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 ($X_i(p)$) 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:

```
I'm Perceptron
                MIN weight are -0.5
                MAX weight are 0.5
                Theta are 0.2
                # of weights 2
                Y is 0
                Weights are [0.3, -0.1]

Training.W0 Before : 0.3
W0 After : 0.3

W1 Before : -0.1
W1 After : -0.1

.W0 Before : 0.3
W0 After : 0.3

W1 Before : -0.1
W1 After : -0.1

.W0 Before : 0.3
W0 After : 0.19999999999999998

W1 Before : -0.1
W1 After : -0.1

.W0 Before : 0.19999999999999998
W0 After : 0.3

W1 Before : -0.1
W1 After : 0.0

.W0 Before : 0.3
W0 After : 0.3

W1 Before : 0.0
W1 After : 0.0

.W0 Before : 0.3
W0 After : 0.3

W1 Before : 0.0
W1 After : 0.0

.W0 Before : 0.3
W0 After : 0.19999999999999998

W1 Before : 0.0
W1 After : 0.0

.W0 Before : 0.19999999999999998
W0 After : 0.3

W1 Before : 0.0
W1 After : 0.1

.W0 Before : 0.3
W0 After : 0.3

W1 Before : 0.1
W1 After : 0.1

.W0 Before : 0.3
W0 After : 0.3

W1 Before : 0.1
W1 After : 0.1

.W0 Before : 0.3
W0 After : 0.19999999999999998

W1 Before : 0.1
W1 After : 0.1

.W0 Before : 0.19999999999999998
W0 After : 0.19999999999999998

W1 Before : 0.1
W1 After : 0.1

.W0 Before : 0.19999999999999998
W0 After : 0.19999999999999998

W1 Before : 0.1
W1 After : 0.1

.W0 Before : 0.19999999999999998
W0 After : 0.19999999999999998

W1 Before : 0.1
W1 After : 0.1

.W0 Before : 0.19999999999999998
W0 After : 0.19999999999999998

W1 Before : 0.1
W1 After : 0.1

Training complete in 1.65seconds 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 0.2
                # of weights 2
                Y is 1
                Weights are [0.19999999999999998, 0.1]
```

Running on OR values:

Example initialization values:

