Lampiran 1

UNIVERSITAS MUHAMMADIYAH PONOROGO
FAKULTAS TEKNIK
Jl. Budi Utomo No. 10 Ponorogo 63471 Jawa Timur Indonesia
Telp (0352) 481124, Fax. (0352) 461796, e-mail : akadc@fk.umpo.ac.id
Website : www.umpo.ac.id

Nomor : 063/III-5/KM/2017
Lamp : -
Hal : *Permohonan ijin penelitian*

Yth. **Pimpinan Home Industri Brem (Ariska)**
Ds. Kaliabu, Kec. Mejayan Kab. Madiun
Di
tempat

Assalamu'alaikum Wr. Wb

Bersama ini, kami mengajukan permohonan ijin Penelitian di **Home Industri Brem " Ariska "** guna mendukung penyusunan Skripsi/ Tugas Akhir pada Semester Ganjil Tahun Akademik 2016/ 2017, dari mahasiswa Teknik Informatika Fakultas Teknik Universitas Muhammadiyah Ponorogo.

Adapun nama penelitian tersebut adalah sebagai berikut :

Nama : Deviardia Putri Nurmayasari
NIM : 13531848
Judul Skripsi : Implementasi Jaringan Syaraf Tiruan pada Home Industri Brem untuk mencegah keterlambatan Distibusi Pengiriman Produk.

Demikian atas perkenan dan kerja samanya kami ucapkan terima kasih.

Wassalamu'alaikum Wr. Wb.

Ponorogo, 04 Februari 2017
Dekan,

Ir. Alivadi, MM, M.Kom
NIK. 19640301 199009 12

Lampiran 2

Data Penjualan Produk pada Home Industi Brem (pcs)

No	Tahun	Jan	Feb	Mar	Apr	Mei	Jun	Jul	Ags	Sep	Okt	Nov	Des
1	**2014**	2400	1500	1200	1350	1200	3000	3600	1650	1350	2250	2100	2550
2	**2015**	2550	1650	1350	1800	1500	4500	3000	1200	1200	2100	1950	2400
3	**2016**	2550	1800	1050	1200	1800	4650	2850	1350	1500	2250	1650	2400

Catatan :

- 1 Kwintal Ketan dalam 1x produksi dapat menghasilkan kurang lebih 500pcs
- Dalam 1x produksi membutuhkan kurang lebih 10hari sampai proses pemasaran tergantung cuaca
- Dalam 1 Bulan Home Industri Brem dapat melakukan 3-5 kali proses produksi

Home Industri Brem Desa Kaliabu, Kecamatan Mejayan, Kabupaten Madiun

Lampiran 3

Coding Program Pelatihan Jaringan :

```
% clc;clear;close all;warning off;

% Proses membaca data latih dari excel
filename = 'Book1.xlsx'
sheet = 2
xlRange = 'D6:P17'

% data input dan target
Data = xlsread(filename, sheet, xlRange)
data_latih = Data(:,1:12)'
target_latih = Data(:,13)'
[m,n] = size(data_latih)

% Membangun Jaringan Syaraf Tiruan
net = newff(minmax(data_latih),[12
1],{'logsig','purelin'},'traingdx')

%melihat bobot-bobot awal input, lapisan, dan bias
BobotAwal_Input        = net.IW{1,1}
BobotAwal_Bias_Input   = net.b{1,1}
BobotAwal_Lapisan      = net.LW{2,1}
BobotAwal_Bias_Lapisan = net.b{2,1}

%set max epoh, goal,learning rate, momentum, show step
net.trainParam.epoch = 50
net.trainParam.goal  = 1e-3
net.trainParam.lr    = 0.1
net.trainParam.mc    = 0.3
net.trainParam.show  = 10

% Memberikan nilai untuk mempengaruhi proses pelatihan
net.performFcn = 'mse'
net.trainParam.goal = 0.001
net.trainParam.show = 20
net.trainParam.epochs = 1000
net.trainParam.mc = 0.95
net.trainParam.lr = 0.1

% Proses training
[net_keluaran,tr,Y,E] = train(net,data_latih,target_latih)

% Hasil setelah pelatihan
bobot_hidden = net_keluaran.IW{1,1}
bobot_keluaran = net_keluaran.LW{2,1}
bias_hidden = net_keluaran.b{1,1}
bias_keluaran = net_keluaran.b{2,1}
jumlah_iterasi = tr.num_epochs
nilai_keluaran = Y
```

```
nilai_error = E
error_MSE = (1/n)*sum(nilai_error.^2)

save net.mat net_keluaran

% Hasil prediksi
hasil_latih = sim(net_keluaran,data_latih)
max_data = 4650
min_data = 1050
hasil_latih = ((hasil_latih-0.1)*(max_data-min_data)/0.8)+min_data

% Performansi hasil prediksi
filename = 'Book1.xlsx'
sheet = 1
xlRange = 'E7:P7'

target_latih_asli = xlsread(filename, sheet, xlRange)

figure,
plotregression(target_latih_asli,hasil_latih,'Regression')

figure,
plotperform(tr)

figure,
plot(hasil_latih,'bo-')
hold on
plot(target_latih_asli,'ro-')
hold off
grid on
title(strcat(['Grafik Keluaran JST vs Target dengan nilai MSE =
',...
    num2str(error_MSE)]))
xlabel('Pola ke-')
ylabel('Permintaan Pasar Produk Brem')
legend('Keluaran JST','Target','Location','Best')
```

