MULTI PERCEPTRON – HEART DISEASE

HANIF IZZUDIN RAHMAN

S2 TEKNIK ELEKTRO

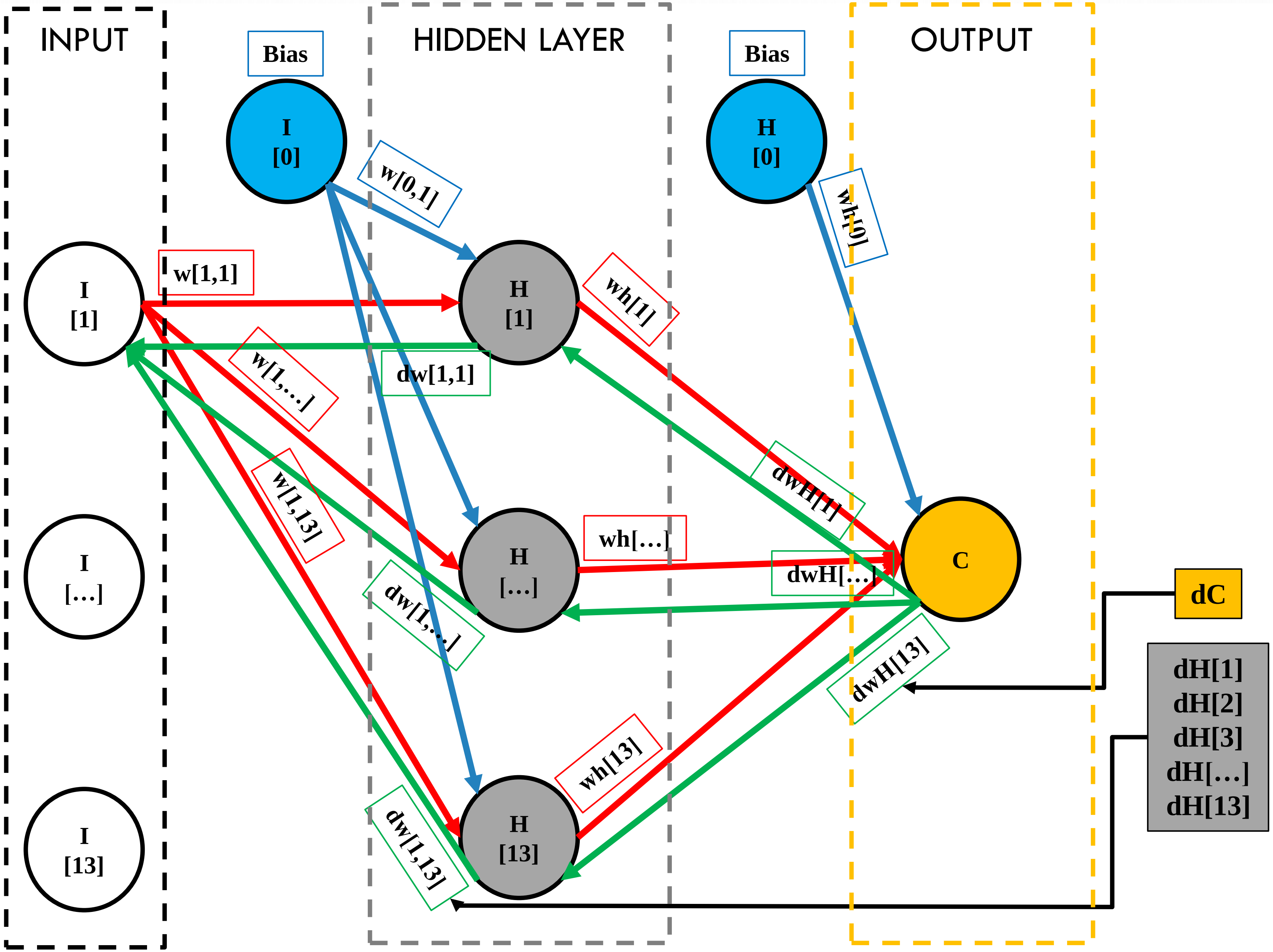
PENS 2020

NEURAL NETWORK - STRUCTURE

Epoch = 10.000
Miu (η) = 0.1

Data = 270
- Absence = 150
- Presence = 120

Input = 13
Hidden Layer = 1
Hidden Input = 13
Output = 1



1.1 IMPORT LIBRARY & DATASET

```
# Final STEP V2.1 (Output Heart Disease = 1, No Heart Disease = 0 **CONVERT**)

import random
import math                # rumus sigmoid
import pandas as pd

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> IMPORT DATASET
dataset = pd.read_csv('C:\Hanif Izzudin Rahman D4 EB 2016\S2 Elektro 2020 -
PENS\Semester 1\Artificial Intelligent -
Aliridho\Tugas\AI_M10_MultiPerceptron_Heart\heartV2.csv')
dataset
```

