Pokok Bahasan XI
Neural Network

Kode Pokok Bahasan: TIK.RPL03.001.007.01

Deskripsi Pokok Bahasan:
Membahas tentang Neural Network pada R dengan dataset yang diberikan.

No	Elemen Kompetensi	Indikator Kinerja	Jml Jam	Hal
1	Memahami proses backpropagation dengan neuralnet library di R	Mampu memahami konsep backpropagation dengan neuralnet pada R	1	12
2	Menerapkan Neural Network untuk melakukan Forecasting.	Mampu melakukan forecasting data menggunakan neural network	2	15

TUGAS PENDAHULUAN
Hal yang harus dilakukan dan acuan yang harus dibaca sebelum praktikum :
1. Menginstal R pada PC masing-masing praktikan.
2. Menginstal R Studio pada PC masing-masing praktikan.

DAFTAR PERTANYAAN
1. Apa itu Neural Network?

Jaringan neural adalah tipe proses machine learning, yang disebut deep learning, yang menggunakan simpul atau neuron yang saling terhubung dalam struktur berlapis yang menyerupai otak manusia.

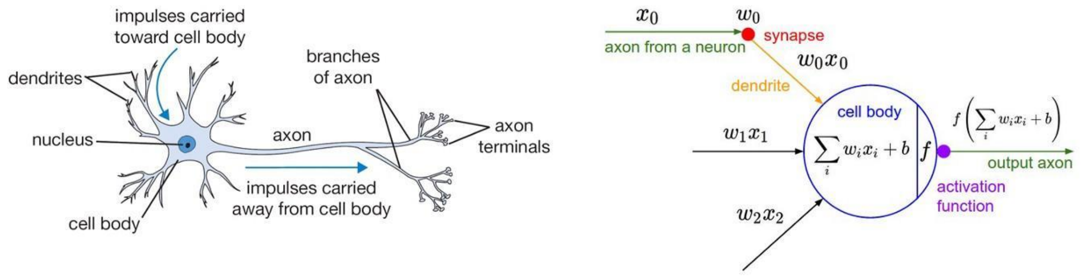
2. Bagaimana gambaran dasar sebuah Neural Network?
    1. Pengklasifikasian pola
    2. Memetakan pola yang didapat dari input ke dalam pola baru pada output
    3. Penyimpan pola yang akan dipanggil kembali
    4. Memetakan pola-pola yang sejenis
    5. Pengoptimasi permasalahan
    6. Prediksi

3. Mengapa Neural Network dibutuhkan?

Jaringan neural dapat membantu komputer membuat keputusan cerdas dengan bantuan manusia yang terbatas. Keputusan cerdas dapat dibuat karena jaringan neural dapat mempelajari dan memodelkan hubungan antara data input dan output yang nonlinier dan kompleks.

TEORI SINGKAT

Neural network adalah model yang terinspirasi oleh bagaimana neuron dalam otak manusia bekerja. Tiap neuron pada otak manusia saling berhubungan dan informasi mengalir dari setiap neuron tersebut. Gambar di bawah adalah ilustrasi neuron dengan model matematisnya.



Tiap neuron menerima input dan melakukan operasi dot dengan sebuah weight, menjumlahkannya (weighted sum) dan menambahkan bias. Hasil dari operasi ini akan dijadikan parameter dari activation function yang akan dijadikan output dari neuron tersebut.

LAB SETUP

Hal yang harus disiapkan dan dilakukan oleh praktikan untuk menjalankan praktikum modul ini.

1. Menginstall library yang dibutuhkan untuk mengerjakan modul.
2. Menjalankan R Studio.

ELEMEN KOMPETENSI I

Deskripsi:
Memahami proses backpropagation dengan neuralnet library di R.

Kompetensi Dasar:
Mampu memahami konsep backpropagation dengan neuralnet pada R.

Latihan 1.1.1

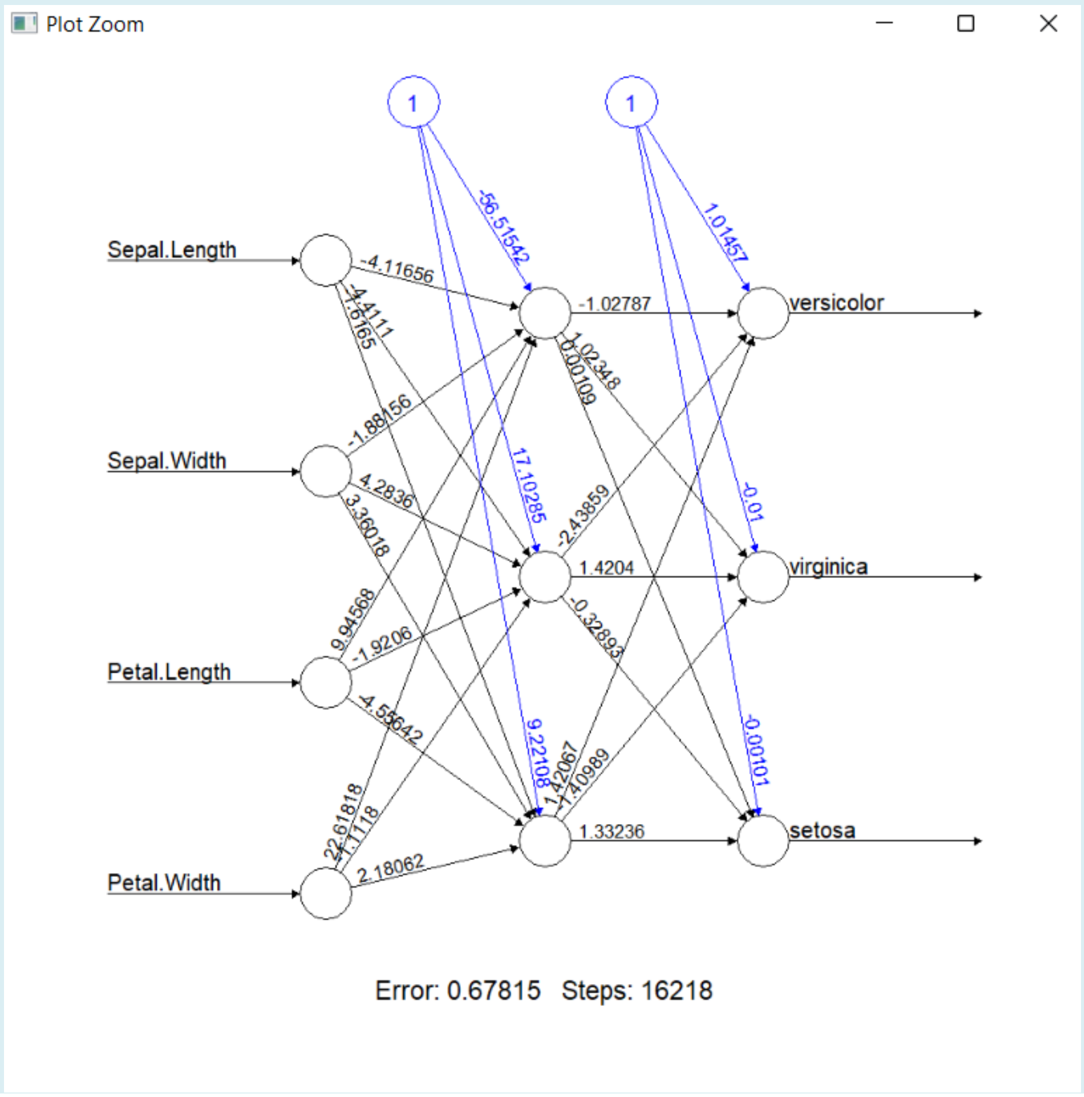
Penjelasan Singkat :
Pada latihan ini anda akan diminta untuk mempersiapkan data dan membangun neural network pada R.

