Notebook Executions

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
#Perlu sklearn hanya untuk confussion matrix
from sklearn import metrics
from sklearn.metrics import confusion_matrix
import itertools

np.set_printoptions(threshold=np.inf)

%matplotlib inline
%load_ext autoreload
%autoreload 2
```

```
def plotConfussionMatrix(a,b,t):
    cf =confusion_matrix(a,b)
    plt.imshow(cf,cmap=plt.cm.YlGnBu,interpolation='nearest')
    plt.colorbar()
    plt.title(t)
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    tick_marks = np.arange(len(set(a))) # length of classes
    class_labels = ['0','1']
    plt.xticks(tick_marks,class_labels)
    plt.yticks(tick_marks,class_labels)
    thresh = cf.max() / 2.
    for i,j in itertools.product(range(cf.shape[0]),range(cf.shape[1])):
        plt.text(j,i,format(cf[i,j],'d'),horizontalalignment='center',color='wh
    plt.show();
```

```
def Sigmoid(Z):
    return 1/(1+np.exp(-Z))

def derivativeSigmoid(Z):
    sigma= 1/(1+np.exp(-Z))
    derivativeZ = sigma * (1-sigma)
    return derivativeZ

class ANN:
    def __init__(self, x, y):
        self.debug = 0;

        self.inputX=x
        self.outputY=y
        self.networkOutputY=np.zeros((1,self.outputY.shape[1]))

        self.Layer=2
        self.dimentions = [16, 25, 1]
        self.parameters = {}
        self.cache = {}
```

```
21
22          self.gradient = {}
23
24          self.loss = []
25          self.learningRate=0.1
26          self.sample = self.outputY.shape[1]
27          self.threshold=0.5
28
29      def neuralAttribute(self):
30          np.random.seed(1)
31          self.parameters['W1'] = np.random.randn(self.dimentions[1], self.diment
32          self.parameters['b1'] = np.zeros((self.dimentions[1], 1))
33          self.parameters['W2'] = np.random.randn(self.dimentions[2], self.diment
34          self.parameters['b2'] = np.zeros((self.dimentions[2], 1))
35
36          return
37
38      def forwardPropagation(self):
39          Z1 = self.parameters['W1'].dot(self.inputX) + self.parameters['b1']
40          A1 = Sigmoid(Z1)
41          self.cache['Z1'],self.cache['A1']=Z1,A1
42
43          Z2 = self.parameters['W2'].dot(A1) + self.parameters['b2']
44          A2 = Sigmoid(Z2)
45          self.cache['Z2'],self.cache['A2']=Z2,A2
46
47          self.networkOutputY=A2
48          loss=self.calculateLoss(A2)
49          return self.networkOutputY, loss
50
51      def calculateLoss(self,networkOutputY):
52          loss = (1./self.sample) * (-np.dot(self.outputY,np.log(networkOutputY).
53          return loss
54
55      def backwardPropagation(self):
56          deltaLoss_networkOutputY = - (np.divide(self.outputY, self.networkOutpu
57
58          deltaLoss_Z2 = deltaLoss_networkOutputY * derivativeSigmoid(self.cache[
59          deltaLoss_A1 = np.dot(self.parameters["W2"].T,deltaLoss_Z2)
60          deltaLoss_W2 = 1./self.cache['A1'].shape[1] * np.dot(deltaLoss_Z2,self.
61          deltaLoss_b2 = 1./self.cache['A1'].shape[1] * np.dot(deltaLoss_Z2, np.o
62
63          deltaLoss_Z1 = deltaLoss_A1 * derivativeSigmoid(self.cache['Z1'])
64          deltaLoss_A0 = np.dot(self.parameters["W1"].T,deltaLoss_Z1)
65          deltaLoss_W1 = 1./self.inputX.shape[1] * np.dot(deltaLoss_Z1,self.input
66          deltaLoss_b1 = 1./self.inputX.shape[1] * np.dot(deltaLoss_Z1, np.ones([
67
68          self.parameters["W1"] = self.parameters["W1"] - self.learningRate * del
69          self.parameters["b1"] = self.parameters["b1"] - self.learningRate * del
70          self.parameters["W2"] = self.parameters["W2"] - self.learningRate * del
71          self.parameters["b2"] = self.parameters["b2"] - self.learningRate * del
72
73          return
74
75
```