1.2 DECLARE VARIABLE

```
print("==========================================================> Declare Variable")

g, h = 13+1, 13+1;
w = [[0 for x in range(g)] for y in range(h)]
dw = [[0 for x in range(g)] for y in range(h)]

wh = {}
summation = {}
H = {}
Cout = {}
CoutTh = {}
dH = {}
dwH = {}

Bias = 1
epoch = 10000+1
miu = 0.1

MSE = 0
TrueC = 0
FalseC = 0
```

2.1 GET INPUT DATA

```
# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> Input Data
print("==========================================================> Input Data")

Input = dataset.iloc[:,0:13]
print(Input)
print("-"*100)

g, h = 13+1, 270+1;
I = [[0 for x in range(g)] for y in range(h)]

for j in range (270):
    for k in range (13):
        I[j+1][k+1] = Input.iloc[j,k]
        # =================================================================== Convert
Input
        if(I[j+1][k+1] >= 100):
            I[j+1][k+1] = I[j+1][k+1]/100
        elif(I[j+1][k+1] >= 10):
            I[j+1][k+1] = I[j+1][k+1]/10
    print("%.2f "*13 %(I[j][1], I[j][2], I[j][3], I[j][4], I[j][5], I[j][6], I[j][7],
I[j][8], I[j][9], I[j][10], I[j][11], I[j][12], I[j][13]))
```

2.2 GET OUTPUT DATA

```
# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> Output Data
print("=========================================================> Output Data")

Output = dataset.iloc[:,-1]
print(Output)
print("-"*100)

g, h = 13+1, 270+1;
Out = {}

for i in range (270):
     # =============================================================== Convert Output
    if (Output[i] == 2):
        Output[i]  = 1
    elif (Output[i] == 1):
        Output[i] = 0
    Out[i+1] = Output[i]
    print("Out[%d] = " %(i+1), Out[i+1])
print("-"*100, i)
```

3.1 CREATE WEIGHTS INPUT LAYER & HIDDEN LAYER

```
# Creates weight input layer
print("==================================================> Creates weight input layer")

for i in range (13+1):
    for j in range (1, 13+1):
        w[i][j] = random.random()
        print("w[%d][%d] = " %(i, j), w[i][j])
print("-"*100)
```

```
# Creates weight hidden layer
print("==========================================================> Creates weight 
hidden layer")

for i in range (13+1):
    wh[i] = random.random()
    print("wh[%d] = " %i, wh[i])
print("-"*100)
```

4.1 LEARNING ➔FORWARD🡐

```
# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> Learning
print("==========================================================> Learning")

for i in range(1, epoch):
    for j in range (1, 270+1):                # 270 = Jumlah dataset (270 input)
        # ========================================================== Forward

        # ========================================================== Input Layer
        for k in range (1, 13+1):                 # 1, 13+1 = 13 - Jumlah hidden layer
            H[k] = Bias*w[0][k]
            for n in range (1, 13+1):              # 1,13+1 = 13 (Jumlah Input)
                H[k] = H[k] + (I[j][n]*w[n][k])
            # ========================================================== Sigmoid Output
            H[k] = 1 / (1 + math.exp(-H[k]))
```

4.1 LEARNING ➔FORWARD🡨

```
# ===================================================================== Hidden Layer 1
        SH = Bias*wh[0]
        for k in range (1, 13+1):
            SH = SH + (H[k]*wh[k])
        # ============================================================ Sigmoid Output
        C = 1 / (1 + math.exp(-SH))
        # print("C= ", C, "Out= ", Out[j])
        # >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> Learning Performance Analysis
        MSE = MSE + (Out[j] - C)**2
```