Langkah-Langkah Praktikum:

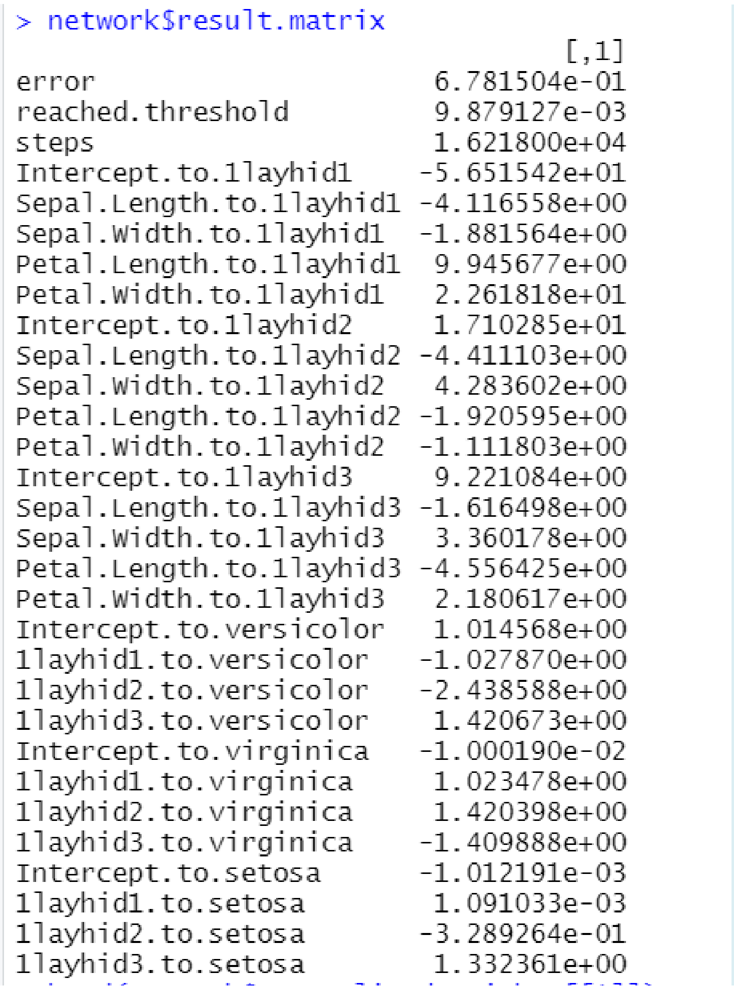
Data iris

```
ind <- sample(2, nrow(iris), replace = TRUE, prob=c(0.7, 0.3))
library(neuralnet)
trainset = iris[ind == 1,]
testset = iris[ind == 2,]
trainset$setosa = trainset$Species == "setosa"
trainset$virginica = trainset$Species == "virginica"
trainset$versicolor = trainset$ Species == "versicolor"

network = neuralnet(versicolor + virginica + setosa~ Sepal.Length + Sepal.Width +
                        Petal.Length + Petal.Width, trainset, hidden=3)

plot(network)
network$result.matrix
head(network$generalized.weights[[1]])
```

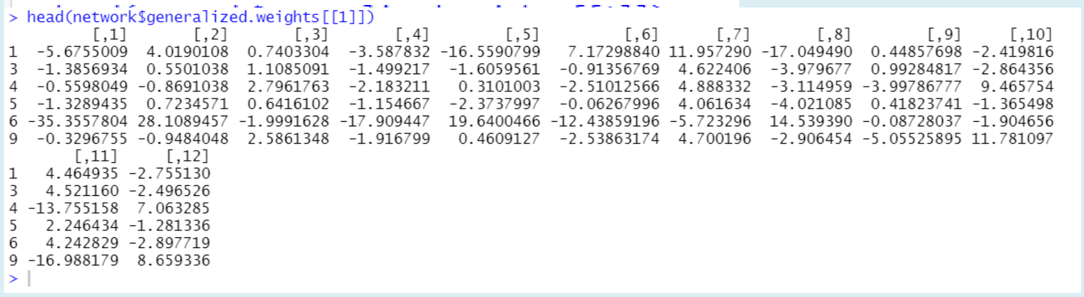
Output :



Error: 0.67815   Steps: 16218



```
> network$result.matrix
                                 [,1]
error                        6.781504e-01
reached.threshold            9.879127e-03
steps                        1.621800e+04
Intercept.to.1layhid1       -5.651542e+01
Sepal.Length.to.1layhid1    -4.116558e+00
Sepal.Width.to.1layhid1     -1.881564e+00
Petal.Length.to.1layhid1     9.945677e+00
Petal.Width.to.1layhid1      2.261818e+01
Intercept.to.1layhid2        1.710285e+01
Sepal.Length.to.1layhid2    -4.411103e+00
Sepal.Width.to.1layhid2      4.283602e+00
Petal.Length.to.1layhid2    -1.920595e+00
Petal.Width.to.1layhid2     -1.111803e+00
Intercept.to.1layhid3        9.221084e+00
Sepal.Length.to.1layhid3    -1.616498e+00
Sepal.Width.to.1layhid3      3.360178e+00
Petal.Length.to.1layhid3    -4.556425e+00
Petal.Width.to.1layhid3      2.180617e+00
Intercept.to.versicolor      1.014568e+00
1layhid1.to.versicolor      -1.027870e+00
1layhid2.to.versicolor      -2.438588e+00
1layhid3.to.versicolor       1.420673e+00
Intercept.to.virginica      -1.000190e-02
1layhid1.to.virginica        1.023478e+00
1layhid2.to.virginica        1.420398e+00
1layhid3.to.virginica       -1.409888e+00
Intercept.to.setosa         -1.012191e-03
1layhid1.to.setosa           1.091033e-03
1layhid2.to.setosa          -3.289264e-01
1layhid3.to.setosa           1.332361e+00
```



```
> head(network$generalized.weights[[1]])
         [,1]       [,2]       [,3]       [,4]        [,5]         [,6]      [,7]       [,8]        [,9]     [,10]
1  -5.6755009  4.0190108  0.7403304  -3.587832 -16.5590799   7.17298840 11.957290 -17.049490  0.44857698 -2.419816
3  -1.3856934  0.5501038  1.1085091  -1.499217  -1.6059561  -0.91356769  4.622406  -3.979677  0.99284817 -2.864356
4  -0.5598049 -0.8691038  2.7961763  -2.183211   0.3101003  -2.51012566  4.888332  -3.114959 -3.99786777  9.465754
5  -1.3289435  0.7234571  0.6416102  -1.154667  -2.3737997  -0.06267996  4.061634  -4.021085  0.41823741 -1.365498
6 -35.3557804 28.1089457 -1.9991628 -17.909447  19.6400466 -12.43859196 -5.723296  14.539390 -0.08728037 -1.904656
9  -0.3296755 -0.9484048  2.5861348  -1.916799   0.4609127  -2.53863174  4.700196  -2.906454 -5.05525895 11.781097
        [,11]     [,12]
1    4.464935 -2.755130
3    4.521160 -2.496526
4  -13.755158  7.063285
5    2.246434 -1.281336
6    4.242829 -2.897719
9  -16.988179  8.659336
> 
```

Penjelasan :

Dari neural network di atas yang berdasarkan suhu, bermain, dan kelembapan dengan hasil ya dan tidak, maka ditemukan error yaitu 2.00003 dan langkahnya yaitu steps = 28

Tugas :
Gunakan library neural net untuk membangun model backpropagation dengan input suhu dan kelembaban menggunakan data di bawah ini.

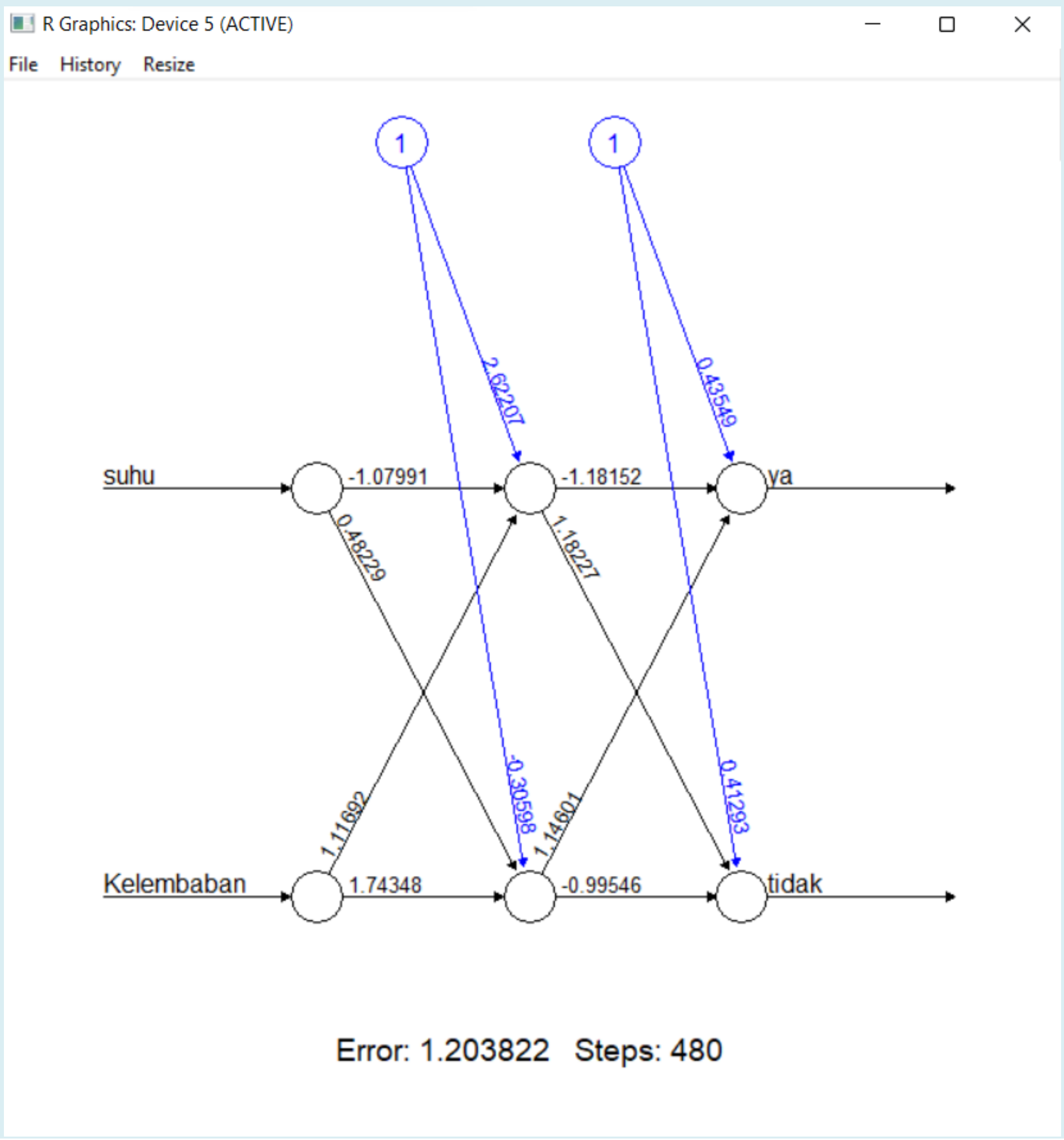
suhu	Kelembaban	bermain
69	70	ya
72	95	tidak
75	70	ya
80	90	tidak
85	85	tidak
65	70	tidak
68	80	ya
70	96	ya
71	80	tidak
75	80	ya
64	65	ya
72	90	ya
81	75	ya
83	78	ya