```
    def predict(self,x, y):
        self.inputX=x
        self.outputY=y
        comp = np.zeros((1,x.shape[1]))
        predict, loss= self.forwardPropagation()

        for i in range(0, predict.shape[1]):
            if predict[0,i] > self.threshold: comp[0,i] = 1
            else: comp[0,i] = 0

        print("Acc: " + str(np.sum((comp == y)/x.shape[1])))

        return comp

    def train(self,inputX, outputY, iter = 3000):
        np.random.seed(1)

        self.neuralAttribute()

        for i in range(0, iter):
            networkOutputY, loss=self.forwardPropagation()
            self.backwardPropagation()

            if i % 100 == 0:
                print ("Cost after iteration %i: %f" %(i, loss))
                self.loss.append(loss)

        plt.plot(np.squeeze(self.loss))
        plt.ylabel('Loss')
        plt.xlabel('Iter')
        plt.title("Lr =" + str(self.learningRate))
        plt.show()

        return
```

```
df = pd.read_csv('UAS_119140047_119140052_119140002_NOMOR1.csv')

df
```

	Age	Gender	Polyuria	Polydipsia	sudden weight loss	weakness	Polyphagia	Genita thrus
0	40	Male	No	Yes	No	Yes	No	N
1	58	Male	No	No	No	Yes	No	N
2	41	Male	Yes	No	No	Yes	Yes	N
3	45	Male	No	No	Yes	Yes	Yes	Ye

```
df['Gender'] =df['Gender'].map({'Male' : 1 , 'Female':0})
df['Polyuria'] =df['Polyuria'].map({'Yes' : 1 , 'No':0})
df['Polydipsia'] =df['Polydipsia'].map({'Yes' : 1 , 'No':0})
df['sudden weight loss'] =df['sudden weight loss'].map({'Yes' : 1 , 'No':0})
df['weakness'] =df['weakness'].map({'Yes' : 1 , 'No':0})
df['Polyphagia'] =df['Polyphagia'].map({'Yes' : 1 , 'No':0})
df['Genital thrush'] =df['Genital thrush'].map({'Yes' : 1 , 'No':0})
df['visual blurring'] =df['visual blurring'].map({'Yes' : 1 , 'No':0})
df['Itching'] =df['Itching'].map({'Yes' : 1 , 'No':0})
df['Irritability'] =df['Irritability'].map({'Yes' : 1 , 'No':0})
df['delayed healing'] =df['delayed healing'].map({'Yes' : 1 , 'No':0})
df['partial paresis'] =df['partial paresis'].map({'Yes' : 1 , 'No':0})
df['muscle stiffness'] =df['muscle stiffness'].map({'Yes' : 1 , 'No':0})
df['Alopecia'] =df['Alopecia'].map({'Yes' : 1 , 'No':0})
df['Obesity'] =df['Obesity'].map({'Yes' : 1 , 'No':0})
df['class'] =df['class'].map({'Positive' : 1 , 'Negative':0})

df['Age'] = (df['Age']-df['Age'].mean())/df['Age'].std()

df = df.astype(float)
scaled_df=df
df.head(100)

```

	Age	Gender	Polyuria	Polydipsia	sudden weight loss	weakness	Polyphagia	Ge... tl...
0	-0.660731	1.0	0.0	1.0	0.0	1.0	0.0	
1	0.820572	1.0	0.0	0.0	0.0	1.0	0.0	

```
df.tail(180)
```

	Age	Gender	Polyuria	Polydipsia	sudden weight loss	weakness	Polyphagia	Geni... thi...
340	-0.743025	0.0	1.0	1.0	1.0	1.0	1.0	
341	-0.413847	0.0	1.0	1.0	1.0	1.0	1.0	
342	-1.072204	0.0	1.0	1.0	0.0	1.0	0.0	
343	-0.084668	0.0	0.0	0.0	1.0	1.0	1.0	
344	1.067456	0.0	1.0	0.0	0.0	0.0	1.0	
...	...	...	...	...	...	...	...	
515	-0.743025	0.0	1.0	1.0	1.0	0.0	1.0	
516	-0.002374	0.0	1.0	1.0	1.0	1.0	1.0	
517	0.820572	0.0	1.0	1.0	1.0	1.0	1.0	
518	-1.319087	0.0	0.0	0.0	0.0	1.0	0.0	
519	-0.496141	1.0	0.0	0.0	0.0	0.0	0.0	

180 rows × 17 columns