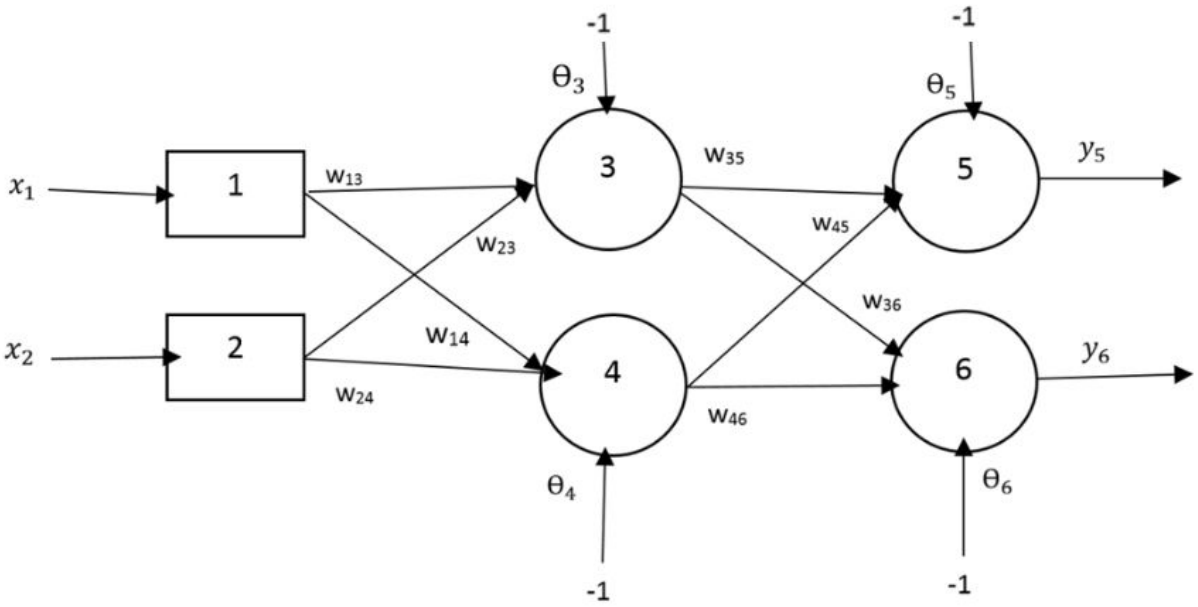
```
I'm Perceptron
                MIN weight are -0.5
                MAX weight are 0.5
                Theta are 0.2
                # of weights 2
                Y is 0
                Weights are [0.3, -0.1]

Training.W0 Before : 0.3
W0 After : 0.3

W1 Before : -0.1
W1 After : -0.1

.W0 Before : 0.3
W0 After : 0.3

W1 Before : -0.1
W1 After : -0.1

.W0 Before : 0.3
W0 After : 0.19999999999999998

W1 Before : -0.1
W1 After : -0.1

.W0 Before : 0.19999999999999998
W0 After : 0.3

W1 Before : -0.1
W1 After : 0.0

.W0 Before : 0.3
W0 After : 0.3

W1 Before : 0.0
W1 After : 0.0

.W0 Before : 0.3
W0 After : 0.3

W1 Before : 0.0
W1 After : 0.0

.W0 Before : 0.3
W0 After : 0.19999999999999998

W1 Before : 0.0
W1 After : 0.0

.W0 Before : 0.19999999999999998
W0 After : 0.3

W1 Before : 0.0
W1 After : 0.1

.W0 Before : 0.3
W0 After : 0.3

W1 Before : 0.1
W1 After : 0.1

.W0 Before : 0.3
W0 After : 0.3

W1 Before : 0.1
W1 After : 0.1

.W0 Before : 0.3
W0 After : 0.19999999999999998

W1 Before : 0.1
W1 After : 0.1

.W0 Before : 0.19999999999999998
W0 After : 0.19999999999999998

W1 Before : 0.1
W1 After : 0.1

.W0 Before : 0.19999999999999998
W0 After : 0.19999999999999998

W1 Before : 0.1
W1 After : 0.1

.W0 Before : 0.19999999999999998
W0 After : 0.19999999999999998

W1 Before : 0.1
W1 After : 0.1

.W0 Before : 0.19999999999999998
W0 After : 0.19999999999999998

W1 Before : 0.1
W1 After : 0.1

Training complete in 1.65seconds 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 0.2
                # of weights 2
                Y is 1
                Weights are [0.19999999999999998, 0.1]
```

Changing initial values for w_i 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?