Hasil Proses Penghitungan Dari Program Pelatihan Jaringan :

filename =

Book1.xlsx

sheet =

2

xlRange =

D6:P17

Data =

0.4000	0.2000	0.1333	0.1667	0.1333	0.5333	0.6667	0.2333	0.1667	0.3667	0.3333	0.4333	0.4333
0.2000	0.1333	0.1667	0.1333	0.5333	0.6667	0.2333	0.1667	0.3667	0.3333	0.4333	0.4333	0.2333
0.1333	0.1667	0.1333	0.5333	0.6667	0.2333	0.1667	0.3667	0.3333	0.4333	0.4333	0.2333	0.1667
0.1667	0.1333	0.5333	0.6667	0.2333	0.1667	0.3667	0.3333	0.4333	0.4333	0.2333	0.1667	0.2667
0.1333	0.5333	0.6667	0.2333	0.1667	0.3667	0.3333	0.4333	0.4333	0.2333	0.1667	0.2667	0.2000
0.5333	0.6667	0.2333	0.1667	0.3667	0.3333	0.4333	0.4333	0.2333	0.1667	0.2667	0.2000	0.8667
0.6667	0.2333	0.1667	0.3667	0.3333	0.4333	0.4333	0.2333	0.1667	0.2667	0.2000	0.8667	0.5333
0.2333	0.1667	0.3667	0.3333	0.4333	0.4333	0.2333	0.1667	0.2667	0.2000	0.8667	0.5333	0.1333
0.1667	0.3667	0.3333	0.4333	0.4333	0.2333	0.1667	0.2667	0.2000	0.8667	0.5333	0.1333	0.1333
0.3667	0.3333	0.4333	0.4333	0.2333	0.1667	0.2667	0.2000	0.8667	0.5333	0.1333	0.1333	0.3333

0.3333	0.4333	0.4333	0.2333	0.1667	0.2667	0.2000	0.8667	0.5333	0.1333	0.1333	0.3333	0.3000
0.4333	0.4333	0.2333	0.1667	0.2667	0.2000	0.8667	0.5333	0.1333	0.1333	0.3333	0.3000	0.4000

data_latih =

0.4000	0.2000	0.1333	0.1667	0.1333	0.5333	0.6667	0.2333	0.1667	0.3667	0.3333	0.4333
0.2000	0.1333	0.1667	0.1333	0.5333	0.6667	0.2333	0.1667	0.3667	0.3333	0.4333	0.4333
0.1333	0.1667	0.1333	0.5333	0.6667	0.2333	0.1667	0.3667	0.3333	0.4333	0.4333	0.2333
0.1667	0.1333	0.5333	0.6667	0.2333	0.1667	0.3667	0.3333	0.4333	0.4333	0.2333	0.1667
0.1333	0.5333	0.6667	0.2333	0.1667	0.3667	0.3333	0.4333	0.4333	0.2333	0.1667	0.2667
0.5333	0.6667	0.2333	0.1667	0.3667	0.3333	0.4333	0.4333	0.2333	0.1667	0.2667	0.2000
0.6667	0.2333	0.1667	0.3667	0.3333	0.4333	0.4333	0.2333	0.1667	0.2667	0.2000	0.8667
0.2333	0.1667	0.3667	0.3333	0.4333	0.4333	0.2333	0.1667	0.2667	0.2000	0.8667	0.5333
0.1667	0.3667	0.3333	0.4333	0.4333	0.2333	0.1667	0.2667	0.2000	0.8667	0.5333	0.1333
0.3667	0.3333	0.4333	0.4333	0.2333	0.1667	0.2667	0.2000	0.8667	0.5333	0.1333	0.1333
0.3333	0.4333	0.4333	0.2333	0.1667	0.2667	0.2000	0.8667	0.5333	0.1333	0.1333	0.3333
0.4333	0.4333	0.2333	0.1667	0.2667	0.2000	0.8667	0.5333	0.1333	0.1333	0.3333	0.3000

target_latih =

0.4333	0.2333	0.1667	0.2667	0.2000	0.8667	0.5333	0.1333	0.1333	0.3333	0.3000	0.4000

```
m =
    12
n =
    12

net =
    Neural Network object:
    architecture:
        numInputs: 1
        numLayers: 2
      biasConnect: [1; 1]
     inputConnect: [1; 0]
     layerConnect: [0 0; 1 0]
    outputConnect: [0 1]

       numOutputs: 1  (read-only)
    numInputDelays: 0  (read-only)
    numLayerDelays: 0  (read-only)

    subobject structures:
            inputs: {1x1 cell} of inputs
            layers: {2x1 cell} of layers
           outputs: {1x2 cell} containing 1 output
            biases: {2x1 cell} containing 2 biases
      inputWeights: {2x1 cell} containing 1 input weight
      layerWeights: {2x2 cell} containing 1 layer weight

    functions:
          adaptFcn: 'trains'
         divideFcn: (none)
       gradientFcn: 'calcgrad'
           initFcn: 'initlay'
```

performFcn: 'mse'
plotFcns: {'plotperform','plottrainstate','plotregression'}
trainFcn: 'traingdx'

parameters:
adaptParam: .passes
divideParam: (none)
gradientParam: (none)
initParam: (none)
performParam: (none)
trainParam: .show, .showWindow, .showCommandLine, .epochs,
.time, .goal, .max_fail, .lr,
.lr_inc, .lr_dec, .max_perf_inc, .mc,
.min_grad

weight and bias values:
IW: {2x1 cell} containing 1 input weight matrix
LW: {2x2 cell} containing 1 layer weight matrix
b: {2x1 cell} containing 2 bias vectors

other:
name: ''
userdata: (user information)