```
names = ['Age' , 'Gender' , 'Polyuria',  'Polydipsia' , 'sudden weight loss'
scaled_df = pd.DataFrame(scaled_df, columns=names)
```

```
scaled_df.iloc[0:17, 0:17].plot.hist(alpha=1)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f16d1e7a290>
```


```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 520 entries, 0 to 519
Data columns (total 17 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Age                 520 non-null    float64
 1   Gender              520 non-null    float64
 2   Polyuria            520 non-null    float64
 3   Polydipsia          520 non-null    float64
 4   sudden weight loss  520 non-null    float64
 5   weakness            520 non-null    float64
 6   Polyphagia          520 non-null    float64
 7   Genital thrush      520 non-null    float64
 8   visual blurring     520 non-null    float64
 9   Itching             520 non-null    float64
 10  Irritability        520 non-null    float64
 11  delayed healing     520 non-null    float64
 12  partial paresis     520 non-null    float64
 13  muscle stiffness    520 non-null    float64
 14  Alopecia            520 non-null    float64
 15  Obesity             520 non-null    float64
 16  class               520 non-null    float64
dtypes: float64(17)
memory usage: 69.2 KB
```

```
1 x=scaled_df.iloc[0:340,0:16].values.transpose()
2 y=df.iloc[0:340,16:].values.transpose()
3
4 xval=scaled_df.iloc[341:520,0:16].values.transpose()
5 yval=df.iloc[341:520,16:].values.transpose()
6
7 print(df.shape, x.shape, y.shape, xval.shape, yval.shape)
8 # print(df.iloc[341:520,16:])
9 neural_network = ANN(x,y)
10 neural_network.learningRate=0.1
11 neural_network.dimentions = [16, 25, 1]
```

```
(520, 17) (16, 340) (1, 340) (16, 179) (1, 179)
```

```
1 neural_network.train(x, y, iter = 25000)
```

```
Cost after iteration 0: 0.742204
Cost after iteration 100: 0.508615
Cost after iteration 200: 0.405512
Cost after iteration 300: 0.329312
Cost after iteration 400: 0.281200
Cost after iteration 500: 0.251775
Cost after iteration 600: 0.233031
Cost after iteration 700: 0.220388
Cost after iteration 800: 0.211408
Cost after iteration 900: 0.204751
Cost after iteration 1000: 0.199644
Cost after iteration 1100: 0.195615
Cost after iteration 1200: 0.192361
Cost after iteration 1300: 0.189680
Cost after iteration 1400: 0.187429
Cost after iteration 1500: 0.185507
Cost after iteration 1600: 0.183838
Cost after iteration 1700: 0.182364
Cost after iteration 1800: 0.181040
Cost after iteration 1900: 0.179830
Cost after iteration 2000: 0.178705
Cost after iteration 2100: 0.177642
Cost after iteration 2200: 0.176621
Cost after iteration 2300: 0.175626
Cost after iteration 2400: 0.174645
Cost after iteration 2500: 0.173667
Cost after iteration 2600: 0.172682
Cost after iteration 2700: 0.171685
Cost after iteration 2800: 0.170669
Cost after iteration 2900: 0.169631
Cost after iteration 3000: 0.168568
Cost after iteration 3100: 0.167479
Cost after iteration 3200: 0.166363
Cost after iteration 3300: 0.165220
Cost after iteration 3400: 0.164050
Cost after iteration 3500: 0.162853
Cost after iteration 3600: 0.161631
Cost after iteration 3700: 0.160384
Cost after iteration 3800: 0.159112
Cost after iteration 3900: 0.157817
Cost after iteration 4000: 0.156499
Cost after iteration 4100: 0.155159
Cost after iteration 4200: 0.153798
Cost after iteration 4300: 0.152415
Cost after iteration 4400: 0.151012
```

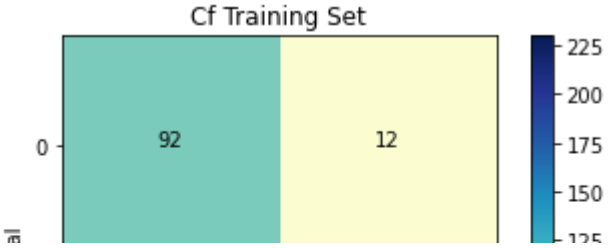
```
pred_train = neural_network.predict(x, y)
print("Prediciton accuracy detail : {}".format(pred_train))
pred_test = neural_network.predict(xval, yval)
print("Test accuracy detail : {}".format(pred_test))
```

