Multilayerperceptron

January 31, 2019

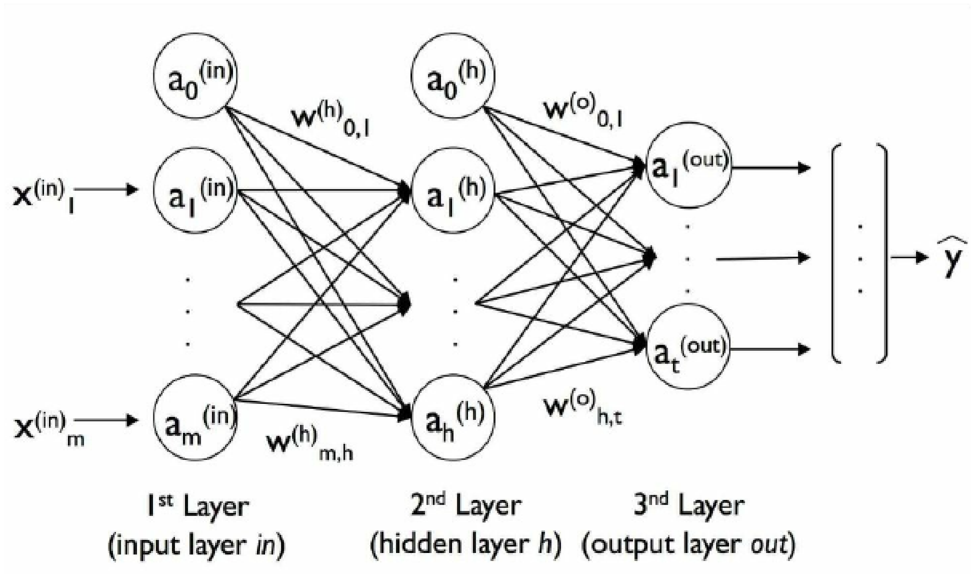
1 Multilayer Perceptron

1.1 Pengantar

- Paling sedikit mempunyai 1 layer antara (intermediate) layer atau sering di sebut hidden layer antara input dan output layer
- Penggunaan:
  - aproksimasi fungsi universal (penyesuaian kurva)
  - pengenalan pola
  - identifikasi proses dan kontrol
  - prediksi time series
  - optimasi sistem

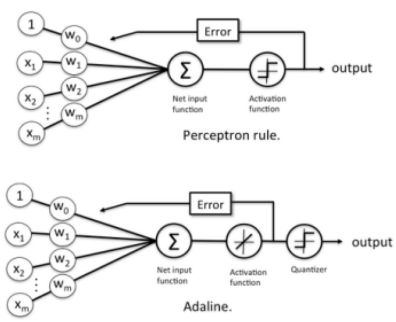
1.2 Arsitektur

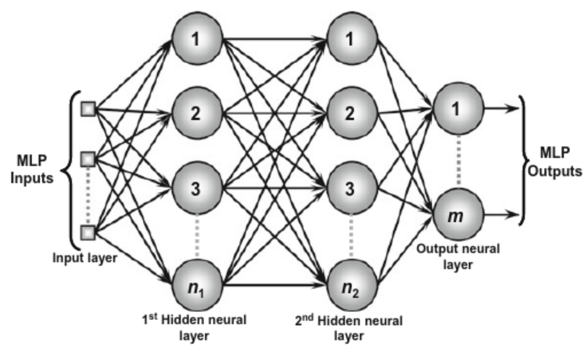
- Termasuk dalam arsitektur jst : Multiple Layer Feedward Architecture
- Training secara tersupervisi
- Mulai populer tahun 1980 dengan dikenalkannya algoritma backpropagation yang memungkinkan proses belajar bagi jaringan ini