Script :

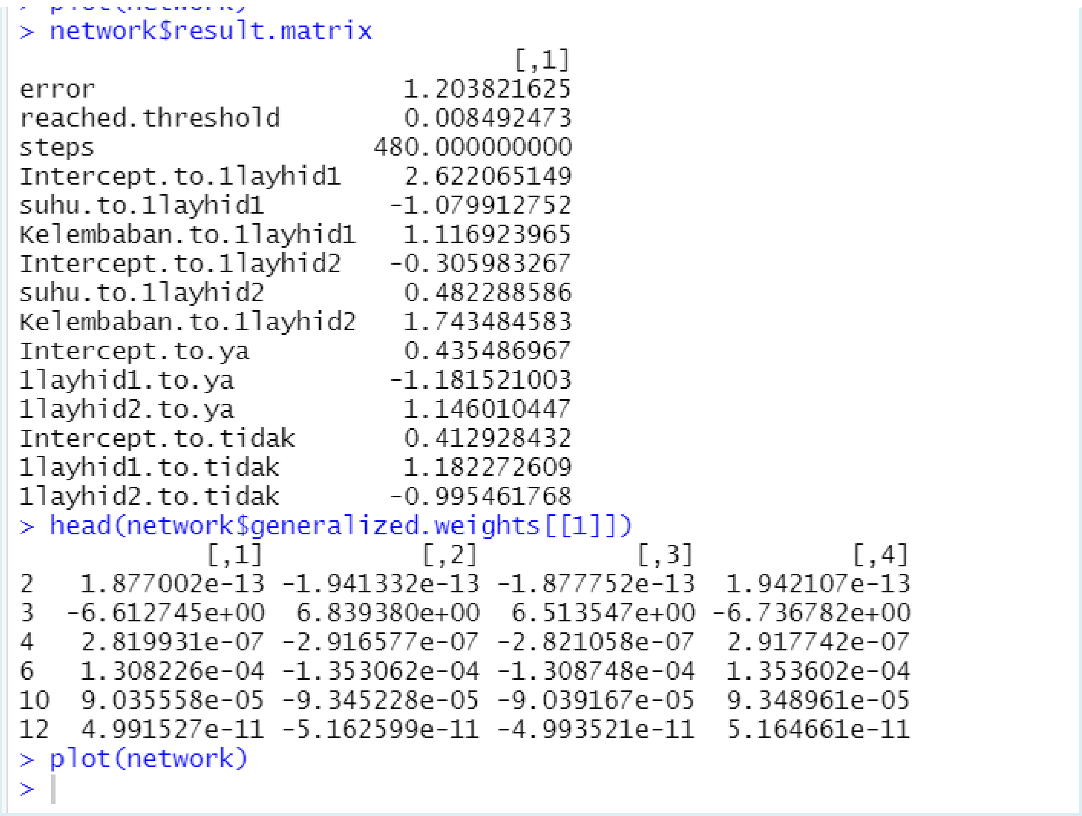
```
data_nama = read.delim("clipboard")
head(data_nama)
ind <- sample(2, nrow(data_nama), replace = TRUE, prob=c(0.7, 0.3))
trainset = data_nama[ind == 1,]
testset = data_nama[ind == 2,]
View(trainset)
View(testset)
trainset$ya = trainset$bermain == "ya"
trainset$tidak = trainset$bermain == "tidak"
network = neuralnet(ya + tidak~ suhu + Kelembaban, trainset, hidden=2)
plot(network)
```

```
network$result.matrix
head(network$generalized.weights[[1]])
```

Output :



Error: 1.203822 Steps: 480



```
> network$result.matrix
                                  [,1]
error                      1.203821625
reached.threshold          0.008492473
steps                    480.000000000
Intercept.to.1layhid1      2.622065149
suhu.to.1layhid1          -1.079912752
Kelembaban.to.1layhid1     1.116923965
Intercept.to.1layhid2     -0.305983267
suhu.to.1layhid2           0.482288586
Kelembaban.to.1layhid2     1.743484583
Intercept.to.ya            0.435486967
1layhid1.to.ya            -1.181521003
1layhid2.to.ya             1.146010447
Intercept.to.tidak         0.412928432
1layhid1.to.tidak          1.182272609
1layhid2.to.tidak         -0.995461768
> head(network$generalized.weights[[1]])
            [,1]          [,2]          [,3]          [,4]
2   1.877002e-13 -1.941332e-13 -1.877752e-13  1.942107e-13
3  -6.612745e+00  6.839380e+00  6.513547e+00 -6.736782e+00
4   2.819931e-07 -2.916577e-07 -2.821058e-07  2.917742e-07
6   1.308226e-04 -1.353062e-04 -1.308748e-04  1.353602e-04
10  9.035558e-05 -9.345228e-05 -9.039167e-05  9.348961e-05
12  4.991527e-11 -5.162599e-11 -4.993521e-11  5.164661e-11
> plot(network)
> 
```

Penjelasan :

Dari neural network di atas yang berdasarkan suhu, bermain, dan kelembapan dengan hasil ya dan tidak, maka ditemukan error yaitu 2.00003 dan langkahnya yaitu steps = 28

ELEMEN KOMPETENSI II

Deskripsi:
Menerapkan Neural Network untuk melakukan Forecasting.
Kompetensi Dasar:
Mampu melakukan forecasting data menggunakan neural network.

Latihan 1.2.1

Penjelasan Singkat :

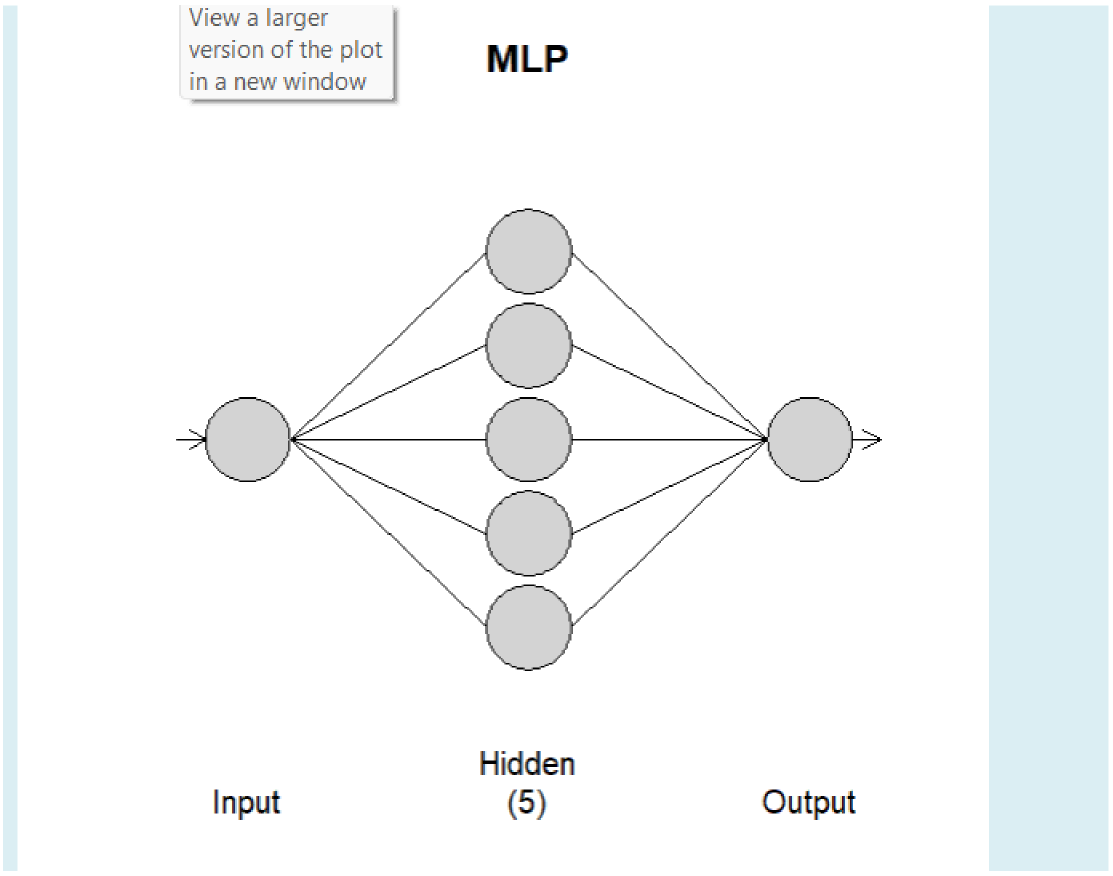
Pada latihan ini anda akan diminta untuk melakukan prediksi menggunakan R.