4.2 LEARNING ➔BACKWARD🡐

```
# ======================================================================= Backward


        # >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> Differensial
        dC = C*(1 - C)*(Out[j] - C)               # Output = TargetC
        for k in range (1, 13+1):
            dH[k] = H[k]*(1 - H[k])*wh[k]*dC
```

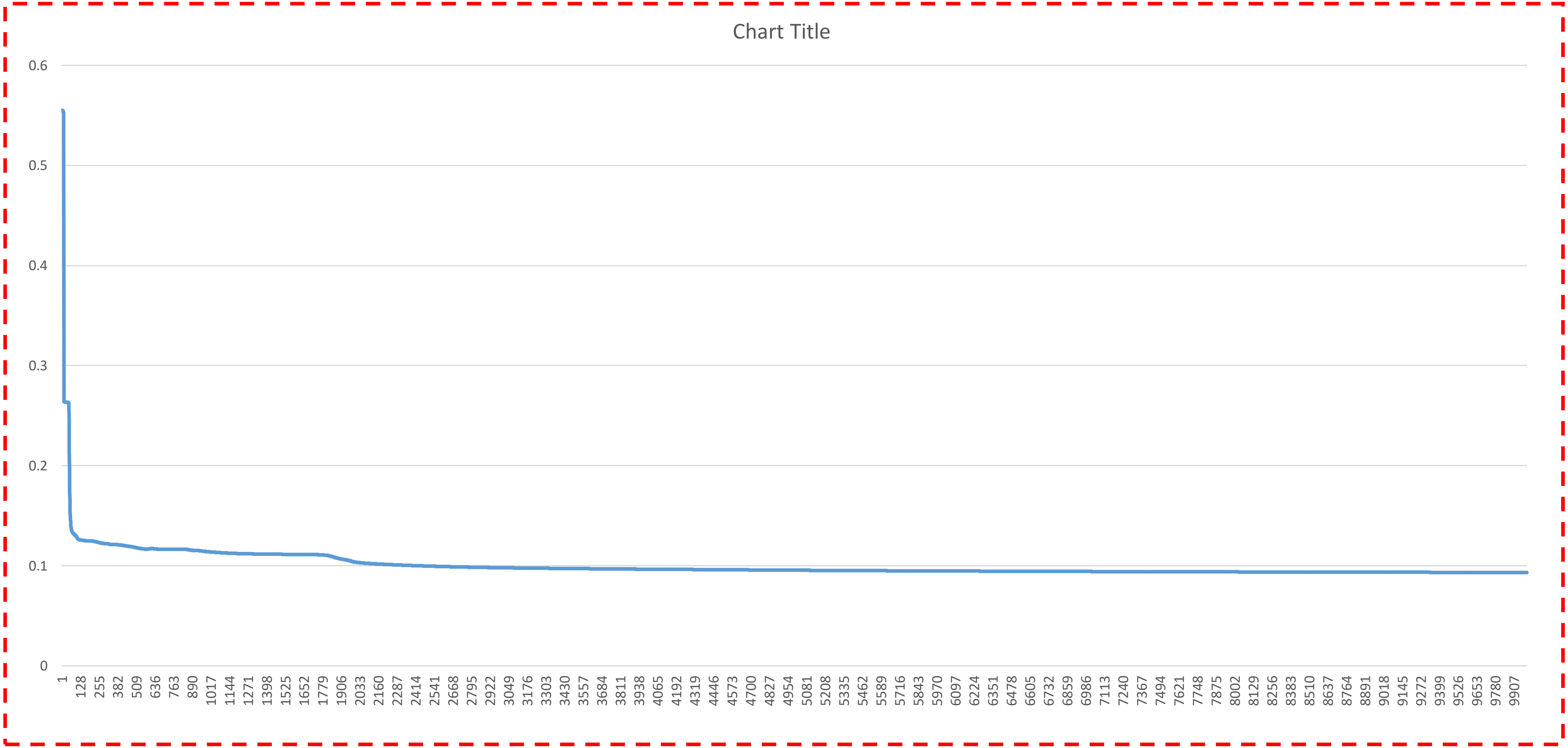
```
# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> Update Weight Hidden Layer
        H[0] = Bias
        for k in range (13+1):               # 13+1 = Jumlah Hidden layer
            dwH[k] = miu*H[k]*dC
            # Update Weight
            wh[k] = wh[k] + dwH[k]
```

```
# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> Update Weight Input Layer
        for k in range (1, 13+1):      # 1, 13+1 = 13 (Jumlah Input)
            for m in range (1, 13+1):      # 1, 13+1 = 13 (Jumlah Hidden Layer)
                dw[k][m] = miu*I[j][k]*dH[m]
                # Update Weight
                w[k][m] = w[k][m] + dw[k][m]
```