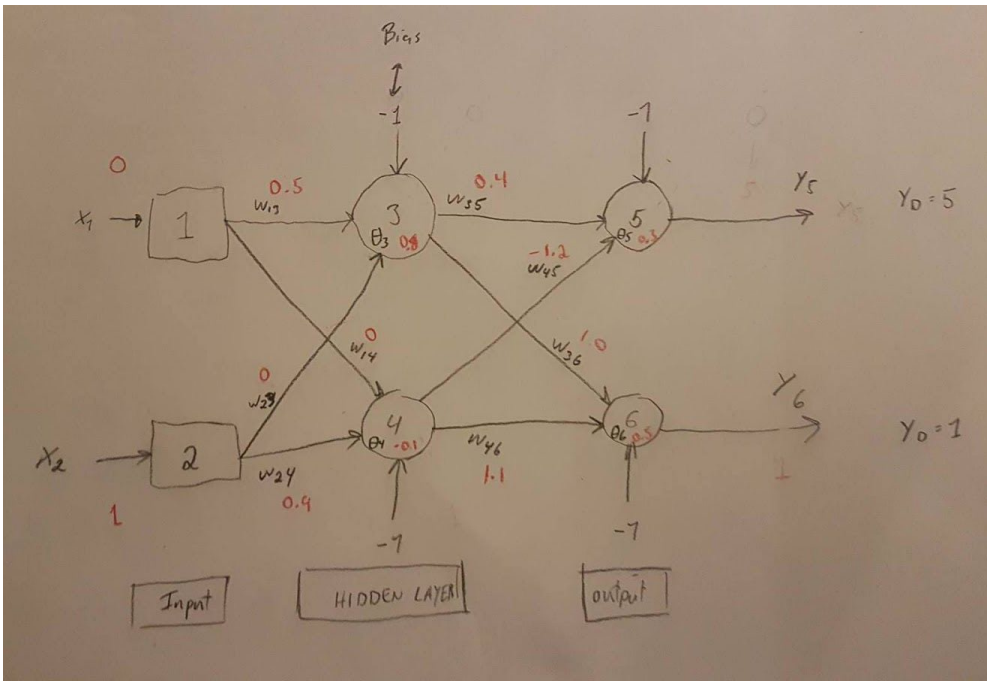


$x1$= 0, $x2$ = 1 || yd,5 = 0, yd,6 = 1

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,

Θ3 = 0.8,
Θ4 = −0.1,
Θ5 = 0.3,
Θ6 = 0.5



STEP 1: Done

Sigmoid: $\frac{1}{1+e^{-x}}$ $\quad B \cdot \theta$ $\quad x'$

Step 2

HL: 3 $\quad Y_3 = \text{Sigmoid}(x_1 w_{13} + x_2 w_{23} - \theta_3) = 0 \times 0.5 + 1 \cdot 0 = 0$

denne kan vare fin for å slippe å regne "e^{-x}". hele $\Rightarrow e^{-x'} + B \cdot \theta$

$Y_3 = \frac{1}{1+e^{-((0 \cdot 0.5) + (1 \cdot 0) + (-1) \cdot 0.8)}} = \frac{1}{1+e^{-(0.8)}} = \underline{0.6899}$

(x_1, w_{13}: $x' + B\theta$; x_2, w_{23}, B, θ_3)

HL: 4 $\quad Y_4 = \text{Sigmoid}(x_2 \cdot w_{24} + x_1 \cdot w_{14} - \theta_4) = 1 \times 0.9 + 0 \cdot 0 = 0.9$

~~$= 0.9 >= \theta_4$~~

~~$= 0.1 >= -0.1$~~

~~$= 1$~~

$Y_4 = \frac{1}{1+e^{-((1 \times 0.9) + (0 \times 0) + (-1) \times (-0.1))}} = \frac{1}{1+e^{-(1)}} \approx \underline{0.7311}$

$0.9 + 0 + 0.1$

OL: 5 $\quad Y_5 = \text{Sigmoid}(Y_3 \cdot w_{35} + Y_4 \cdot w_{45} - \theta_5) = 0.6899 \cdot 0.4 + 0.7311 \cdot (-1.2) = -0.6014$

~~$= -0.6014 >= \theta_5$~~

~~$= -0.6014 >= 0.3$~~

~~$= 0$ (TAKE GTE)~~

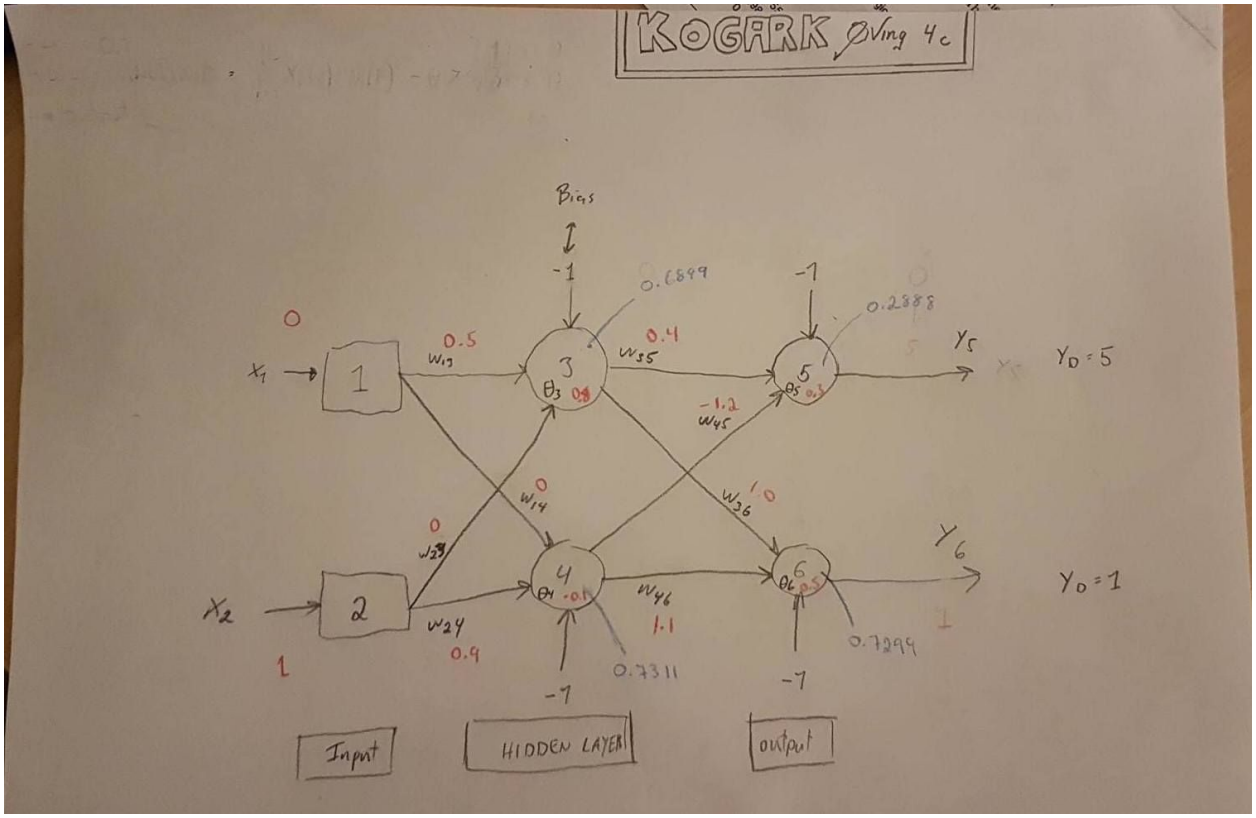
$Y_5 = \frac{1}{1+e^{-(0.6899 \times 0.4 + 0.7311 \times (-1.2) + (-1) \times 0.3)}} = \frac{1}{1+e^{-(-0.9014)}} = \underline{0.2888}$