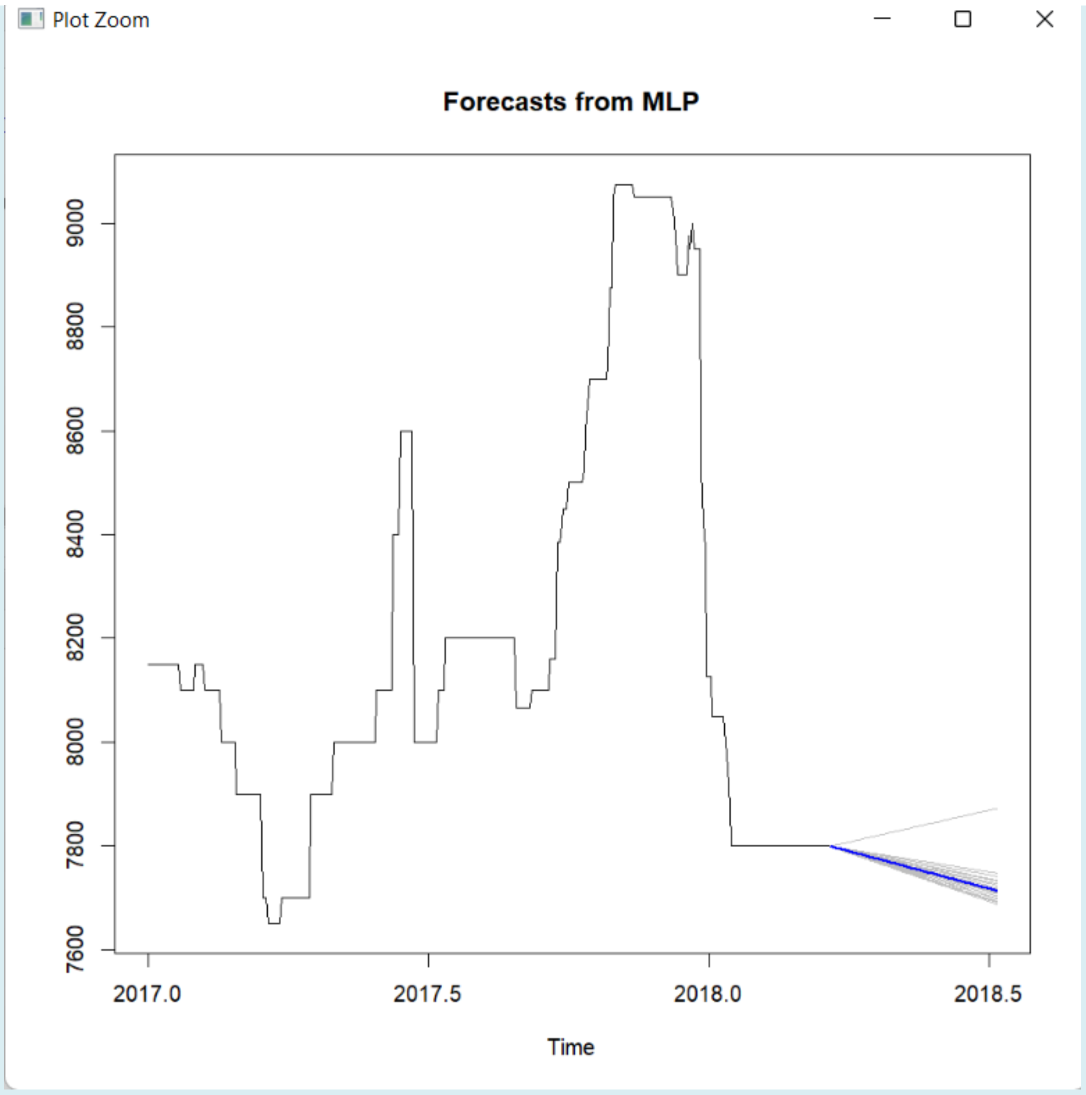
Langkah-Langkah Praktikum:

Gunakan database db_pibc_olap.sql

```
> library(RMySQL)
> library(dplyr)
> con = dbConnect(MySQL(), user = 'root', password = '', dbname = 'db_pibc_olap', host =
'localhost')
> dbListTables(con)
> myQuery <- "select * from fact_harga;"
> df <- dbGetQuery(con, myQuery)
> df1<-filter(df,SK_RICE_TYPE==10,
SK_DATE>=20170101,SK_DATE<=20171231, SK_MARKET==0)
> df2<- df1[order(df1$SK_DATE),]
> View(df2)
> tseries <- ts(df2$PRICE, start = c(2017, 1), frequency = 300)
> library(nnfor)
> library(forecast)
#MLP
> fit<-mlp(tseries)
> plot(fit)
> f2=forecast(fit, h=90)
> plot(f2)
> summary(f2)
```

Output :


View a larger version of the plot in a new window
MLP
Input
Hidden (5)
Output



Penjelasan :

Dari neural network di atas untuk melakukan forecasting sesuai dengan fact_harga, maka ditemukan error MAPE sebesar 0.1805326 dan MPE sebesar 0.002042281

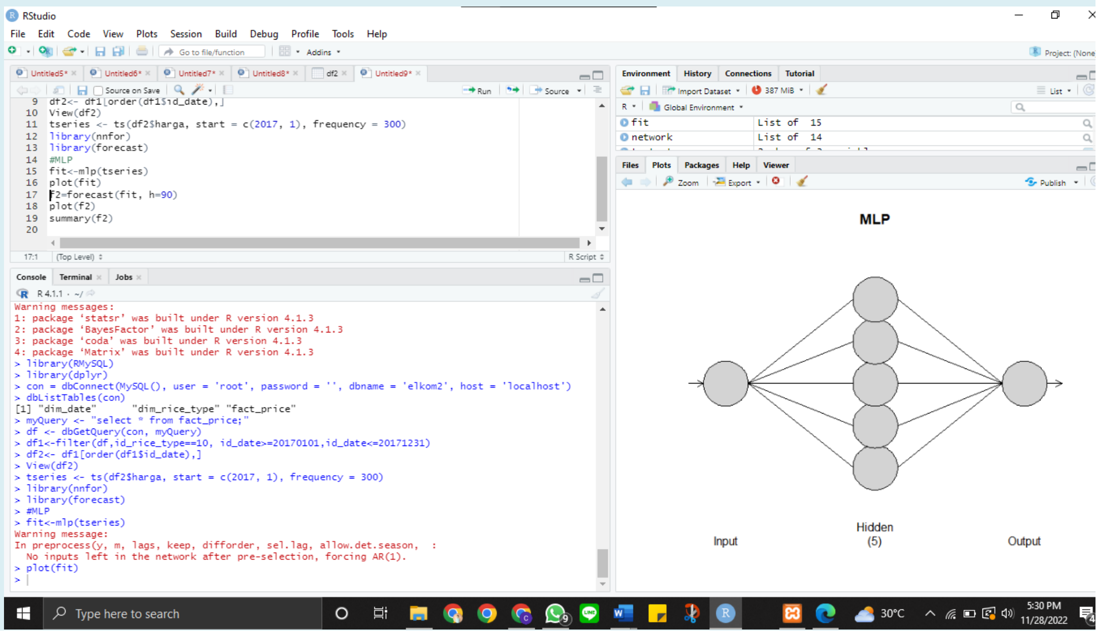
Tugas :

Gunakan script di atas untuk membangun model peramalan dengan menggunakan data pada database db_pasokanberas. Pilih interval waktu tertentu sebagai input. Bandingkan error yang terjadi antara data prediksi dengan sesungguhnya.