BobotAwal_Input =

3.9590	5.7508	2.2483	2.4508	2.6336	3.3715	3.0122	3.9984	0.6296	1.7310	-3.5985	-2.9106
5.7428	-0.2070	3.6476	-2.5884	3.6044	-3.6970	-2.7655	-2.3092	-0.3150	2.5541	4.7541	-2.4311
-4.0667	3.2738	2.6507	4.9085	-2.4419	0.0693	3.5657	2.1365	-3.8701	-0.3922	-3.9278	-2.8105
5.1044	-4.4221	-1.3308	-5.7488	2.2190	2.6221	-1.4114	2.3871	-1.4627	-3.7375	2.4688	-3.2683
1.9617	-1.1596	2.3044	-0.9079	2.2987	6.1799	-3.4262	-1.3500	-3.6414	-2.9214	3.4202	3.9806
-4.6530	4.8065	-3.8015	-1.3693	-3.9007	5.6640	-2.1926	0.5974	2.4744	3.4754	3.1001	0.6702
-2.9830	3.9351	2.7748	3.5757	-5.1309	0.6782	1.1907	-4.3520	-1.8490	-3.4047	-4.0701	0.4883
0.5792	5.6766	-5.7838	3.6469	-0.0202	-4.7621	-0.2514	-4.1985	0.2564	2.9274	-0.9004	-3.1900
5.8104	1.9779	-2.8331	-3.9768	5.8388	-4.7510	-1.4354	0.2980	-3.0882	0.3541	-2.2180	3.2608
5.3262	-5.3193	-5.1994	-0.1173	-1.8287	-2.9634	2.8878	2.4369	0.8497	4.1339	2.5002	1.0170
-4.4571	4.5448	-5.2444	-0.7083	1.1100	4.7310	0.8457	4.3046	-2.2440	-3.9936	-0.6493	-1.4111
6.0546	5.5837	4.1616	1.8824	-3.5534	-3.3721	0.4874	-3.6279	1.4417	-0.5364	3.8424	0.1240

BobotAwal_Bias_Input =

-13.2139
-5.0164
2.9873
1.5089

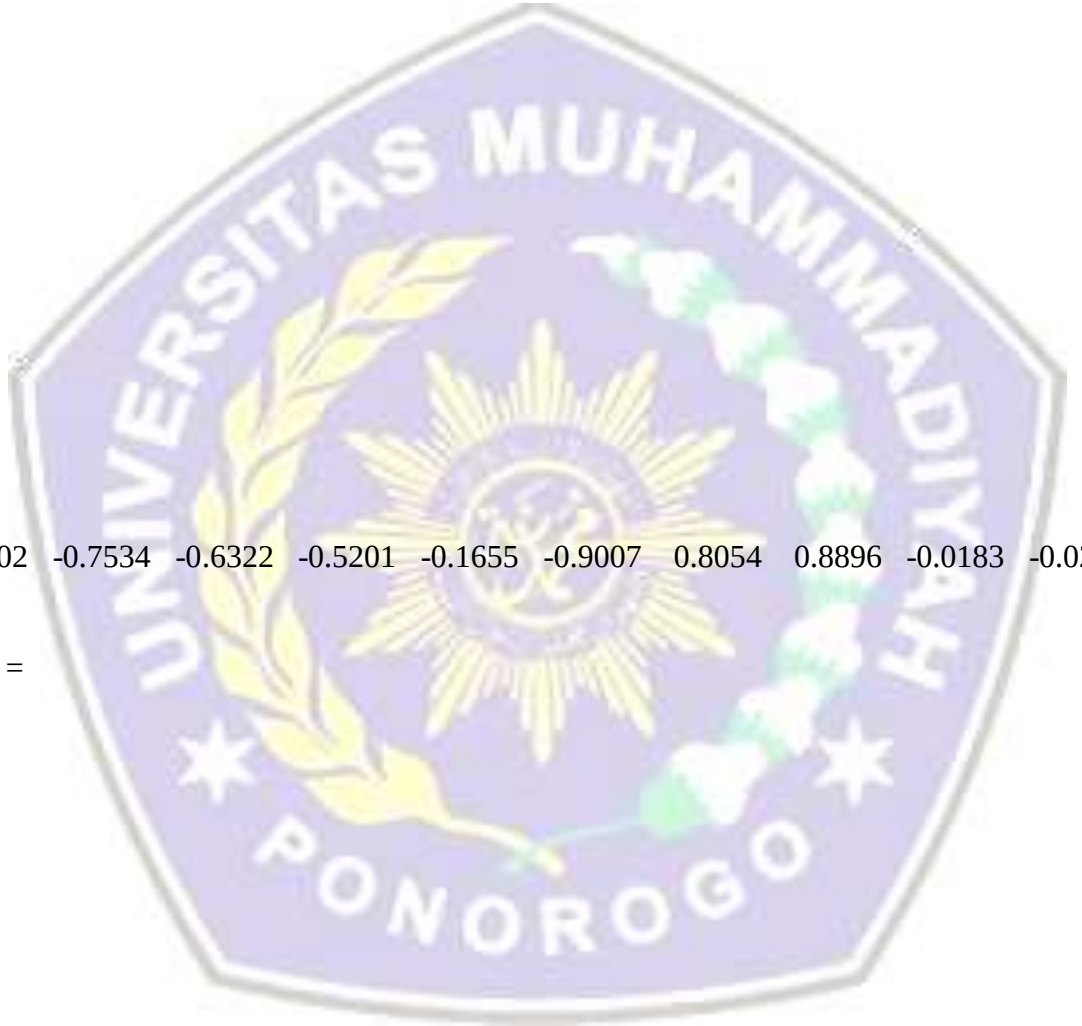
-3.2646
-2.5156
4.5866
4.0365
2.2515
-0.7196
-1.3992
-1.6156

BobotAwal_Lapisan =

-0.1964 -0.8481 -0.5202 -0.7534 -0.6322 -0.5201 -0.1655 -0.9007 0.8054 0.8896 -0.0183 -0.0215