$x' = 0.7311 \cdot 1.1 + 0.6899 \cdot 1.0$

$= 0.8042 + 0.6899 = 1.4941$

OL: 6 $\quad Y_6 = \text{Sigmoid}(Y_4 \cdot w_{46} + Y_3 \cdot w_{36} - \theta) = \frac{1}{1+e^{-x'} + B \cdot \theta}$

$= \frac{1}{1+e^{-(1.4941 + (-1 \cdot 0.5))}} = \frac{1}{1+e^{-(0.9941)}} \approx \underline{0.7299}$



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

STEP 3

The first thing we need to do is calculate the 'error gradient' (derivative of sigmoid).

$\delta_k = Y_k(1-Y_k) \cdot error$

$error = Y_{desired} - Y_{predicted}$

$\delta_5 = Y_5(1-Y_5)e_5 = 0.2888(1-0.288) \cdot 4.7112 = \underline{0.9677}$

$e_5 = 5 - 0.2888 = 4.7112$

$\delta_6 = Y_6(1-Y_6)e_6 = 0.7299 \cdot (1-0.7299) \cdot 0.2701 = \underline{0.0532}$

$e_6 = 1 - 0.7299 = 0.2701$

Next up is the actuall weight training /correction

learning rate (α) = 0.1

$w_{35} = w_{35} + \Delta w_{35} = 0.4 + 0.0668 = \underline{0.4668}$

$\Delta w_{35} = \alpha \cdot Y_3 \cdot \delta_5 = 0.1 \cdot 0.6899 \cdot 0.9677 = 0.0668$

$\theta_5 = \theta_5 + \Delta\theta_5 = 0.3 + -0.0968 = \underline{0.2032}$

$\Delta\theta_5 = \alpha \cdot BIAS \cdot \delta_5 = 0.1 \cdot (-1) \cdot 0.9677 = -0.09677$

$w_{45} = w_{45} + \Delta w_{45} = 71.2 + 0.0707 = \underline{-1.7293}$

$\Delta w_{45} = \alpha \cdot Y_4 \cdot \delta_5 = 0.1 \cdot 0.7311 \cdot 0.9677 = 0.0707$

$\theta_6 = \theta_6 + \Delta\theta_6 = 0.5 + -0.0053 = \underline{0.4947}$

$\Delta\theta_6 = \alpha \cdot BIAS \cdot \delta_6 = 0.1 \cdot (-1) \cdot 0.0532 = -0.0053$

$w_{36} = w_{36} + \Delta w_{36} = 1.0 + 0.0037 = \underline{1.0037}$

$\Delta w_{36} = \alpha Y_3 \cdot \delta_6 = 0.1 \cdot 0.6899 \cdot 0.0532 = 0.0037$