Script :

```
library(statsr)
library(RMySQL)
library(dplyr)
con = dbConnect(MySQL(), user = 'root', password = '', dbname = 'db_pasokanberas',
host = 'localhost')
dbListTables(con)
myQuery <- "select * from fact_price;"
df <- dbGetQuery(con, myQuery)
df1<-filter(df,id_rice_type==10, id_date>=20170101,id_date<=20171231)
df2<- df1[order(df1$id_date),]
View(df2)
tseries <- ts(df2$harga, start = c(2017, 1), frequency = 300)
library(nnfor)
library(forecast)
#MLP
fit<-mlp(tseries)
plot(fit)
f2=forecast(fit, h=90)
plot(f2)
summary(f2)
```

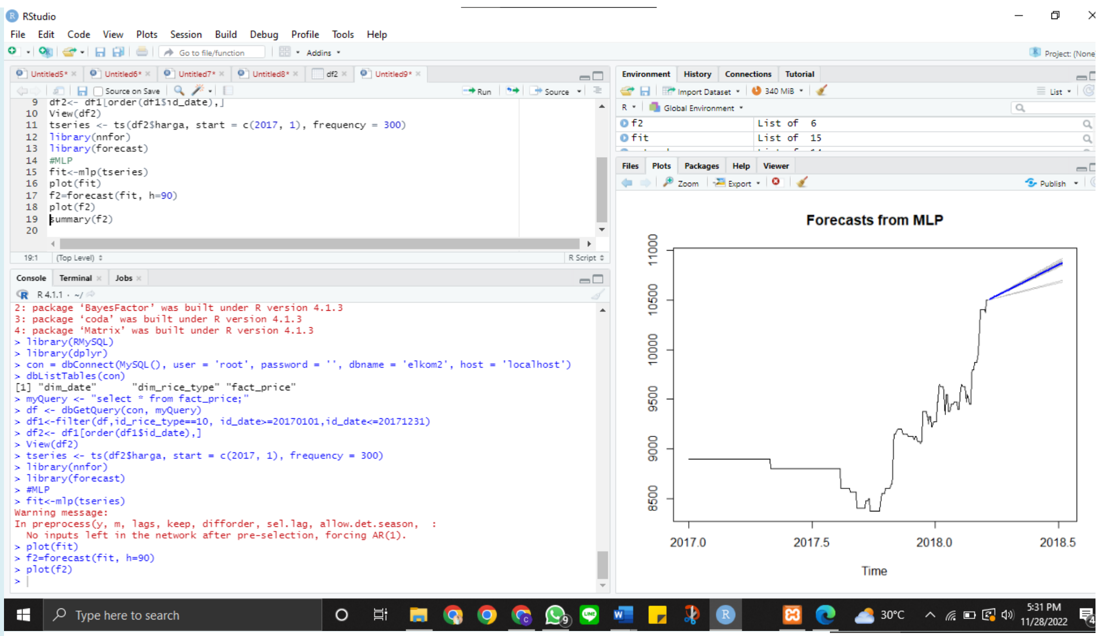
Output :



```
df2<- df1[order(df1$id_date),]
View(df2)
tseries <- ts(df2$harga, start = c(2017, 1), frequency = 300)
library(nnfor)
library(forecast)
#MLP
fit<-mlp(tseries)
plot(fit)
f2=forecast(fit, h=90)
plot(f2)
summary(f2)
```

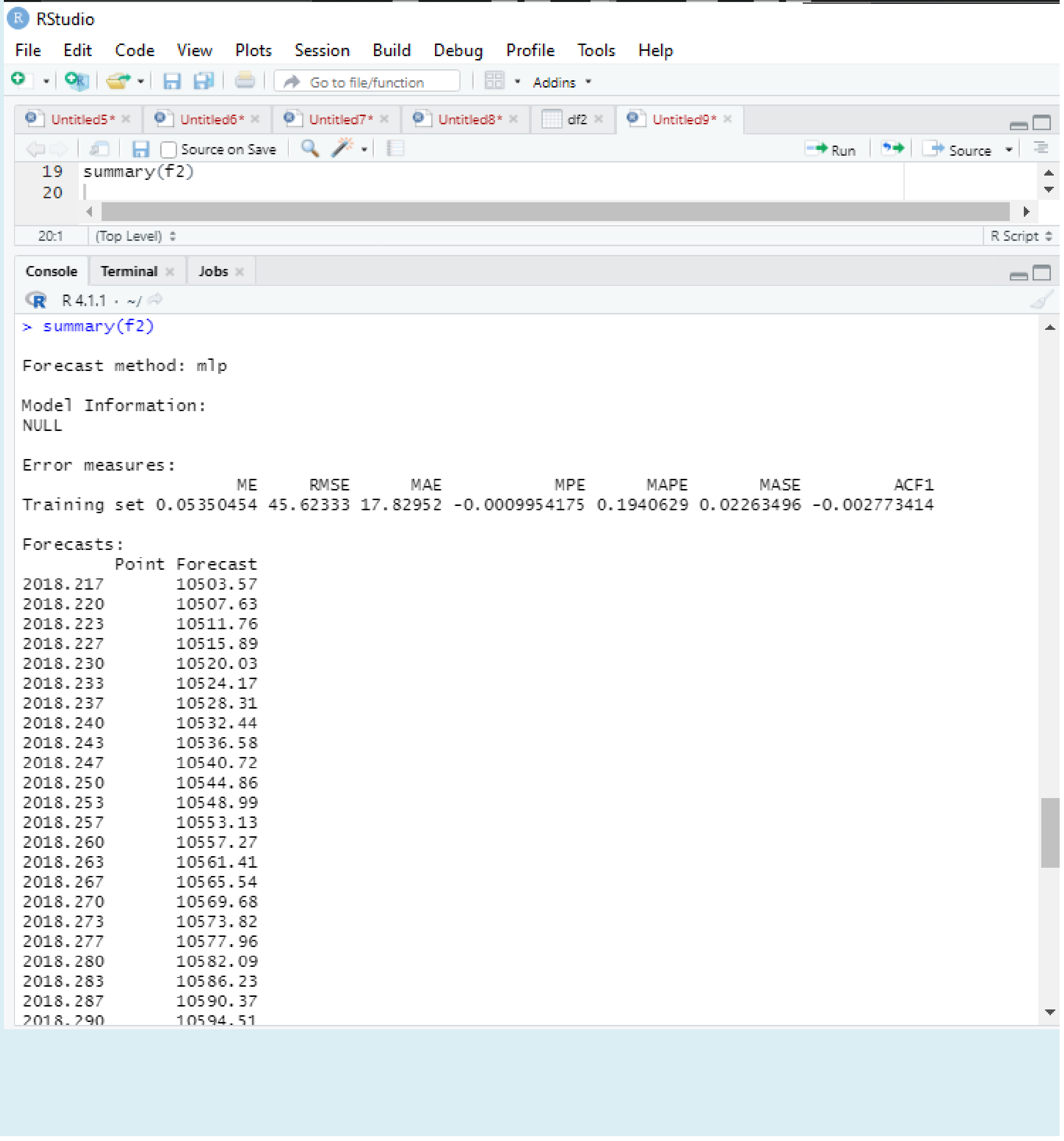
```
Warning messages:
1: package 'statsr' was built under R version 4.1.3
2: package 'BayesFactor' was built under R version 4.1.3
3: package 'coda' was built under R version 4.1.3
4: package 'Matrix' was built under R version 4.1.3
> library(RMySQL)
> library(dplyr)
> con = dbConnect(MySQL(), user = 'root', password = '', dbname = 'elkom2', host = 'localhost')
> dbListTables(con)
[1] "dim_date"      "dim_rice_type" "fact_price"
> myQuery <- "select * from fact_price;"
> df <- dbGetQuery(con, myQuery)
> df1<-filter(df,id_rice_type==10, id_date>=20170101,id_date<=20171231)
> df2<- df1[order(df1$id_date),]
> View(df2)
> tseries <- ts(df2$harga, start = c(2017, 1), frequency = 300)
> library(nnfor)
> library(forecast)
> #MLP
> fit<-mlp(tseries)
Warning message:
In preprocess(y, m, lags, keep, difforder, sel.lag, allow.det.season,  :
  No inputs left in the network after pre-selection, forcing AR(1).
> plot(fit)
> 
```

```
fit       List of  15
network   List of  14
```



```
df2<- df1[order(df1$id_date),]
View(df2)
tseries <- ts(df2$harga, start = c(2017, 1), frequency = 300)
library(nnfor)
library(forecast)
#MLP
fit<-mlp(tseries)
plot(fit)
f2=forecast(fit, h=90)
plot(f2)
summary(f2)
```

```
2: package ‘BayesFactor’ was built under R version 4.1.3
3: package ‘coda’ was built under R version 4.1.3
4: package ‘Matrix’ was built under R version 4.1.3
> library(RMySQL)
> library(dplyr)
> con = dbConnect(MySQL(), user = 'root', password = '', dbname = 'elkom2', host = 'localhost')
> dbListTables(con)
[1] "dim_date"      "dim_rice_type" "fact_price"
> myQuery <- "select * from fact_price;"
> df <- dbGetQuery(con, myQuery)
> df1<-filter(df,id_rice_type==10, id_date>=20170101,id_date<=20171231)
> df2<- df1[order(df1$id_date),]
> View(df2)
> tseries <- ts(df2$harga, start = c(2017, 1), frequency = 300)
> library(nnfor)
> library(forecast)
> #MLP
> fit<-mlp(tseries)
Warning message:
In preprocess(y, m, lags, keep, difforder, sel.lag, allow.det.season,  :
  No inputs left in the network after pre-selection, forcing AR(1).
> plot(fit)
> f2=forecast(fit, h=90)
> plot(f2)
> 
```



```
summary(f2)
```

```
> summary(f2)

Forecast method: mlp

Model Information:
NULL

Error measures:
                     ME     RMSE      MAE           MPE      MAPE       MASE         ACF1
Training set 0.05350454 45.62333 17.82952 -0.0009954175 0.1940629 0.02263496 -0.002773414

Forecasts:
         Point Forecast
2018.217       10503.57
2018.220       10507.63
2018.223       10511.76
2018.227       10515.89
2018.230       10520.03
2018.233       10524.17
2018.237       10528.31
2018.240       10532.44
2018.243       10536.58
2018.247       10540.72
2018.250       10544.86
2018.253       10548.99
2018.257       10553.13
2018.260       10557.27
2018.263       10561.41
2018.267       10565.54
2018.270       10569.68
2018.273       10573.82
2018.277       10577.96
2018.280       10582.09
2018.283       10586.23
2018.287       10590.37
2018.290       10594.51
```

Penjelasan :

Dari neural network di atas untuk melakukan forecasting sesuai dengan fact_harga, maka ditemukan error MAPE sebesar 0.1923416 dan MPE sebesar 0.0004696102.

