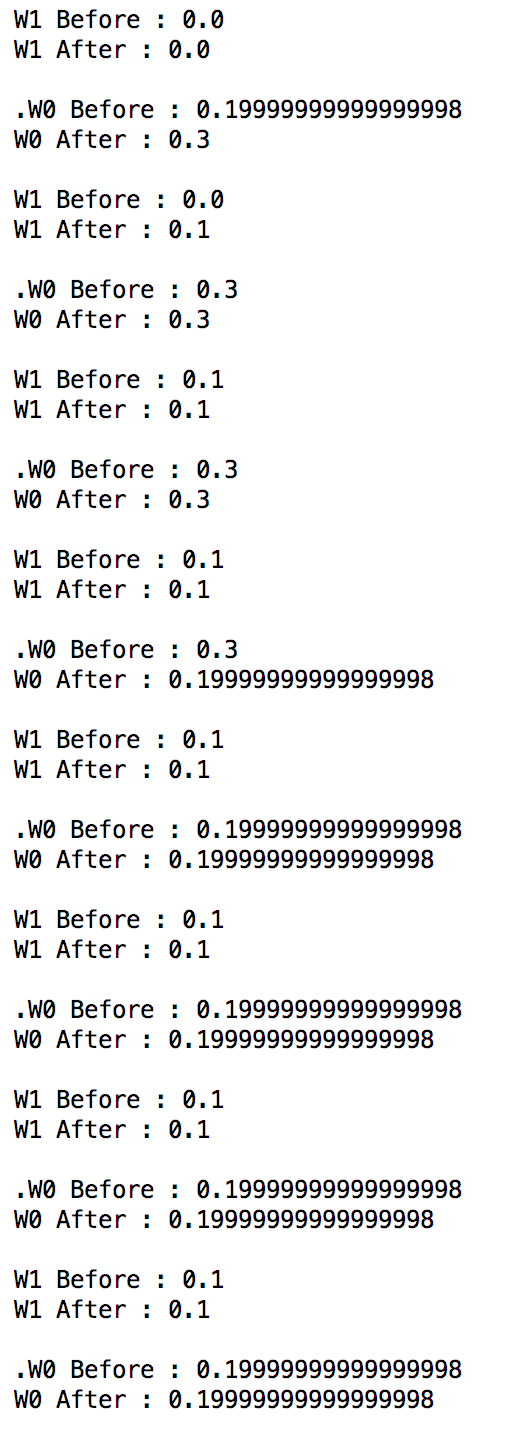
Theta are 0.3737228107137689

# of weights 2

Y is 1

Weights are [0.10814540749451867, 0.33292114726442956]

##### Example initialization values:



#### Running on OR values:

##### Example initialization values:

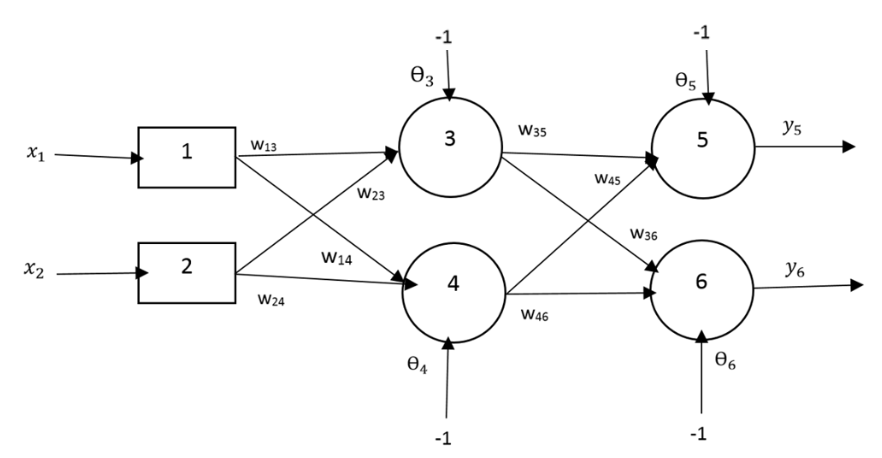
#### Skjermbilde 2017-10-11 kl. 18.59.44.pngSkjermbilde 2017-10-11 kl. 18.59.56.pngSkjermbilde 2017-10-11 kl. 18.59.31.png

#### Changing initial values for wi and theta:

Make the world in difference, if the initial weights are way of (wrong), and there is a small learning rate, then the time to learn and get good enough weights are tremendous! This because As explained earlier, the weight changes between +/- learning rate for each iteration (not epoch). Thus if the distance between the correct weights and the initialized weights are huge and the learning rate is not good eonugh, it will take a lot of time to “walk the distance with a distance for each step equal to the learning rate for each iteration”.