$w_{46} = w_{46} + \Delta w_{46} = 1.1 + 0.0039 = \underline{1.1039}$

$\Delta w_{46} = \alpha \cdot Y_4 \cdot \delta_6 = 0.1 \cdot 0.7377 \cdot 0.0532 = 0.0039$

Now we are done with output layer incoming weights for this Iteration, last thing to do, befor this iteration is over, is to upate the weights between Input & Hidden Layers

There are some minor changes to calculating 'error gradient' for hidden layers

The Algo changes from $Y_k(1-Y_k)e_k$ to $Y_j(1-Y_j)\sum_{k=1}^{n} \delta_k \cdot w_{jk}$

$\delta_3 = Y_3(1-Y_3)\cdot\sum_{k=1}^{n} \delta_k w_{ik}$, n = antall outputs noder

$= 0.6899\,(1-0.6899)\cdot\left[\delta_5 \cdot w_{35} + \delta_6 \cdot w_{36}\right]$

$\cdot\left((0.9677 \cdot 0.4) + (0.0532 \cdot 1.0)\right)$

$= 0.6899\,(0.3107)$, 0.4403

$= 0.0942$

$\delta_4 = Y_4(1-Y_4)\sum_{k=1}^{n} \delta_k w_{jk}$

$= 0.7311\cdot(1-0.7311)\cdot\left[\delta_5 \cdot w_{45} + \delta_6 \cdot w_{46}\right]$

$\cdot\left[(0.9677\cdot(-7.2)) + (0.0532 \cdot 7.7)\right]$

$= 0.7311 \times 0.2689 \times (-7.1027)$

$= -0.2168$

$W_{13} = W_{13} + \Delta W_{13} = 0.5 + 0 = 0.5$

$\Delta W_{13} = \alpha \cdot x_i \cdot \delta_j = \alpha \cdot x_1 \cdot \delta_3 = 0.7 \times 0 \times 0.0942 = 0$

$W_{14} = W_{14} + \Delta W_{14} = 0 + 0 = 0$

$\Delta W_{14} = \alpha \cdot x_1 \cdot \delta_j = 0.7 \times 0 \times \ldots = 0$

$W_{23} = W_{23} + \Delta W_{23} = 0 + 0.00942 = 0.00942$

$\Delta W_{23} = \alpha \cdot x_2 \cdot \delta_3 = 0.7 \times 7 \times 0.0942 = 0.00942$

$W_{24} = W_{24} \cdot \Delta W_{24} = 0.9 + (-0.02168) = 0.8783$

$\Delta W_{24} = \alpha \cdot x_2 \cdot \delta_4 = 0.7 \times 7 \times (-0.2168) = -0.02168$

$\theta_3 = \theta_3 + \Delta\theta_3 = 0.8 + (-0.00942) = 0.7906$

$\Delta\theta_3 = \alpha \cdot \beta \cdot \delta_3$

$= 0.7 \cdot (-7) \cdot 0.0942$

$= -0.00942$

$\theta_4 = \theta_4 + \Delta\theta_4 = -0.1 + 0.02168 = 0.07832$

$\Delta\theta_4 = \alpha \cdot \beta \cdot \delta_4$

$= 0.7 \cdot (-7) \times (-0.2168)$

$= 0.02168$

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
    a) When using 8 neurons in hidden layer the net got all numbers 1 -> 8 correct.
    b) When reducing amount of neurons in hidden layer by 50% (down to 4), the net still got all numbers 1 -> 8 correct.
    c) 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.
        i) Run 1: 7 / 8
        ii) Run 2: 8 / 8
        iii) Run 3: 8 / 8
        iv) Run 4: 7 / 8
        v) Run 5: 7 / 8
        vi) (7 + 8 + 8 + 7 + 7) / 5 = 7,4
    d) 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.
        i) Run 1: 5 / 8
        ii) Run 2: 5 / 8
        iii) Run 3: 7 / 8
        iv) Run 4: 7 / 8
        v) Run 5: 8 / 8
        vi) (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]
    1. 9 → 8.46 ± 0.44
    2. 0 → 0.60 ± 0.60
    3. -2.5 → -1.82 ± 0.68
    4. 9.7 → 8.76 ± 0.94

5. Infinity → 9.426 ± ….