Sumber :
https://hub.packtpub.com/training-and-visualizing-a-neural-network-with-r/
https://datascienceplus.com/neuralnet-train-and-test-neural-networks-using-r/

Elemen Kompetensi III

Perceptron Menggunakan Python

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read
_csv)
```

Definisikan step activation

```
def step_activation(z):
    if z>0:
        f_z = 1
    else:
        f_z = 0
    return f_z
```

Definisikan sign activation

```
def sign_activation(z):
    if z>=0:
        f_z = 1
    else:
        f_z = -1
    return f_z
```

Import librarty math dan definisikan sigmoid activation

```
import math

def sigmoid_activation(z):
    return 1 / (1 + math.exp(-z))
```

Varible data

```
data = np.array([[1,0,0,-
1],[1,0,1,1],[1,1,0,1],[1,1,1,1],[0,0,1,-1],[0,1,0,-
1],[0,1,1,1],[0,0,0,-1]])
m,n = data.shape
print(m,n)

X = data[:,:-1]
y = data[:,n-1]
print(X)
print(y)
```

Definisikan perceptron

```
def perceptron(X,y,lr, epochs):
    m,n = X.shape
    #inisiasi bobot wi
    w = np.zeros((n+1,1))
    str_w = ' '.join([str(elem) for elem in w])
    print(f'w ke 0:{str_w}')
    n_miss_list=[]
    for epoch in range(epochs):
        print(f'epoch: {epoch}')
        n_miss =0
        for idx, x_i in enumerate(X):
           # print(idx)
            #print(x_i)
            # tambahkan w0 (bias)pada posisi kolom ke 0
            x_i = np.insert(x_i,0,1).reshape(-1,1)
            #print(x_i)
            #hitung y_hat
           # y_hat = step_activation(np.dot(x_i.T,w))
            #y_hat = sign_activation(np.dot(x_i.T,w))
            y_hat = sigmoid_activation(np.dot(x_i.T,w))
            #update bobot jika
            delta = y[idx] - y_hat
            print(f'y:{y[idx]}')
            print(f'y_hat:{y_hat}')
            print(f'delta:{delta}')
            squeze = np.squeeze(delta)
            #print(f'sq:{squeze}')
            if squeze!=0:
                w += lr*((y[idx] - y_hat)*x_i)
                n_miss += 1
            str_w = ' '.join([str(elem) for elem in w])
            print(f'w ke {idx}:{str_w}')
        n_miss_list.append(n_miss)
    return w, n_miss_list
```

Hasil Prediksi

```
w, nmiss_list = perceptron(X,y,0.1,7)
```

Back propagation menggunakan python

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read
_csv)

from random import seed
from random import random
import math
```

# Initialize a network

```
def initialize_network(n_inputs, n_hidden, n_outputs):
  network = list()
  hidden_layer = [{'weights':[random() for i in range(n_inputs +
1)]} for i in range(n_hidden)]
  network.append(hidden_layer)
  output_layer = [{'weights':[random() for i in range(n_hidden +
1)]} for i in range(n_outputs)]
  network.append(output_layer)
  return network

seed(1)
network = initialize_network(2, 1, 2)
for layer in network:
  print(layer)
```

# Calculate neuron activation for an input

```
def activate(weights, inputs):
  activation = weights[-1]
  for i in range(len(weights)-1):
    activation += weights[i] * inputs[i]
  return activation
```

# Transfer neuron activation

```
def transfer(activation):
  return 1.0 / (1.0 + math.exp(-activation))
```

# Forward propagate input to a network output

```
def forward_propagate(network, row):
  inputs = row
  for layer in network:
    new_inputs = []
    for neuron in layer:
      activation = activate(neuron['weights'], inputs)
      neuron['output'] = transfer(activation)
      new_inputs.append(neuron['output'])
    inputs = new_inputs
  return inputs
```

# Data network

```
network = [[{'weights': [0.13436424411240122, 0.8474337369372327,
 0.763774618976614]}],\
           [{'weights': [0.2550690257394217, 0.49543508709194095]
}, {'weights': [0.4494910647887381, 0.651592972722763]}]]
row = [1, 0, None]
output = forward_propagate(network, row)
print(output)
```

# using the sigmoid transfer function, the derivative:
# derivative = output * (1.0 - output)
# Calculate the derivative of an neuron output

```
def transfer_derivative(output):
  return output * (1.0 - output)
```

# error = (expected - output) * transfer_derivative(output)
# Backpropagate error and store in neurons

```
def backward_propagate_error(network, expected):
  for i in reversed(range(len(network))):
    layer = network[i]
    errors = list()
    if i != len(network)-1:
      for j in range(len(layer)):
        error = 0.0
        for neuron in network[i + 1]:
          error += (neuron['weights'][j] * neuron['delta'])
        errors.append(error)
    else:
      for j in range(len(layer)):
        neuron = layer[j]
        errors.append(expected[j] - neuron['output'])
    for j in range(len(layer)):
      neuron = layer[j]
      neuron['delta'] = errors[j] * transfer_derivative(neuron['o
utput'])
```

# Calculate the derivative of an neuron output

```
def transfer_derivative(output):
  return output * (1.0 - output)
```

# Backpropagate error and store in neurons

```
def backward_propagate_error(network, expected):
  for i in reversed(range(len(network))):
    layer = network[i]
    errors = list()
    if i != len(network)-1:
      for j in range(len(layer)):
        error = 0.0
        for neuron in network[i + 1]:
          error += (neuron['weights'][j] * neuron['delta'])
        errors.append(error)
    else:
      for j in range(len(layer)):
        neuron = layer[j]
        errors.append(expected[j] - neuron['output'])
    for j in range(len(layer)):
      neuron = layer[j]
      neuron['delta'] = errors[j] * transfer_derivative(neuron['o
utput'])
```

# test backpropagation of error

```
network = [[{'output': 0.7105668883115941, 'weights': [0.13436424
411240122, 0.8474337369372327, 0.763774618976614]}],
    [{'output': 0.6213859615555266, 'weights': [0.255069025739421
7, 0.4954350870919409]}, {'output': 0.6573693455986976, 'weights
': [0.4494910647887381, 0.651592972722763]}]]
expected = [0, 1]
backward_propagate_error(network, expected)
for layer in network:
  print(layer)
```

#update bobot
# weight = weight + learning_rate * error * input
# Update network weights with error

```
def update_weights(network, row, l_rate):
  for i in range(len(network)):
    inputs = row[:-1]
    if i != 0:
      inputs = [neuron['output'] for neuron in network[i - 1]]
    for neuron in network[i]:
      for j in range(len(inputs)):
        neuron['weights'][j] += l_rate * neuron['delta'] * inputs
[j]
      neuron['weights'][-1] += l_rate * neuron['delta']
```

# Train a network for a fixed number of epochs

```
def train_network(network, train, l_rate, n_epoch, n_outputs):
  for epoch in range(n_epoch):
    sum_error = 0
    for row in train:
      outputs = forward_propagate(network, row)
      expected = [0 for i in range(n_outputs)]
      expected[row[-1]] = 1
      sum_error += sum([(expected[i]-
outputs[i])**2 for i in range(len(expected))])
      backward_propagate_error(network, expected)
      update_weights(network, row, l_rate)
    print('&gt;epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate
, sum_error))
```

# Test training backprop algorithm

```
seed(1)
dataset = [[2.7810836,2.550537003,0],
  [1.465489372,2.362125076,0],
  [3.396561688,4.400293529,0],
  [1.38807019,1.850220317,0],
  [3.06407232,3.005305973,0],
  [7.627531214,2.759262235,1],
  [5.332441248,2.088626775,1],
  [6.922596716,1.77106367,1],
  [8.675418651,-0.242068655,1],
  [7.673756466,3.508563011,1]]
n_inputs = len(dataset[0]) - 1
n_outputs = len(set([row[-1] for row in dataset]))
network = initialize_network(n_inputs, 2, n_outputs)
train_network(network, dataset, 0.5, 20, n_outputs)
for layer in network:
  print(layer)
```

# Make a prediction with a network

```
def predict(network, row):
  outputs = forward_propagate(network, row)
  return outputs.index(max(outputs))
```

