**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**

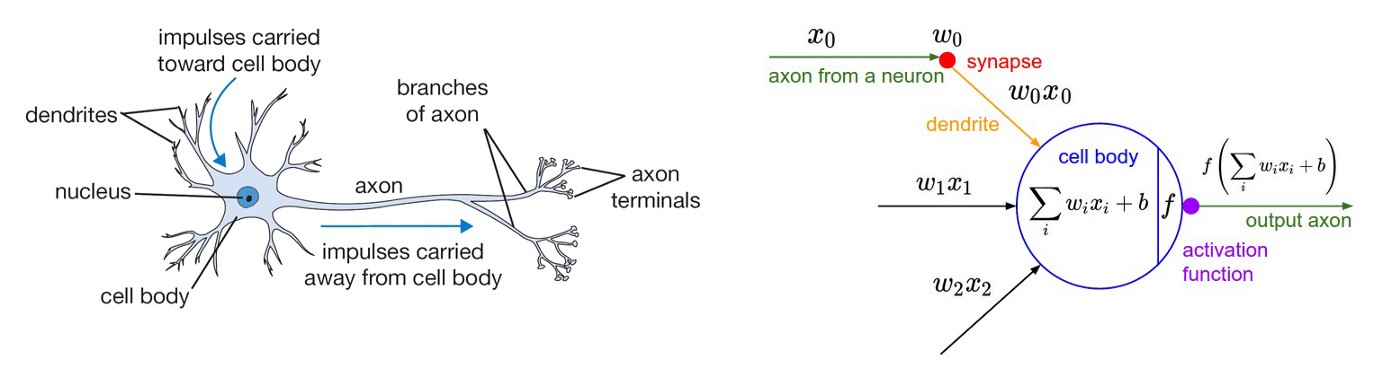
1. Apa itu Neural Network?

2. Bagaimana gambaran dasar sebuah Neural Network?

3. Mengapa Neural Network dibutuhkan?

**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 :

|  |
| --- |
|  |

Penjelasan :

|  |
| --- |
|  |

**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 :

|  |
| --- |
|  |

Penjelasan :

|  |
| --- |
|  |

**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 :

|  |
| --- |
|  |

Penjelasan :

|  |
| --- |
|  |

**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 :

|  |
| --- |
|  |

Output :

|  |
| --- |
|  |

Penjelasan :

|  |
| --- |
|  |

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['output'])

**# 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['output'])

**# test backpropagation of error**

network = [[{'output': 0.7105668883115941, 'weights': [0.13436424411240122, 0.8474337369372327, 0.763774618976614]}],

    [{'output': 0.6213859615555266, 'weights': [0.2550690257394217, 0.49543508709194095]}, {'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]}, {'weights': [0.23244990332399884, 0.3621998343835864, 0.40289821191094327]}],

  [{'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))

**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 |  |  | Tidak Tertarik |  |
|  |  |  |  |  |  |  |
|  |  | Mudah |  |  | Cukup Tertarik |  |
|  |  |  |  |  |  |  |
|  |  | Biasa |  |  | Tertarik |  |
|  |  |  |  |  |  |  |
|  |  | Sulit |  |  | Sangat Tertarik |  |
|  |  |  |  |  |  |  |
|  |  | Sangat Sulit |  |  |  |  |
|  |  |  |  |  |  |  |
| Menerapkan Neural Network untuk melakukan Forecasting. |  |  |  |  |  |  |  |
|  |  | Sangat Mudah |  |  | Tidak Tertarik |  |
|  |  |  |  |  |  |  |
|  |  | Mudah |  |  | Cukup Tertarik |  |
|  |  |  |  |  |  |  |
|  |  | Biasa |  |  | Tertarik |  |
|  |  |  |  |  |  |  |
|  |  | Sulit |  |  | Sangat Tertarik |  |
|  |  |  |  |  |  |  |
|  |  | Sangat Sulit |  |  |  |  |
|  |  |  |  |  |  |  |