4.2 LEARNING → ERROR MSE ←

```
# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> Learning Performance Analysis
    MSE = MSE / 270    # 270 = jumlah baris data
    print("%.12f => MSE %d " %(MSE, i))
    #print(MSE)
    MSE = 0
```

4.2 LEARNING ➔ERROR MSE🡐



5. GET OPTIMUM WEIGHTS

```
# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> Final Weight
print()
print("==========================================================> Final Weight")

# weight input layer
for i in range (13+1):
    for j in range (1, 13+1):
        print("w[%d][%d] = " %(i, j), w[i][j])
print("-"*100)

# weight hidden layer
for i in range (13+1):
    print("wh[%d] = " %i, wh[i])
print("-"*100)
```

5. GET OPTIMUM WEIGHTS

===========================> Final Weight
w[0][1] = 0.10173769002006494
w[0][2] = 0.9929962034836618
w[0][3] = 0.46445803568005317
w[0][4] = 0.21339313172735097
w[0][5] = 0.09232006630712886
w[0][6] = 0.42003742994769944
w[0][7] = 0.8377910119721461
w[0][8] = 0.875361500154096
w[0][9] = 0.5025274536743032
w[0][10] = 0.6572871669827787
w[0][11] = 0.5407291346903219
w[0][12] = 0.7317753302774509
w[0][13] = 0.2365283673444344
w[1][1] = 1.0078036055349147
w[1][2] = 0.19732905535401254
w[1][3] = 0.39399787117579765
w[1][4] = 0.7707759859564876
w[1][5] = -1.875129083852956
w[1][6] = 0.6798386103463409
w[1][7] = 0.9842597958546563
w[1][8] = 0.006201067590528075
w[1][9] = -8.925166744753387
w[1][10] = 0.7537607463795805
w[1][11] = 0.8042811003587109
w[1][12] = 0.31660063447364833
w[1][13] = 0.44809020998848825

w[13][3] = 1.0071716520229952
w[13][4] = 0.10700245483023144
w[13][5] = 4.897137551664517
w[13][6] = 0.3053732612925707
w[13][7] = 0.4171503496470268
w[13][8] = 0.5656933521103874
w[13][9] = 5.285189910689762
w[13][10] = 0.964320765588564
w[13][11] = 0.8343164860359066
w[13][12] = 0.4946162931203695
w[13][13] = 0.857484994829693

----
wh[0] = 0.09465600388881003
wh[1] = -0.5681264012991167
wh[2] = 0.10710523670164758
wh[3] = -0.4697932826116844
wh[4] = -0.20600406408394265
wh[5] = 2.4683899034434096
wh[6] = -0.6940598266154612
wh[7] = 0.0900978640588572
wh[8] = 0.3286937436087643
wh[9] = 4.5178453767015885
wh[10] = 0.15768453972374105
wh[11] = -0.7567484010359258
wh[12] = -0.049569631078456 45
wh[13] = -0.46543835529402866

6. TESTING INPUT DATA

```
# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> Check Re-Check

print()
print("="*100)
print("==========================================================> Final Result")

for j in range (1, 270+1):                # 270 = Jumlah dataset (270 input)

        # ========================================================== Forward
        # ========================================================== Input Layer
        for k in range (1,13+1):             # 1,13+1 = 13 Jumlah hidden layer

            H[k] = Bias*w[0][k]
            for n in range (1, 13+1):          # 1, 13+1 = 13 (Jumlah Input)
                H[k] = H[k] + (I[j][n]*w[n][k])
            # ========================================================== Sigmoid Output
            H[k] = ( 1 / (1 + math.exp(-H[k])))
```

6. TESTING INPUT DATA

```
# =================================================================== Hidden Layer 1
        SH = Bias*wh[0]
        for k in range (1, 13+1):
            SH = SH + (H[k]*wh[k])
        # =========================================================== Sigmoid Output
        Cout[j] = 1 / (1 + math.exp(-SH))
        # =========================================================== Output Threshold
        if(Cout[j] <= 0.5):
            CoutTh[j] = 0
        else:
            CoutTh[j] = 1
```