```
Acc: 0.9970588235294118
Prediciton accuracy detail : [[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1
  1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
  1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
  1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
  1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
  1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
```

```
  1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
  1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
  1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
  1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 1. 1. 1.]]
Acc: 0.9944134078212291
Test accuracy detail : [[1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1
  1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
  1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1.
  1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 0.]]
```

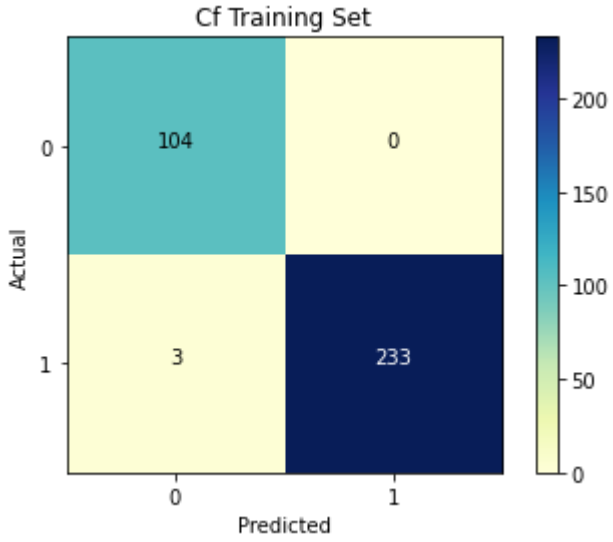
```
 1 neural_network.threshold=0.5
 2
 3 neural_network.inputX,neural_network.outputY=x, y
 4 target=np.around(np.squeeze(y), decimals=0).astype(np.int)
 5 predicted=np.around(np.squeeze(neural_network.predict(x,y)), decimals=0).astype
 6 plotConfussionMatrix(target,predicted,'Cf Training Set')
 7
 8 neural_network.inputX,neural_network.outputY=xval, yval
 9 target=np.around(np.squeeze(yval), decimals=0).astype(np.int)
10 predicted=np.around(np.squeeze(neural_network.predict(xval,yval)), decimals=0).
11 plotConfussionMatrix(target,predicted,'Cf Validation Set')
```

```
Acc: 0.9470588235294117
```

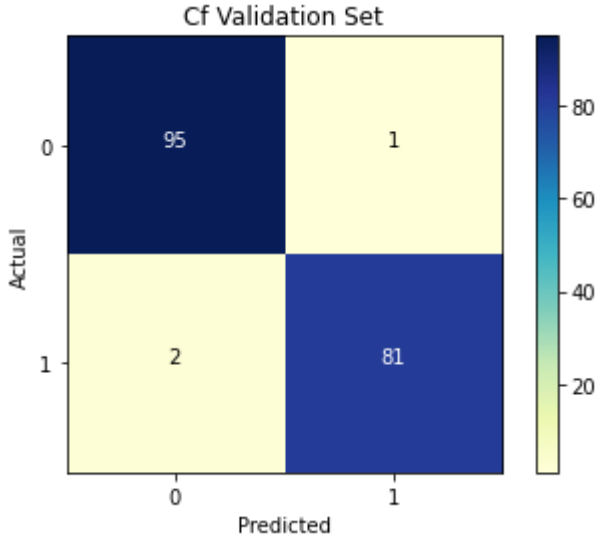



```
1 neural_network.threshold=0.7
2
3 neural_network.inputX,neural_network.Y=x, y
4 target=np.around(np.squeeze(y), decimals=0).astype(np.int)
5 predicted=np.around(np.squeeze(neural_network.predict(x,y)), decimals=0).astype
6 plotConfussionMatrix(target,predicted,'Cf Training Set')
7
8 neural_network.inputX,neural_network.outputY=xval, yval
9 target=np.around(np.squeeze(yval), decimals=0).astype(np.int)
10 predicted=np.around(np.squeeze(neural_network.predict(xval,yval)), decimals=0).
11 plotConfussionMatrix(target,predicted,'Cf Validation Set')
```