# Test making predictions with the network

```
dataset = [[2.7810836,2.550537003,0],
  [1.465489372,2.362125076,0],
  [3.396561688,4.400293529,0],
  [1.38807019,1.850220317,0],
  [3.06407232,3.005305973,0],
  [7.627531214,2.759262235,1],
  [5.332441248,2.088626775,1],
  [6.922596716,1.77106367,1],
  [8.675418651,-0.242068655,1],
  [7.673756466,3.508563011,1]]
network = [[{'weights': [-
1.482313569067226, 1.8308790073202204, 1.078381922048799]}, {'wei
ghts': [0.23244990332399884, 0.3621998343835864, 0.40289821191094
327]}],
  [{'weights': [2.5001872433501404, 0.7887233511355132, -
1.1026649757805829]}, {'weights': [-
2.429350576245497, 0.8357651039198697, 1.0699217181280656]}]]
for row in dataset:
  prediction = predict(network, row)
  print('Expected=%d, Got=%d' % (row[-1], prediction))
```

```
8 4
[[1 0 0]
 [1 0 1]
 [1 1 0]
 [1 1 1]
 [0 0 1]
 [0 1 0]
 [0 1 1]
 [0 0 0]]
[-1  1  1  1 -1 -1  1 -1]
w ke 0:[0.] [0.] [0.] [0.]
epoch: 0
y:-1
y_hat:0.5
delta:-1.5
w ke 0:[-0.15] [-0.15] [0.] [0.]
y:1
y_hat:0.425557483188341
delta:0.5744425168116589
w ke 1:[-0.09255575] [-0.09255575] [0.] [0.05744425]
y:1
y_hat:0.4538538220813519
delta:0.5461461779186481
w ke 2:[-0.03794113] [-0.03794113] [0.05461462] [0.05744425]
y:1
y_hat:0.5090431658582721
delta:0.49095683414172786
w ke 3:[0.01115455] [0.01115455] [0.1037103] [0.10653994]
y:-1
y_hat:0.5293897043283174
delta:-1.5293897043283176
w ke 4:[-0.14178442] [0.01115455] [0.1037103] [-0.04639904]
y:-1
y_hat:0.49048262061715864
delta:-1.4904826206171586
w ke 5:[-0.29083268] [0.01115455] [-0.04533796] [-0.04639904]
y:1
y_hat:0.40550727231760336
```