## C) Execute and calculate the backpropagation (learning) of a ANN.

* What are the values for each weight and thresholds (theta) after one iteration?



𝑥1= 0, 𝑥2 = 1 || y𝑑,5 = 0, y𝑑,6 = 1

initialNetwork.jpg

𝛼 = 0.1,

𝑤13 = 0.5,

𝑤14 = 0.0,

𝑤23 = 0.0,

𝑤24 = 0.9,

𝑤35 =0.4,

𝑤36 = 1.0,

𝑤45 = −1.2,

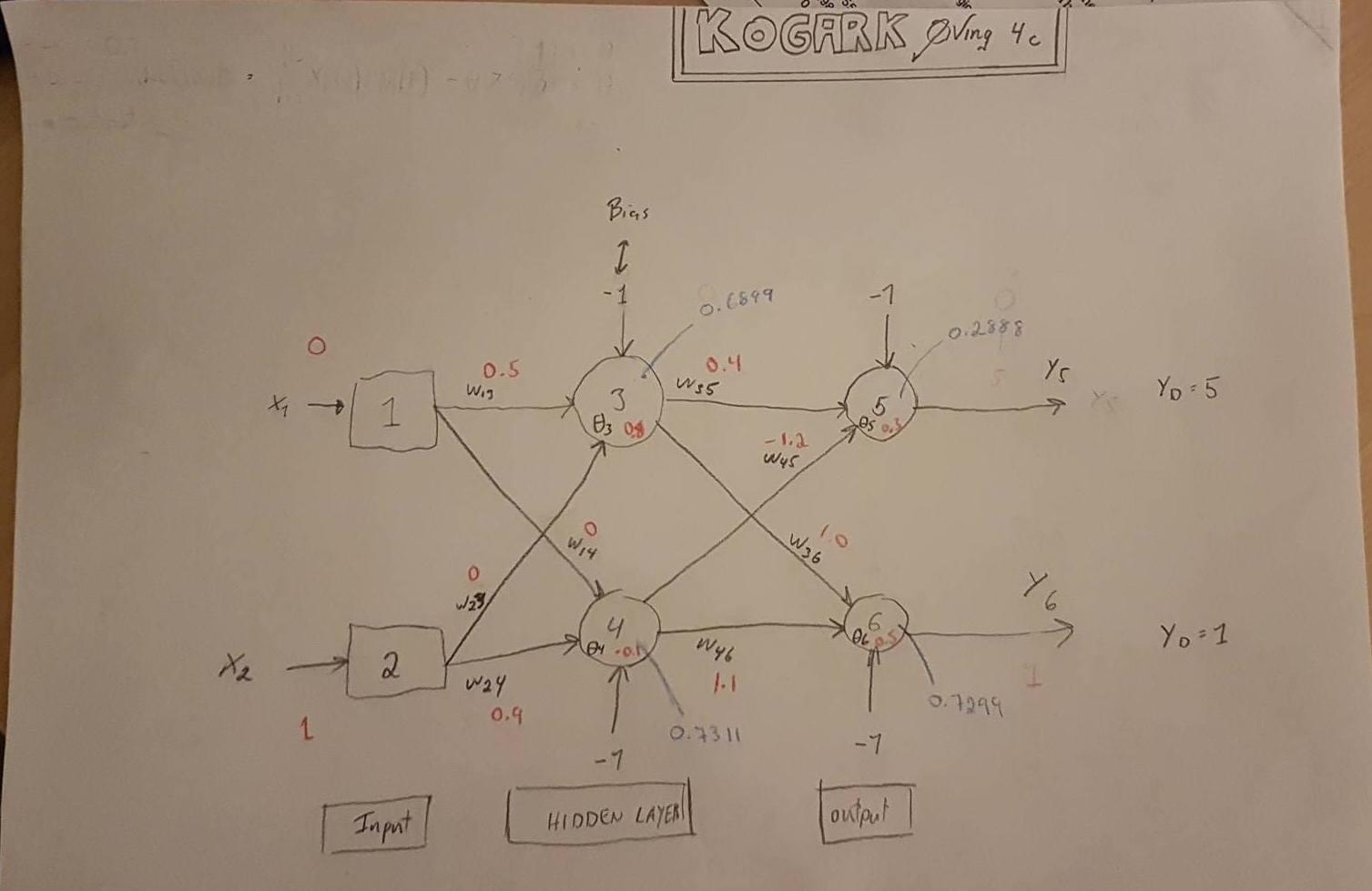
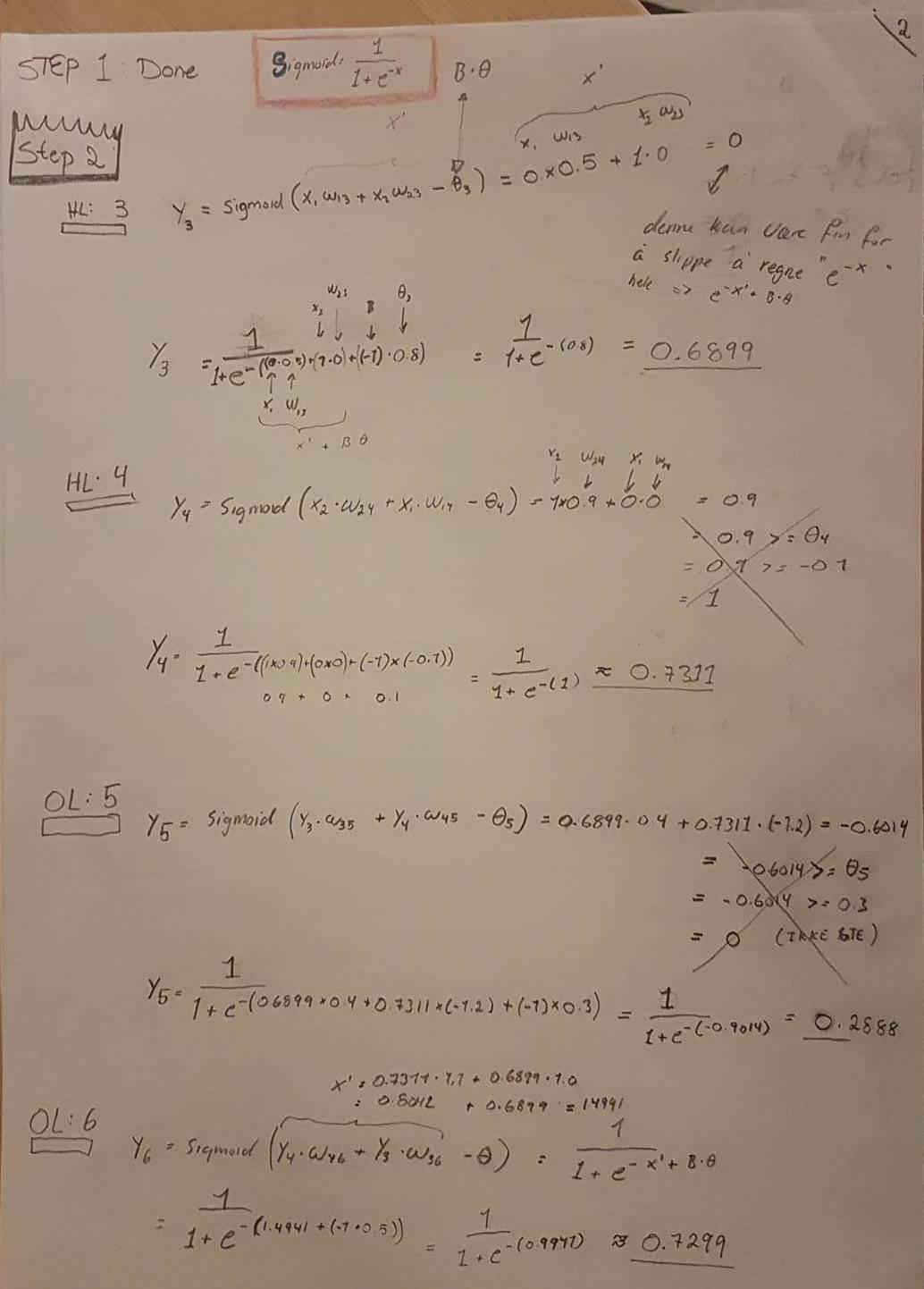
𝑤46 = 1.1,