BobotAwal_Bias_Lapisan =

-0.3246



```
net =
    Neural Network object:
    architecture:
        numInputs: 1
        numLayers: 2
      biasConnect: [1; 1]
     inputConnect: [1; 0]
     layerConnect: [0 0; 1 0]
    outputConnect: [0 1]

       numOutputs: 1  (read-only)
    numInputDelays: 0  (read-only)
    numLayerDelays: 0  (read-only)

    subobject structures:
           inputs: {1x1 cell} of inputs
           layers: {2x1 cell} of layers
          outputs: {1x2 cell} containing 1 output
           biases: {2x1 cell} containing 2 biases
     inputWeights: {2x1 cell} containing 1 input weight
     layerWeights: {2x2 cell} containing 1 layer weight

    functions:
         adaptFcn: 'trains'
        divideFcn: (none)
      gradientFcn: 'calcgrad'
          initFcn: 'initlay'
       performFcn: 'mse'
         plotFcns: {'plotperform','plottrainstate','plotregression'}
         trainFcn: 'traingdx'

    parameters:
```

```
        adaptParam: .passes
       divideParam: (none)
     gradientParam: (none)
         initParam: (none)
      performParam: (none)
        trainParam: .show, .showWindow, .showCommandLine, .epochs,
                    .time, .goal, .max_fail, .lr,
                    .lr_inc, .lr_dec, .max_perf_inc, .mc,
                    .min_grad, .epoch

    weight and bias values:
                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

    other:
                name: ''
            userdata: (user information)


net =
    Neural Network object:
    architecture:
            numInputs: 1
            numLayers: 2
       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

        numOutputs: 1  (read-only)
    numInputDelays: 0  (read-only)
    numLayerDelays: 0  (read-only)
```

subobject structures:

inputs: {1x1 cell} of inputs

layers: {2x1 cell} of layers

outputs: {1x2 cell} containing 1 output

biases: {2x1 cell} containing 2 biases

inputWeights: {2x1 cell} containing 1 input weight

layerWeights: {2x2 cell} containing 1 layer weight

functions:

adaptFcn: 'trains'

divideFcn: (none)

gradientFcn: 'calcgrad'

initFcn: 'initlay'

performFcn: 'mse'

plotFcns: {'plotperform','plottrainstate','plotregression'}

trainFcn: 'traingdx'

parameters:

adaptParam: .passes

divideParam: (none)

gradientParam: (none)

initParam: (none)

performParam: (none)

trainParam: .show, .showWindow, .showCommandLine, .epochs,
.time, .goal, .max_fail, .lr,
.lr_inc, .lr_dec, .max_perf_inc, .mc,
.min_grad, .epoch

weight and bias values:

IW: {2x1 cell} containing 1 input weight matrix

LW: {2x2 cell} containing 1 layer weight matrix

b: {2x1 cell} containing 2 bias vectors

other:

name: ''

userdata: (user information)

net =

Neural Network object:

architecture:

numInputs: 1

numLayers: 2

biasConnect: [1; 1]

inputConnect: [1; 0]

layerConnect: [0 0; 1 0]

outputConnect: [0 1]

numOutputs: 1 (read-only)

numInputDelays: 0 (read-only)

numLayerDelays: 0 (read-only)

subobject structures:

inputs: {1x1 cell} of inputs

layers: {2x1 cell} of layers

outputs: {1x2 cell} containing 1 output

biases: {2x1 cell} containing 2 biases

inputWeights: {2x1 cell} containing 1 input weight

layerWeights: {2x2 cell} containing 1 layer weight

functions:

adaptFcn: 'trains'

divideFcn: (none)

gradientFcn: 'calcgrad'

```
        initFcn: 'initlay'
      performFcn: 'mse'
       plotFcns: {'plotperform','plottrainstate','plotregression'}
       trainFcn: 'traingdx'

  parameters:
     adaptParam: .passes
    divideParam: (none)
  gradientParam: (none)
     initParam: (none)
   performParam: (none)
     trainParam: .show, .showWindow, .showCommandLine, .epochs,
                 .time, .goal, .max_fail, .lr,
                 .lr_inc, .lr_dec, .max_perf_inc, .mc,
                 .min_grad, .epoch, .Ir

  weight and bias values:
              IW: {2x1 cell} containing 1 input weight matrix
              LW: {2x2 cell} containing 1 layer weight matrix
               b: {2x1 cell} containing 2 bias vectors

  other:
            name: ''
        userdata: (user information)

net =
  Neural Network object:
  architecture:
      numInputs: 1
      numLayers: 2
    biasConnect: [1; 1]
   inputConnect: [1; 0]
```

layerConnect: [0 0; 1 0]
outputConnect: [0 1]

numOutputs: 1 (read-only)
numInputDelays: 0 (read-only)
numLayerDelays: 0 (read-only)

subobject structures:

inputs: {1x1 cell} of inputs
layers: {2x1 cell} of layers
outputs: {1x2 cell} containing 1 output
biases: {2x1 cell} containing 2 biases
inputWeights: {2x1 cell} containing 1 input weight
layerWeights: {2x2 cell} containing 1 layer weight

functions:
adaptFcn: 'trains'
divideFcn: (none)
gradientFcn: 'calcgrad'
initFcn: 'initlay'
performFcn: 'mse'
plotFcns: {'plotperform','plottrainstate','plotregression'}
trainFcn: 'traingdx'

parameters:
adaptParam: .passes
divideParam: (none)
gradientParam: (none)
initParam: (none)
performParam: (none)
trainParam: .show, .showWindow, .showCommandLine, .epochs,

```
                .time, .goal, .max_fail, .lr,
                .lr_inc, .lr_dec, .max_perf_inc, .mc,
                .min_grad, .epoch, .Ir

    weight and bias values:
            IW: {2x1 cell} containing 1 input weight matrix
            LW: {2x2 cell} containing 1 layer weight matrix
             b: {2x1 cell} containing 2 bias vectors

    other:
           name: "
        userdata: (user information)

net =
    Neural Network object:
    architecture:
        numInputs: 1
        numLayers: 2
      biasConnect: [1; 1]
     inputConnect: [1; 0]
     layerConnect: [0 0; 1 0]
    outputConnect: [0 1]

       numOutputs: 1  (read-only)
    numInputDelays: 0  (read-only)
    numLayerDelays: 0  (read-only)

    subobject structures:
            inputs: {1x1 cell} of inputs
            layers: {2x1 cell} of layers
           outputs: {1x2 cell} containing 1 output
            biases: {2x1 cell} containing 2 biases
```