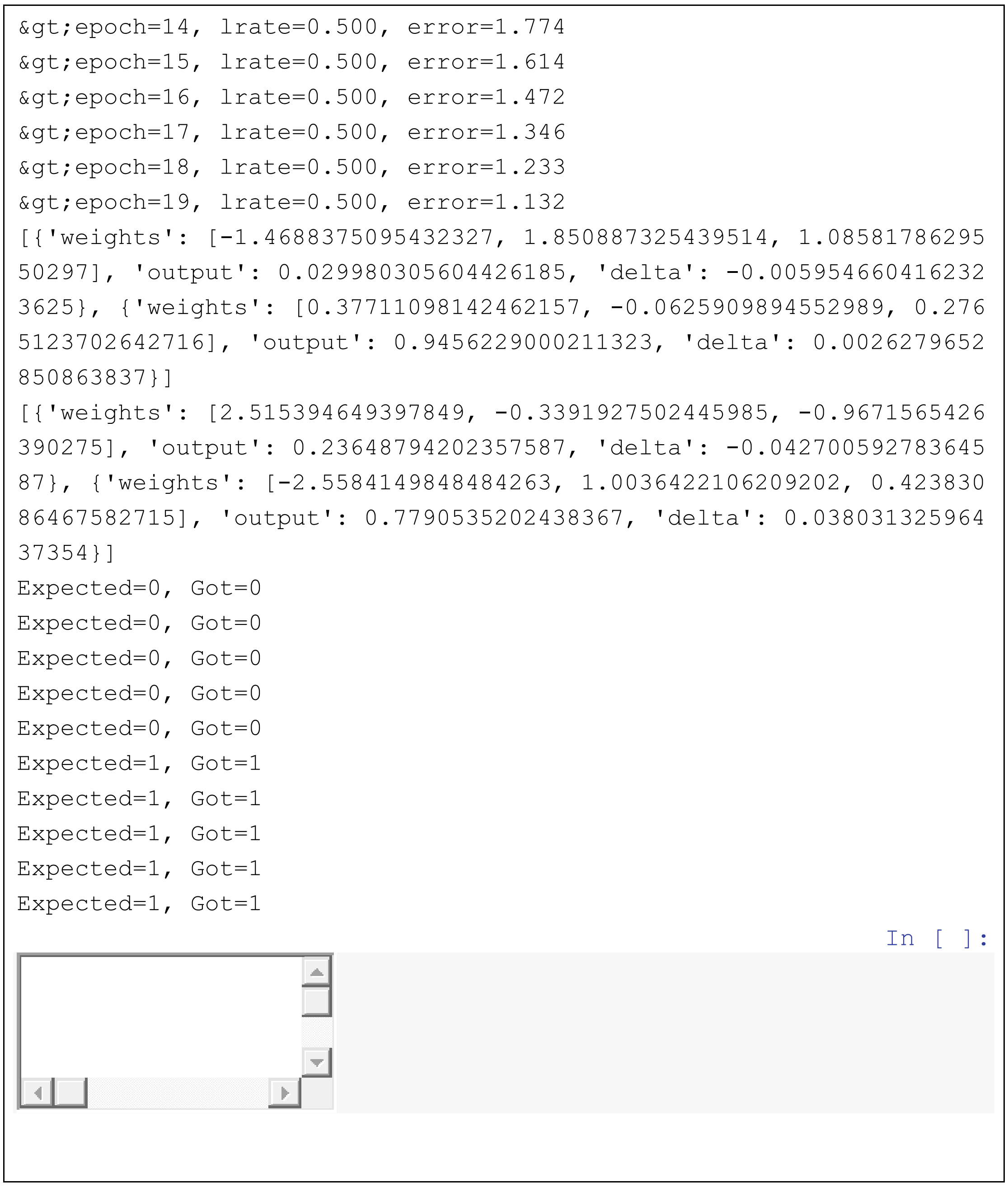
```
delta:0.5944927276823966
w ke 6:[-0.23138341] [0.01115455] [0.01411131] [0.01305024]
y:-1
y_hat:0.4424108546378813
delta:-1.4424108546378813
w ke 7:[-0.37562449] [0.01115455] [0.01411131] [0.01305024]
epoch: 1
y:-1
y_hat:0.40987795194523086
delta:-1.409877951945231
w ke 0:[-0.51661229] [-0.12983324] [0.01411131] [0.01305024]
y:1
y_hat:0.3467410647009
delta:0.6532589352991001
w ke 1:[-0.45128639] [-0.06450735] [0.01411131] [0.07837613]
y:1
y_hat:0.37714537284283317
delta:0.6228546271571669
w ke 2:[-0.38900093] [-0.00222189] [0.07639677] [0.07837613]
y:1
y_hat:0.44116139876251725
delta:0.5588386012374827
w ke 3:[-0.33311707] [0.05366197] [0.13228063] [0.13425999]
y:-1
y_hat:0.45044891040057744
delta:-1.4504489104005773
w ke 4:[-0.47816196] [0.05366197] [0.13228063] [-0.0107849]
y:-1
y_hat:0.41438154347920514
delta:-1.414381543479205
w ke 5:[-0.61960012] [0.05366197] [-0.00915752] [-0.0107849]
y:1
y_hat:0.3453499570602599
delta:0.6546500429397402
w ke 6:[-0.55413511] [0.05366197] [0.05630748] [0.0546801]
y:-1
y_hat:0.36490556455252204
delta:-1.364905564552522
w ke 7:[-0.69062567] [0.05366197] [0.05630748] [0.0546801]
epoch: 2
y:-1
y_hat:0.34593322250506525
delta:-1.3459332225050653
w ke 0:[-0.82521899] [-0.08093135] [0.05630748] [0.0546801]
y:1
y_hat:0.29912453320305715
```

```
delta:0.7008754667969428
w ke 1:[-0.75513144] [-0.0108438] [0.05630748] [0.12476765]
y:1
y_hat:0.3296722570077944
delta:0.6703277429922057
w ke 2:[-0.68809867] [0.05618897] [0.12334026] [0.12476765]
y:1
y_hat:0.4052102804658546
delta:0.5947897195341454
w ke 3:[-0.6286197] [0.11566794] [0.18281923] [0.18424662]
y:-1
y_hat:0.3906994477563182
delta:-1.3906994477563182
w ke 4:[-0.76768964] [0.11566794] [0.18281923] [0.04517668]
y:-1
y_hat:0.3578126836243525
delta:-1.3578126836243525
w ke 5:[-0.90347091] [0.11566794] [0.04703796] [0.04517668]
y:1
y_hat:0.3076228567293863
delta:0.6923771432706137
w ke 6:[-0.8342332] [0.11566794] [0.11627568] [0.11441439]
y:-1
y_hat:0.3027507276926821
delta:-1.3027507276926822
w ke 7:[-0.96450827] [0.11566794] [0.11627568] [0.11441439]
epoch: 3
y:-1
y_hat:0.299676182426241
delta:-1.299676182426241
w ke 0:[-1.09447589] [-0.01429967] [0.11627568] [0.11441439]
y:1
y_hat:0.27005152628919166
delta:0.7299484737108084
w ke 1:[-1.02148104] [0.05869517] [0.11627568] [0.18740924]
y:1
y_hat:0.3001654369062018
delta:0.6998345630937982
w ke 2:[-0.95149758] [0.12867863] [0.18625913] [0.18740924]
y:1
y_hat:0.3895627420367486
delta:0.6104372579632513
w ke 3:[-0.89045386] [0.18972236] [0.24730286] [0.24845297]
y:-1
y_hat:0.3447943750370855
delta:-1.3447943750370854
```

```
w ke 4:[-1.0249333] [0.18972236] [0.24730286] [0.11397353]
y:-1
y_hat:0.3148308057337168
delta:-1.3148308057337168
w ke 5:[-1.15641638] [0.18972236] [0.11581978] [0.11397353]
y:1
y_hat:0.28361032236936234
delta:0.7163896776306377
w ke 6:[-1.08477741] [0.18972236] [0.18745875] [0.1856125]
y:-1
y_hat:0.2526030015372409
delta:-1.252603001537241
w ke 7:[-1.21003771] [0.18972236] [0.18745875] [0.1856125]
epoch: 4
y:-1
y_hat:0.2649659781766723
delta:-1.2649659781766722
w ke 0:[-1.33653431] [0.06322576] [0.18745875] [0.1856125]
y:1
y_hat:0.2520523749499759
delta:0.7479476250500241
w ke 1:[-1.26173954] [0.13802052] [0.18745875] [0.26040726]
y:1
y_hat:0.28165636773799557
delta:0.7183436322620045
w ke 2:[-1.18990518] [0.20985488] [0.25929311] [0.26040726]
y:1
y_hat:0.386902814232141
delta:0.613097185767859
w ke 3:[-1.12859546] [0.2711646] [0.32060283] [0.32171698]
y:-1
y_hat:0.30855607014826897
delta:-1.308556070148269
w ke 4:[-1.25945107] [0.2711646] [0.32060283] [0.19086137]
y:-1
y_hat:0.2811330511681677
delta:-1.2811330511681678
w ke 5:[-1.38756437] [0.2711646] [0.19248952] [0.19086137]
y:1
y_hat:0.2681138075762698
delta:0.7318861924237302
w ke 6:[-1.31437576] [0.2711646] [0.26567814] [0.26404999]
y:-1
y_hat:0.2117555431458844
delta:-1.2117555431458844
w ke 7:[-1.43555131] [0.2711646] [0.26567814] [0.26404999]
```

```
epoch: 5
y:-1
y_hat:0.23787111162327032
delta:-1.2378711116232703
w ke 0:[-1.55933842] [0.14737749] [0.26567814] [0.26404999]
y:1
y_hat:0.24087086520780338
delta:0.7591291347921967
w ke 1:[-1.48342551] [0.2232904] [0.26567814] [0.3399629]
y:1
y_hat:0.27003264393280246
delta:0.7299673560671975
w ke 2:[-1.41042877] [0.29628714] [0.33867488] [0.3399629]
y:1
y_hat:0.3928128315454028
delta:0.6071871684545972
w ke 3:[-1.34971005] [0.35700586] [0.39939359] [0.40068162]
y:-1
y_hat:0.2790802534814989
delta:-1.279080253481499
w ke 4:[-1.47761808] [0.35700586] [0.39939359] [0.27277359]
y:-1
y_hat:0.25384216332342147
delta:-1.2538421633234216
w ke 5:[-1.6030023] [0.35700586] [0.27400938] [0.27277359]
y:1
y_hat:0.2580326094623316
delta:0.7419673905376685
w ke 6:[-1.52880556] [0.35700586] [0.34820612] [0.34697033]
y:-1
y_hat:0.17816851421687213
delta:-1.178168514216872
w ke 7:[-1.64662241] [0.35700586] [0.34820612] [0.34697033]
epoch: 6
y:-1
y_hat:0.2159177205199018
delta:-1.2159177205199019
w ke 0:[-1.76821418] [0.23541409] [0.34820612] [0.34697033]
y:1
y_hat:0.23400560895054215
delta:0.7659943910494579
w ke 1:[-1.69161474] [0.31201352] [0.34820612] [0.42356977]
y:1
y_hat:0.26281372431404415
delta:0.7371862756859558
w ke 2:[-1.61789611] [0.38573215] [0.42192474] [0.42356977]
```

```
y:1
y_hat:0.40451932045050526
delta:0.5954806795494947
w ke 3:[-1.55834805] [0.44528022] [0.48147281] [0.48311784]
y:-1
y_hat:0.254409716708183
delta:-1.254409716708183
w ke 4:[-1.68378902] [0.44528022] [0.48147281] [0.35767687]
y:-1
y_hat:0.23106343279385008
delta:-1.23106343279385
w ke 5:[-1.80689536] [0.44528022] [0.35836647] [0.35767687]
y:1
y_hat:0.25145787087398086
delta:0.7485421291260191
w ke 6:[-1.73204115] [0.44528022] [0.43322068] [0.43253108]
y:-1
y_hat:0.15032668067527968
delta:-1.1503266806752797
w ke 7:[-1.84707382] [0.44528022] [0.43322068] [0.43253108]
[{'weights': [0.13436424411240122, 0.8474337369372327, 0.7637746189
76614]}]
[{'weights': [0.2550690257394217, 0.49543508709194095]}, {'weights'
: [0.4494910647887381, 0.651592972722763]}]
[0.6629970129852887, 0.7253160725279748]
[{'output': 0.7105668883115941, 'weights': [0.13436424411240122, 0.
8474337369372327, 0.763774618976614], 'delta': -0.00053480480466105
17}]
[{'output': 0.6213859615555266, 'weights': [0.2550690257394217, 0.4
9543508709194095], 'delta': -0.14619064683582808}, {'output': 0.657
3693455986976, 'weights': [0.4494910647887381, 0.651592972722763],
'delta': 0.0771723774346327}]
&gt;epoch=0, lrate=0.500, error=6.350
&gt;epoch=1, lrate=0.500, error=5.531
&gt;epoch=2, lrate=0.500, error=5.221
&gt;epoch=3, lrate=0.500, error=4.951
&gt;epoch=4, lrate=0.500, error=4.519
&gt;epoch=5, lrate=0.500, error=4.173
&gt;epoch=6, lrate=0.500, error=3.835
&gt;epoch=7, lrate=0.500, error=3.506
&gt;epoch=8, lrate=0.500, error=3.192
&gt;epoch=9, lrate=0.500, error=2.898
&gt;epoch=10, lrate=0.500, error=2.626
&gt;epoch=11, lrate=0.500, error=2.377
&gt;epoch=12, lrate=0.500, error=2.153
&gt;epoch=13, lrate=0.500, error=1.953
```



```
&gt;epoch=14, lrate=0.500, error=1.774
&gt;epoch=15, lrate=0.500, error=1.614
&gt;epoch=16, lrate=0.500, error=1.472
&gt;epoch=17, lrate=0.500, error=1.346
&gt;epoch=18, lrate=0.500, error=1.233
&gt;epoch=19, lrate=0.500, error=1.132
[{'weights': [-1.4688375095432327, 1.850887325439514, 1.08581786295
50297], 'output': 0.029980305604426185, 'delta': -0.005954660416232
3625}, {'weights': [0.37711098142462157, -0.0625909894552989, 0.276
5123702642716], 'output': 0.9456229000211323, 'delta': 0.0026279652
850863837}]
[{'weights': [2.515394649397849, -0.3391927502445985, -0.9671565426
390275], 'output': 0.23648794202357587, 'delta': -0.042700592783645
87}, {'weights': [-2.5584149848484263, 1.0036422106209202, 0.423830
86467582715], 'output': 0.7790535202438367, 'delta': 0.038031325964
37354}]
Expected=0, Got=0
Expected=0, Got=0
Expected=0, Got=0
Expected=0, Got=0
Expected=0, Got=0
Expected=1, Got=1
Expected=1, Got=1
Expected=1, Got=1
Expected=1, Got=1
Expected=1, Got=1
                                                          In [ ]:
```

CEK LIST

Elemen Kompetensi	No Latihan	Penyelesaian	
		Selesai	Tidak selesai
1	1.1.1	✓	
2	1.2.1	✓	

FORM UMPAN BALIK

Elemen Kompetensi	**Tingkat Kesulitan**	**Tingkat Ketertarikan**	**Waktu Penyelesaian dalam menit**
Memahami proses backpropagation dengan neuralnet library di R	☐ Sangat Mudah ☐ Mudah ☐ ✓ Biasa ☐ Sulit ☐ Sangat Sulit	☐ Tidak Tertarik ☐ Cukup Tertarik ☐ Tertarik ☐ ✓ Sangat Tertarik	
Menerapkan Neural Network untuk melakukan Forecasting.	☐ Sangat Mudah ☐ Mudah ☐ ✓ Biasa ☐ Sulit ☐ Sangat Sulit	☐ Tidak Tertarik ☐ Cukup Tertarik ☐ Tertarik ☐ ✓ Sangat Tertarik	