Ө3 = 0.8,

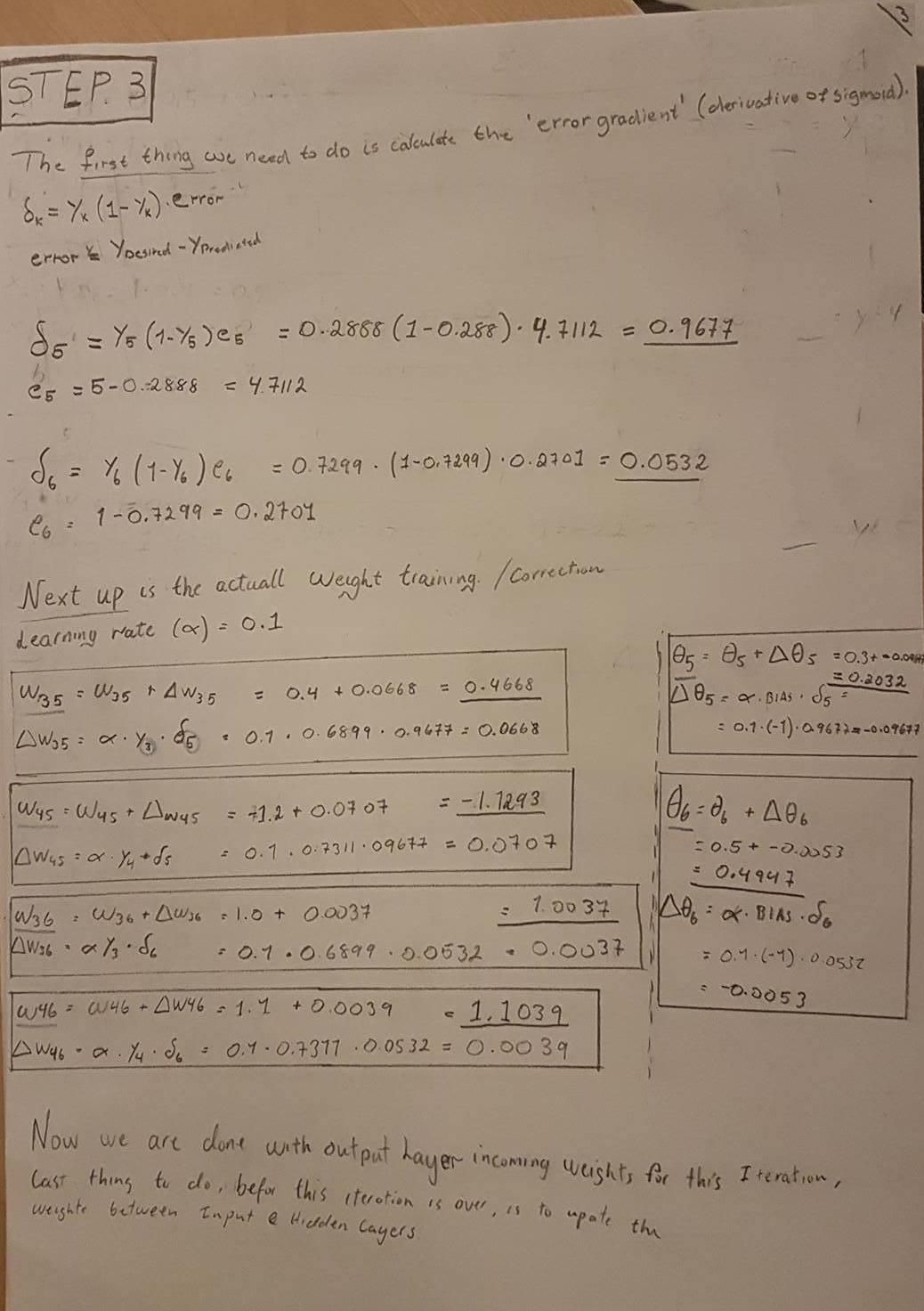
Ө4 = −0.1,

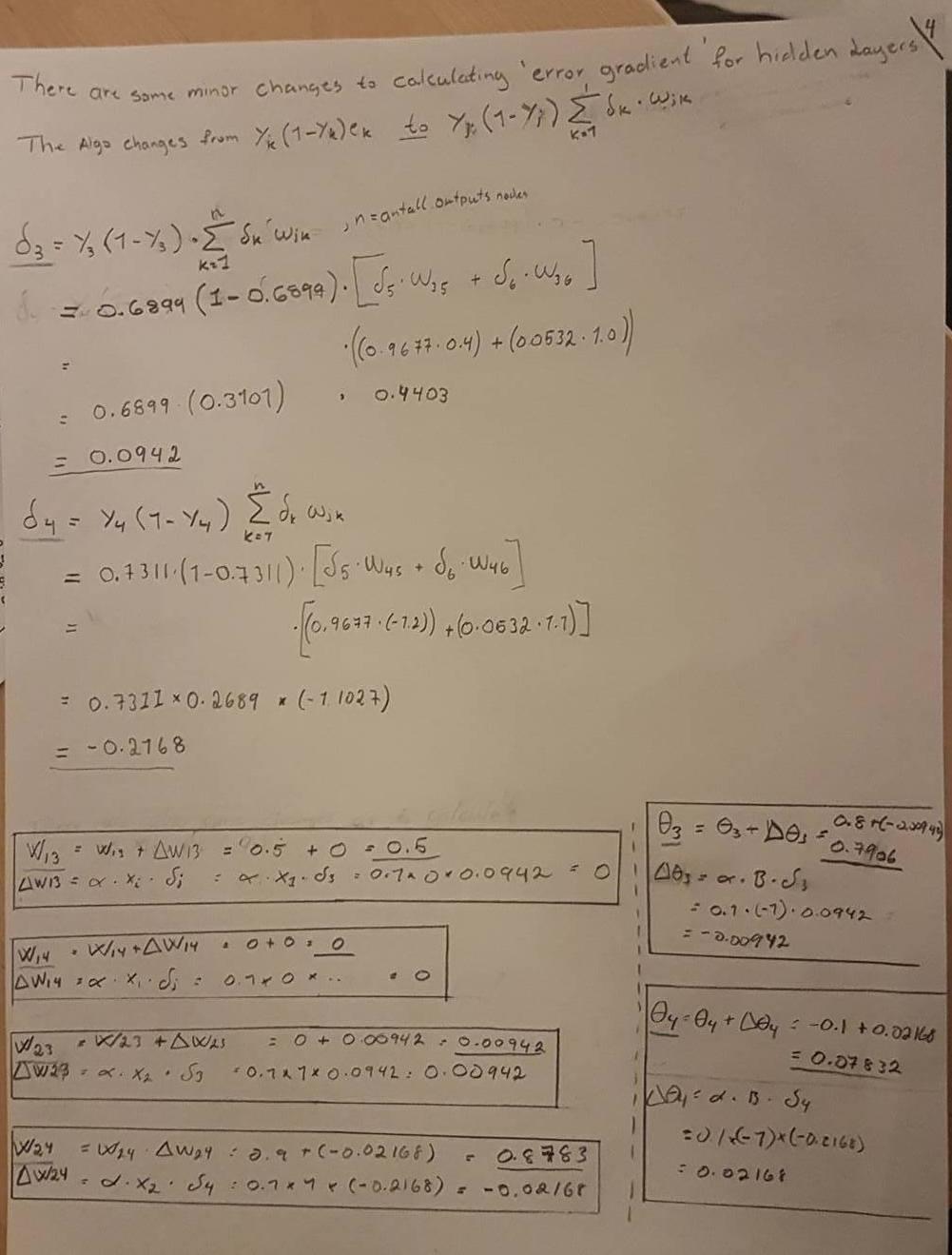
Ө5 = 0.3,

Ө6 = 0.5



The blue numbers are the numer we just calculated, the output of the nodes.

****

****

## D) Program a `Auto encoder` `Feed Forward Network`.

* You are free to use any library and language you like, but we suggest Python and PyBrain. If trouble during installation with pip, use Anaconda instead.

The code can be found in the file autoencoder\_feed\_forward.py

1. What is the minimum amount of neurons in the hidden layer still gives a reasonably good result? → input = output
   1. When using 8 neurons in hidden layer the net got all numbers 1 -> 8 correct.
   2. When reducing amount of neurons in hidden layer by 50% (down to 4), the net still got all numbers 1 -> 8 correct.
   3. When reducing again, now down to 3 the correctness stays pretty high but the networks certainty of each output is high, its certainty / error is only ±0.4 on the correct predictions. if rounding output to Integer, the following is number of corrects on 5 runs.
      1. Run 1: 7 / 8
      2. Run 2: 8 / 8
      3. Run 3: 8 / 8
      4. Run 4: 7 / 8
      5. Run 5: 7 / 8
      6. (7 + 8 + 8 + 7 + 7) / 5 = 7,4
   4. When reducing again, now down to 2 the correctness stays pretty high but the networks certainty of each output is has reduced a lot from high to vary about ±0.40 on the correct ones, if rounding output to Integer, the following is number of corrects on 5 runs.
      1. Run 1: 5 / 8
      2. Run 2: 5 / 8
      3. Run 3: 7 / 8
      4. Run 4: 7 / 8
      5. Run 5: 8 / 8
      6. (5 + 5 + 7 + 7 + 8) / 5 = 6,4
2. What is it, that the neural net has recreated through the hidden layer to be able to produce a good result, when running on the lowest amount of neurons in hidden layer for good result.
3. How does the result react to numbers it has not seen before, such as negative numbers, numbers greater than 8, etc. [Using 8 hiddenNeurons]
4. 9 → 8.46 ± 0.44
5. 0 → 0.60 ± 0.60
6. -2.5 → -1.82 ± 0.68
7. 9.7 → 8.76 ± 0.94
8. Infinity → 9.426 ± ….