inputWeights: {2x1 cell} containing 1 input weight
layerWeights: {2x2 cell} containing 1 layer weight

functions:
adaptFcn: 'trains'
divideFcn: (none)
gradientFcn: 'calcgrad'
initFcn: 'initlay'
performFcn: 'mse'
plotFcns: {'plotperform','plottrainstate','plotregression'}
trainFcn: 'traingdx'

parameters:
adaptParam: .passes
divideParam: (none)
gradientParam: (none)
initParam: (none)
performParam: (none)
trainParam: .show, .showWindow, .showCommandLine, .epochs,
.time, .goal, .max_fail, .lr,
.lr_inc, .lr_dec, .max_perf_inc, .mc,
.min_grad, .epoch, .Ir

weight and bias values:
IW: {2x1 cell} containing 1 input weight matrix
LW: {2x2 cell} containing 1 layer weight matrix
b: {2x1 cell} containing 2 bias vectors

other:
name: "
userdata: (user information)

```
net =
    Neural Network object:
    architecture:
        numInputs: 1
        numLayers: 2
      biasConnect: [1; 1]
     inputConnect: [1; 0]
     layerConnect: [0 0; 1 0]
    outputConnect: [0 1]

       numOutputs: 1  (read-only)
    numInputDelays: 0  (read-only)
    numLayerDelays: 0  (read-only)

    subobject structures:
           inputs: {1x1 cell} of inputs
           layers: {2x1 cell} of layers
          outputs: {1x2 cell} containing 1 output
           biases: {2x1 cell} containing 2 biases
     inputWeights: {2x1 cell} containing 1 input weight
     layerWeights: {2x2 cell} containing 1 layer weight

    functions:
         adaptFcn: 'trains'
        divideFcn: (none)
      gradientFcn: 'calcgrad'
          initFcn: 'initlay'
       performFcn: 'mse'
         plotFcns: {'plotperform','plottrainstate','plotregression'}
         trainFcn: 'traingdx'

    parameters:
```

```
    adaptParam: .passes
   divideParam: (none)
 gradientParam: (none)
    initParam: (none)
  performParam: (none)
    trainParam: .show, .showWindow, .showCommandLine, .epochs,
                .time, .goal, .max_fail, .lr,
                .lr_inc, .lr_dec, .max_perf_inc, .mc,
                .min_grad, .epoch, .Ir

  weight and bias values:
              IW: {2x1 cell} containing 1 input weight matrix
              LW: {2x2 cell} containing 1 layer weight matrix
               b: {2x1 cell} containing 2 bias vectors

  other:
            name: ''
        userdata: (user information)

net =
  Neural Network object:
  architecture:

     numInputs: 1
     numLayers: 2
   biasConnect: [1; 1]
  inputConnect: [1; 0]
  layerConnect: [0 0; 1 0]
 outputConnect: [0 1]

    numOutputs: 1  (read-only)
  numInputDelays: 0  (read-only)
```

numLayerDelays: 0 (read-only)

subobject structures:
        inputs: {1x1 cell} of inputs
        layers: {2x1 cell} of layers
       outputs: {1x2 cell} containing 1 output
        biases: {2x1 cell} containing 2 biases
  inputWeights: {2x1 cell} containing 1 input weight
  layerWeights: {2x2 cell} containing 1 layer weight

functions:
      adaptFcn: 'trains'
     divideFcn: (none)
   gradientFcn: 'calcgrad'
       initFcn: 'initlay'
    performFcn: 'mse'
      plotFcns: {'plotperform','plottrainstate','plotregression'}
      trainFcn: 'traingdx'

parameters:
     adaptParam: .passes
    divideParam: (none)
  gradientParam: (none)
      initParam: (none)
   performParam: (none)
     trainParam: .show, .showWindow, .showCommandLine, .epochs,
                 .time, .goal, .max_fail, .lr,
                 .lr_inc, .lr_dec, .max_perf_inc, .mc,
                 .min_grad, .epoch, .Ir