7. FINAL RESULT

```
# ===============================================================> Final Result
print()

for j in range (1, 270+1):                          # 270 = Jumlah dataset (270 input)
    print("%.2f "*13 %(I[j][1], I[j][2], I[j][3], I[j][4], I[j][5], I[j][6], I[j][7],
I[j][8], I[j][9], I[j][10], I[j][11], I[j][12], I[j][13]), " || Output= %.9f"
%(Cout[j]), " || CoutTh= ", CoutTh[j], " || Out Real = ", Out[j])
    # ============================== Calculate Error (Positive/Negative)
    if (CoutTh[j] == Out[j]):
        TrueC += 1
    else:
        FalseC += 1
```

```
# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> Print Error
Error = FalseC/(TrueC+FalseC)

print()
print("==============================================================> Error")
print("True Data = %d || False Data = %d || Error = %f" %(TrueC, FalseC, Error))
```

7. FINAL RESULT

```
=====================================================================================
=================================================> Final Result

7.00 1.00 4.00 1.30 3.22 0.00 2.00 1.09 0.00 2.40 2.00 3.00 3.00  || Output= 0.986831984 || CoutTh=  1 || Out Real =  1
6.70 0.00 3.00 1.15 5.64 0.00 2.00 1.60 0.00 1.60 2.00 0.00 7.00  || Output= 0.083863392 || CoutTh=  0 || Out Real =  0
5.70 1.00 2.00 1.24 2.61 0.00 0.00 1.41 0.00 0.30 1.00 0.00 7.00  || Output= 0.508551750 || CoutTh=  1 || Out Real =  1
6.40 1.00 4.00 1.28 2.63 0.00 0.00 1.05 1.00 0.20 2.00 1.00 7.00  || Output= 0.889569509 || CoutTh=  1 || Out Real =  0
7.40 0.00 2.00 1.20 2.69 0.00 2.00 1.21 1.00 0.20 1.00 1.00 3.00  || Output= 0.080934945 || CoutTh=  0 || Out Real =  0
6.50 1.00 4.00 1.20 1.77 0.00 0.00 1.40 0.00 0.40 1.00 0.00 7.00  || Output= 0.080800518 || CoutTh=  0 || Out Real =  0
5.60 1.00 3.00 1.30 2.56 1.00 2.00 1.42 1.00 0.60 2.00 1.00 6.00  || Output= 0.977700585 || CoutTh=  1 || Out Real =  1
5.90 1.00 4.00 1.10 2.39 0.00 2.00 1.42 1.00 1.20 2.00 1.00 7.00  || Output= 0.989550551 || CoutTh=  1 || Out Real =  1
6.00 1.00 4.00 1.40 2.93 0.00 2.00 1.70 0.00 1.20 2.00 2.00 7.00  || Output= 0.989577162 || CoutTh=  1 || Out Real =  1
6.30 0.00 4.00 1.50 4.07 0.00 2.00 1.54 0.00 4.00 2.00 3.00 7.00  || Output= 0.989592151 || CoutTh=  1 || Out Real =  1
5.90 1.00 4.00 1.35 2.34 0.00 0.00 1.61 0.00 0.50 2.00 0.00 7.00  || Output= 0.080801248 || CoutTh=  0 || Out Real =  0
5.30 1.00 4.00 1.42 2.26 0.00 2.00 1.11 1.00 0.00 1.00 0.00 7.00  || Output= 0.889762753 || CoutTh=  1 || Out Real =  0
4.40 1.00 3.00 1.40 2.35 0.00 2.00 1.80 0.00 0.00 1.00 0.00 3.00  || Output= 0.080798885 || CoutTh=  0 || Out Real =  0
6.10 1.00 1.00 1.34 2.34 0.00 0.00 1.45 0.00 2.60 2.00 2.00 3.00  || Output= 0.509217076 || CoutTh=  1 || Out Real =  1
5.70 0.00 4.00 1.28 3.03 0.00 2.00 1.59 0.00 0.00 1.00 1.00 3.00  || Output= 0.080806924 || CoutTh=  0 || Out Real =  0
```

Input | Output | Output Threshold | Output Real

```
=================================================================> Error
True Data = 232 || False Data = 38 || Error = 0.140741
```