```
Acc: 0.9911764705882353
```



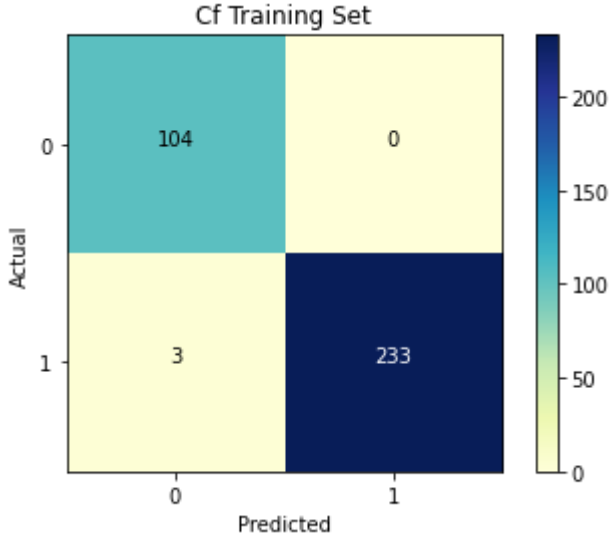

```
Acc: 0.9832402234636872
```



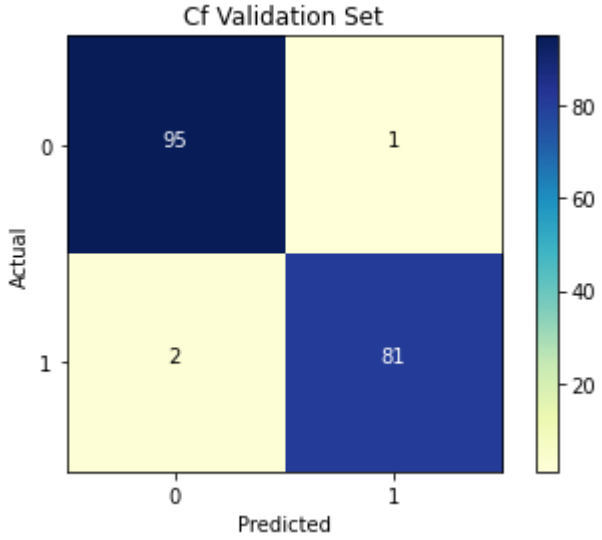

```
1 neural_network.threshold=0.9
```

```
2
3 neural_network.inputX,neural_network.outputY=x, y
4 target=np.around(np.squeeze(y), decimals=0).astype(np.int)
5 predicted=np.around(np.squeeze(neural_network.predict(x,y)), decimals=0).astype
6 plotConfussionMatrix(target,predicted,'Cf Training Set')
7
8 neural_network.inputX,neural_network.outputY=xval, yval
9 target=np.around(np.squeeze(yval), decimals=0).astype(np.int)
10 predicted=np.around(np.squeeze(neural_network.predict(xval,yval)), decimals=0).
11 plotConfussionMatrix(target,predicted,'Cf Validation Set')
```

```
Acc: 0.9911764705882353
```




```
Acc: 0.9832402234636872
```




```
1 neural_network.inputX,neural_network.outputY=xval, yval
2 yvalh, loss = neural_network.forwardPropagation()
3 print("\ny (Nilai target seharusnya) = \n",np.around(yval[:,0:179,], decimals=0
4 print("\nyh (Nilai prediksi yang didapat) = \n",np.around(yvalh[:,0:179,], deci
```

```
y (Nilai target seharusnya) =
 [[1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
  0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
  1 1 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0]]
```

```
yh (Nilai prediksi yang didapat) =
 [[1 1 1 1 1 1 1 0 0 0 0 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
  0 1 1 1 1 1 1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0
  0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1
  1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
  1 1 1 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0]]
```

```
# xvall = np.array([[-0.3, 1,0,0,0,0,1,1,0,1,0,0,0,1,1,1]]).T ga usah diubah !!
xvall = np.array([[40, 1,0,0,0,0,1,1,0,1,1,0,0,1,1,1]]).T
yvall = np.array([1]).T
# pred_test = neural_network.predict(xvall, yvall)
# print("Test accuracy detail : {}".format(pred_test))
neural_network.inputX,neural_network.outputY=xvall, yvall
yvalh, loss = neural_network.forwardPropagation()
# print("\ny (Nilai target seharusnya) = ",np.around(yvall, decimals=0).astype(
print("\nyh (Nilai prediksi yang didapat) = ",np.around(yvalh, decimals=0).asty
```

```
yh (Nilai prediksi yang didapat) =  [[1]]
```

```
# Dhifaf Athiyah Zhabiyan
# 119140047
# Eliza Maharani Sutowo
# 119140002
# Abdurrachman Farras
# 119140052

```

✓ 1s completed at 7:06 PM