weight and bias values:
            IW: {2x1 cell} containing 1 input weight matrix

```
            LW: {2x2 cell} containing 1 layer weight matrix
             b: {2x1 cell} containing 2 bias vectors

    other:
            name: ''
        userdata: (user information)

net =
    Neural Network object:
    architecture:
        numInputs: 1
        numLayers: 2
      biasConnect: [1; 1]
     inputConnect: [1; 0]
     layerConnect: [0 0; 1 0]
    outputConnect: [0 1]

      numOutputs: 1  (read-only)
    numInputDelays: 0  (read-only)
    numLayerDelays: 0  (read-only)

    subobject structures:
           inputs: {1x1 cell} of inputs
           layers: {2x1 cell} of layers
          outputs: {1x2 cell} containing 1 output
           biases: {2x1 cell} containing 2 biases
     inputWeights: {2x1 cell} containing 1 input weight
     layerWeights: {2x2 cell} containing 1 layer weight

    functions:
        adaptFcn: 'trains'
       divideFcn: (none)
```

```
    gradientFcn: 'calcgrad'
        initFcn: 'initlay'
    performFcn: 'mse'
      plotFcns: {'plotperform','plottrainstate','plotregression'}
      trainFcn: 'traingdx'

  parameters:
     adaptParam: .passes
    divideParam: (none)
  gradientParam: (none)
      initParam: (none)
   performParam: (none)
     trainParam: .show, .showWindow, .showCommandLine, .epochs,
                 .time, .goal, .max_fail, .lr,
                 .lr_inc, .lr_dec, .max_perf_inc, .mc,
                 .min_grad, .epoch, .Ir

  weight and bias values:
              IW: {2x1 cell} containing 1 input weight matrix
              LW: {2x2 cell} containing 1 layer weight matrix
               b: {2x1 cell} containing 2 bias vectors

  other:
            name: ''
        userdata: (user information)

net =
  Neural Network object:
  architecture:
      numInputs: 1
      numLayers: 2
    biasConnect: [1; 1]
```

```
  inputConnect: [1; 0]
  layerConnect: [0 0; 1 0]
 outputConnect: [0 1]

    numOutputs: 1  (read-only)
numInputDelays: 0  (read-only)
numLayerDelays: 0  (read-only)

subobject structures:
        inputs: {1x1 cell} of inputs
        layers: {2x1 cell} of layers
       outputs: {1x2 cell} containing 1 output
        biases: {2x1 cell} containing 2 biases
  inputWeights: {2x1 cell} containing 1 input weight
  layerWeights: {2x2 cell} containing 1 layer weight

functions:
      adaptFcn: 'trains'
     divideFcn: (none)
   gradientFcn: 'calcgrad'
       initFcn: 'initlay'
    performFcn: 'mse'
      plotFcns: {'plotperform','plottrainstate','plotregression'}
      trainFcn: 'traingdx'

parameters:
    adaptParam: .passes
   divideParam: (none)
 gradientParam: (none)
     initParam: (none)
  performParam: (none)
    trainParam: .show, .showWindow, .showCommandLine, .epochs,
```

```
                .time, .goal, .max_fail, .lr,
                .lr_inc, .lr_dec, .max_perf_inc, .mc,
                .min_grad, .epoch, .Ir

    weight and bias values:
            IW: {2x1 cell} containing 1 input weight matrix
            LW: {2x2 cell} containing 1 layer weight matrix
             b: {2x1 cell} containing 2 bias vectors

    other:
           name: "
        userdata: (user information)

net =
    Neural Network object:
    architecture:
        numInputs: 1
        numLayers: 2
      biasConnect: [1; 1]
     inputConnect: [1; 0]
     layerConnect: [0 0; 1 0]
    outputConnect: [0 1]

       numOutputs: 1  (read-only)
    numInputDelays: 0  (read-only)
    numLayerDelays: 0  (read-only)

    subobject structures:
           inputs: {1x1 cell} of inputs
           layers: {2x1 cell} of layers
          outputs: {1x2 cell} containing 1 output
           biases: {2x1 cell} containing 2 biases
```

```
  inputWeights: {2x1 cell} containing 1 input weight
  layerWeights: {2x2 cell} containing 1 layer weight

functions:
      adaptFcn: 'trains'
     divideFcn: (none)
   gradientFcn: 'calcgrad'
       initFcn: 'initlay'
    performFcn: 'mse'
      plotFcns: {'plotperform','plottrainstate','plotregression'}
      trainFcn: 'traingdx'

parameters:
    adaptParam: .passes
   divideParam: (none)
 gradientParam: (none)
     initParam: (none)
  performParam: (none)
    trainParam: .show, .showWindow, .showCommandLine, .epochs,
                .time, .goal, .max_fail, .lr,
                .lr_inc, .lr_dec, .max_perf_inc, .mc,
                .min_grad, .epoch, .Ir

weight and bias values:
            IW: {2x1 cell} containing 1 input weight matrix
            LW: {2x2 cell} containing 1 layer weight matrix
             b: {2x1 cell} containing 2 bias vectors

other:
          name: ''
      userdata: (user information)
```

```
net =
    Neural Network object:
    architecture:
        numInputs: 1
        numLayers: 2
      biasConnect: [1; 1]
     inputConnect: [1; 0]
     layerConnect: [0 0; 1 0]
    outputConnect: [0 1]

       numOutputs: 1  (read-only)
    numInputDelays: 0  (read-only)
    numLayerDelays: 0  (read-only)

    subobject structures:
           inputs: {1x1 cell} of inputs
           layers: {2x1 cell} of layers
          outputs: {1x2 cell} containing 1 output
           biases: {2x1 cell} containing 2 biases
     inputWeights: {2x1 cell} containing 1 input weight
     layerWeights: {2x2 cell} containing 1 layer weight

    functions:
         adaptFcn: 'trains'
        divideFcn: (none)
      gradientFcn: 'calcgrad'
          initFcn: 'initlay'
       performFcn: 'mse'
         plotFcns: {'plotperform','plottrainstate','plotregression'}
         trainFcn: 'traingdx'

    parameters:
```

```
        adaptParam: .passes
       divideParam: (none)
     gradientParam: (none)
         initParam: (none)
      performParam: (none)
        trainParam: .show, .showWindow, .showCommandLine, .epochs,
                    .time, .goal, .max_fail, .lr,
                    .lr_inc, .lr_dec, .max_perf_inc, .mc,
                    .min_grad, .epoch, .Ir

    weight and bias values:
                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

    other:
                name: ''
            userdata: (user information)

net_keluaran =
    Neural Network object:
    architecture:
            numInputs: 1
            numLayers: 2
       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

        numOutputs: 1  (read-only)
    numInputDelays: 0  (read-only)
    numLayerDelays: 0  (read-only)
```

subobject structures:

inputs: {1x1 cell} of inputs

layers: {2x1 cell} of layers

outputs: {1x2 cell} containing 1 output

biases: {2x1 cell} containing 2 biases

inputWeights: {2x1 cell} containing 1 input weight

layerWeights: {2x2 cell} containing 1 layer weight

functions:

adaptFcn: 'trains'

divideFcn: (none)

gradientFcn: 'calcgrad'

initFcn: 'initlay'

performFcn: 'mse'

plotFcns: {'plotperform','plottrainstate','plotregression'}

trainFcn: 'traingdx'

parameters:

adaptParam: .passes

divideParam: (none)

gradientParam: (none)

initParam: (none)

performParam: (none)

trainParam: .show, .showWindow, .showCommandLine, .epochs,
.time, .goal, .max_fail, .lr,
.lr_inc, .lr_dec, .max_perf_inc, .mc,
.min_grad, .epoch, .Ir

weight and bias values:

IW: {2x1 cell} containing 1 input weight matrix

LW: {2x2 cell} containing 1 layer weight matrix

b: {2x1 cell} containing 2 bias vectors

other:

name: ''

userdata: (user information)

tr =

trainFcn: 'traingdx'

trainParam: [1x1 struct]

performFcn: 'mse'

performParam: [1x1 struct]

divideFcn: ''

divideParam: []

trainInd: [1 2 3 4 5 6 7 8 9 10 11 12]

valInd: []

testInd: []

stop: 'Performance goal met.'

num_epochs: 728

best_epoch: 728

goal: 1.0000e-003

states: {'epoch' 'time' 'perf' 'vperf' 'tperf' 'gradient' 'val_fail' 'lr'}

epoch: [1x729 double]

time: [1x729 double]

perf: [1x729 double]

vperf: [1x729 double]

tperf: [1x729 double]

gradient: [1x729 double]

val_fail: [1x729 double]

lr: [1x729 double]

Y =

0.4082	0.2502	0.1635	0.2365	0.2402	0.8407	0.5234	0.1334	0.1314	0.3460	0.2508	0.4720

E =

0.0252	-0.0169	0.0032	0.0302	-0.0402	0.0259	0.0099	-0.0001	0.0019	-0.0126	0.0492	-0.0720

bobot_hidden =

3.9790	5.7725	2.2546	2.4579	2.6440	3.3842	2.9952	4.0287	0.6496	1.7339	-3.5977	-2.9035
5.6279	-0.3688	3.6218	-2.6084	3.5373	-3.7631	-2.8343	-2.4047	-0.2880	2.5366	4.7175	-2.4462
-4.1344	3.2218	2.6736	4.9241	-2.3878	0.0767	3.5031	2.1038	-3.7939	-0.3681	-3.8887	-2.8253
5.1247	-4.4306	-1.2393	-5.6717	2.2345	2.6062	-1.2814	2.5972	-1.3737	-3.7206	2.5038	-3.1502
1.9663	-1.2221	2.2734	-0.8857	2.2768	6.1763	-3.4280	-1.3390	-3.6440	-2.9029	3.4164	4.0180
-4.5444	4.8278	-3.8656	-1.3463	-3.8969	5.6820	-2.1540	0.7142	2.4613	3.4831	3.1072	0.7121
-2.9270	4.0188	2.8044	3.6134	-5.1861	0.7251	0.9671	-4.2889	-1.8012	-3.3846	-4.1660	0.4811
0.7328	5.4410	-6.0148	3.7298	-0.2867	-4.8990	-0.6430	-3.9971	0.3043	2.8989	-1.1541	-3.1192
5.7896	1.8131	-2.9653	-3.9556	5.7701	-4.7692	-1.4587	0.2415	-3.2585	0.3407	-2.2330	3.2454
5.4018	-5.2368	-5.2199	-0.1293	-1.7056	-2.8365	2.9918	2.2892	0.8085	4.1850	2.6090	1.0313
-4.3960	4.6200	-5.3644	-0.7370	1.0389	4.7802	0.1243	4.2219	-2.1380	-4.0053	-0.8653	-1.5581
6.0884	5.5787	4.1827	1.9298	-3.5817	-3.3337	0.4935	-3.6160	1.4705	-0.4845	3.8401	0.1479

bobot_keluaran =

0.3748 -0.5941 -0.0805 0.0352 0.3066 0.1721 0.3954 0.5897 0.2244 -0.0825 0.5364 0.1800

bias_hidden =

-13.1842
-5.1839
2.9918
1.6917
-3.2719
-2.4358
4.5891
3.7669
2.1046
-0.5929
-1.6770
-1.5507

bias_keluaran =

-0.7365

jumlah_iterasi =

728

nilai_keluaran =

0.4082 0.2502 0.1635 0.2365 0.2402 0.8407 0.5234 0.1334 0.1314 0.3460 0.2508 0.4720

nilai_error =

0.0252 -0.0169 0.0032 0.0302 -0.0402 0.0259 0.0099 -0.0001 0.0019 -0.0126 0.0492 -0.0720

error_MSE =

9.9975e-004

hasil_latih =

0.4082 0.2502 0.1635 0.2365 0.2402 0.8407 0.5234 0.1334 0.1314 0.3460 0.2508 0.4720

max_data =

4650

min_data =

1050

hasil_latih =

1.0e+003 *

2.4367 1.7261 1.3357 1.6643 1.6809 4.3833 2.9554 1.2003 1.1915 2.1569 1.7285 2.7240

filename =

Book1.xlsx

sheet =

1

xlRange =

E7:P7

target_latih_asli =

2550 1650 1350 1800 1500 4500 3000 1200 1200 2100 1950 2400

Lampiran 4

Coding Program Pengujian Jaringan :

```
clc;clear;close all;

% load jaringan yang sudah dibuat pada proses pelatihan
load net.mat

% Proses membaca data uji dari excel
filename = 'Book1.xlsx'
sheet = 2
xlRange = 'D24:P35'

Data = xlsread(filename, sheet, xlRange)
data_uji = Data(:,1:12)'
target_uji = Data(:,13)'
[m,n] = size(data_uji)

% Hasil prediksi
hasil_uji = sim(net_keluaran,data_uji)
nilai_error = hasil_uji-target_uji

max_data = 4650
min_data = 1050
hasil_uji = ((hasil_uji-0.1)*(max_data-min_data)/0.8)+min_data

% Performansi hasil prediksi
error_MSE = (1/n)*sum(nilai_error.^2)

filename = 'Book1.xlsx'
sheet = 1
xlRange = 'E8:P8'

target_uji_asli = xlsread(filename, sheet, xlRange)

figure,
plot(hasil_uji,'bo-')
hold on
plot(target_uji_asli,'ro-')
hold off
grid on
title(strcat(['Grafik Keluaran JST vs Target dengan nilai MSE =
',...
    num2str(error_MSE)]))
xlabel('Pola ke -')
ylabel('Permintaan Pasar Produk Brem')
legend('Keluaran JST','Target','Location','Best')
```

Hasil Proses Penghitungan Dari Program Pengujian Jaringan :

filename =

Book1.xlsx

sheet =

2

xlRange =

D24:P35

Data =

0.4333	0.2333	0.1667	0.2667	0.2000	0.8667	0.5333	0.1333	0.1333	0.3333	0.3000	0.4000	0.4333
0.2333	0.1667	0.2667	0.2000	0.8667	0.5333	0.1333	0.1333	0.3333	0.3000	0.4000	0.4333	0.2667
0.1667	0.2667	0.2000	0.8667	0.5333	0.1333	0.1333	0.3333	0.3000	0.4000	0.4333	0.2667	0.1000
0.2667	0.2000	0.8667	0.5333	0.1333	0.1333	0.3333	0.3000	0.4000	0.4333	0.2667	0.1000	0.1333
0.2000	0.8667	0.5333	0.1333	0.1333	0.3333	0.3000	0.4000	0.4333	0.2667	0.1000	0.1333	0.2667
0.8667	0.5333	0.1333	0.1333	0.3333	0.3000	0.4000	0.4333	0.2667	0.1000	0.1333	0.2667	0.9000
0.5333	0.1333	0.1333	0.3333	0.3000	0.4000	0.4333	0.2667	0.1000	0.1333	0.2667	0.9000	0.5000
0.1333	0.1333	0.3333	0.3000	0.4000	0.4333	0.2667	0.1000	0.1333	0.2667	0.9000	0.5000	0.1667
0.1333	0.3333	0.3000	0.4000	0.4333	0.2667	0.1000	0.1333	0.2667	0.9000	0.5000	0.1667	0.2000

0.3333	0.3000	0.4000	0.4333	0.2667	0.1000	0.1333	0.2667	0.9000	0.5000	0.1667	0.2000	0.3667
0.3000	0.4000	0.4333	0.2667	0.1000	0.1333	0.2667	0.9000	0.5000	0.1667	0.2000	0.3667	0.2333
0.4000	0.4333	0.2667	0.1000	0.1333	0.2667	0.9000	0.5000	0.1667	0.2000	0.3667	0.2333	0.4333

data_uji =

0.4333	0.2333	0.1667	0.2667	0.2000	0.8667	0.5333	0.1333	0.1333	0.3333	0.3000	0.4000
0.2333	0.1667	0.2667	0.2000	0.8667	0.5333	0.1333	0.1333	0.3333	0.3000	0.4000	0.4333
0.1667	0.2667	0.2000	0.8667	0.5333	0.1333	0.1333	0.3333	0.3000	0.4000	0.4333	0.2667
0.2667	0.2000	0.8667	0.5333	0.1333	0.1333	0.3333	0.3000	0.4000	0.4333	0.2667	0.1000
0.2000	0.8667	0.5333	0.1333	0.1333	0.3333	0.3000	0.4000	0.4333	0.2667	0.1000	0.1333
0.8667	0.5333	0.1333	0.1333	0.3333	0.3000	0.4000	0.4333	0.2667	0.1000	0.1333	0.2667
0.5333	0.1333	0.1333	0.3333	0.3000	0.4000	0.4333	0.2667	0.1000	0.1333	0.2667	0.9000
0.1333	0.1333	0.3333	0.3000	0.4000	0.4333	0.2667	0.1000	0.1333	0.2667	0.9000	0.5000
0.1333	0.3333	0.3000	0.4000	0.4333	0.2667	0.1000	0.1333	0.2667	0.9000	0.5000	0.1667
0.3333	0.3000	0.4000	0.4333	0.2667	0.1000	0.1333	0.2667	0.9000	0.5000	0.1667	0.2000
0.3000	0.4000	0.4333	0.2667	0.1000	0.1333	0.2667	0.9000	0.5000	0.1667	0.2000	0.3667
0.4000	0.4333	0.2667	0.1000	0.1333	0.2667	0.9000	0.5000	0.1667	0.2000	0.3667	0.2333

target_uji =

0.4333 0.2667 0.1000 0.1333 0.2667 0.9000 0.5000 0.1667 0.2000 0.3667 0.2333 0.4333

m =

12

n =

12

hasil_uji =

0.5626 0.1156 0.4034 0.0383 0.8795 0.5485 0.3757 0.1921 0.1660 0.3309 0.1542 0.4284

nilai_error =

0.1293 -0.1510 0.3034 -0.0950 0.6129 -0.3515 -0.1243 0.0254 -0.0340 -0.0358 -0.0792 -0.0050

max_data =

4650

min_data =

1050

hasil_uji =

  1.0e+003 *

    3.1317   1.1204   2.4151   0.7725   4.5579   3.0682   2.2909   1.4645   1.3470   2.0890   1.2938   2.5276

error_MSE =

    0.0554

filename =

Book1.xlsx

sheet =

     1

xlRange =

E8:P8

target_uji_asli =

       2550   1800   1050   1200   1800   4650   2850   1350   1500   2250   1650   2550