1.3 Beda dengan Adaline dan Perceptron

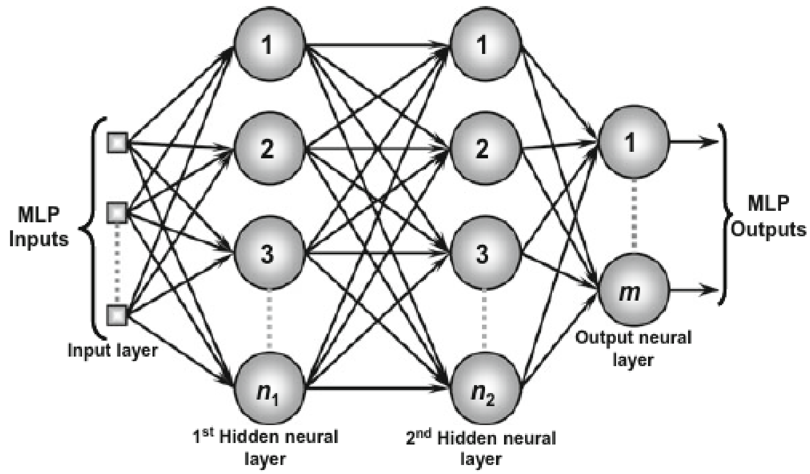
- Ada hidden layer
- Output layer bisa berisi banyak neuron
  - setiap neuron merepresentasikan sebuah output dari proses





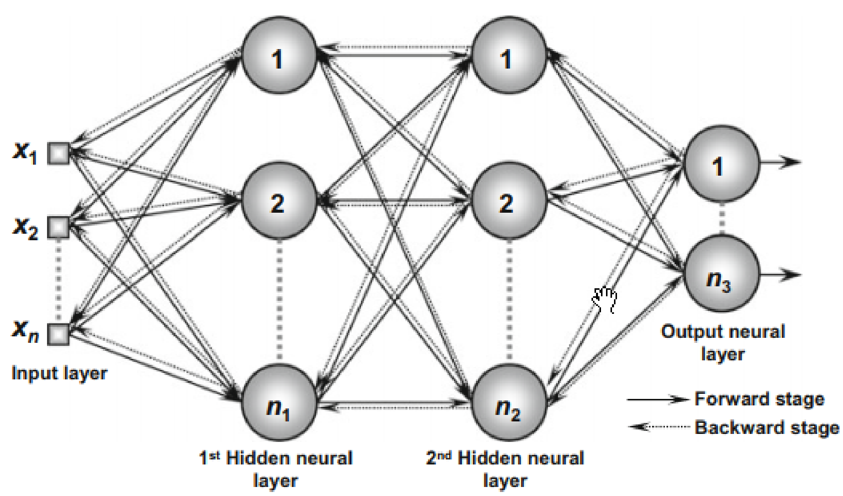
1.4 Prinsip kerja Multilayer Perceptron

Sinyal input merambat dari layer input menuju layer ouput



1.5 Proses Training Multilayer Perceptron

- Proses trainiing menggunakan algoritma backpropagation
- Juga disebut generalized Delta Rule
- Terdiri dari dua tahap:
  - Forward propagation
  - Backward propagation

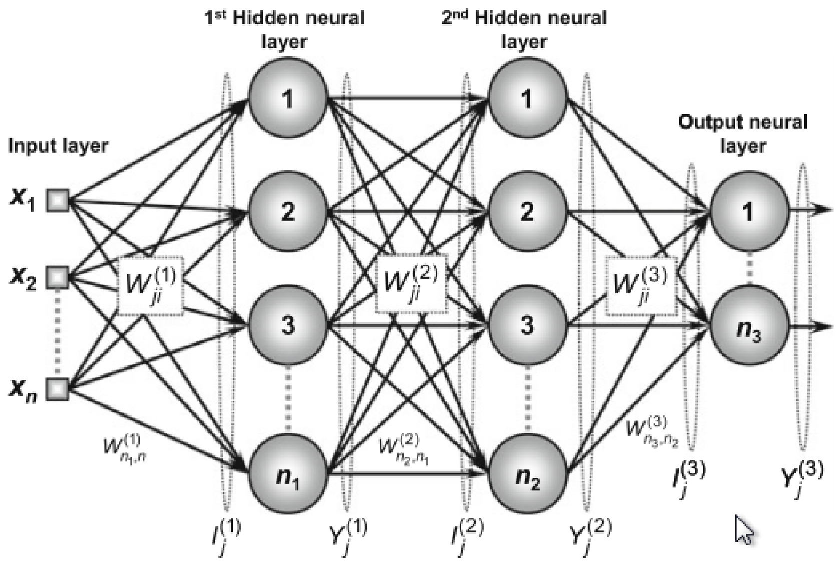


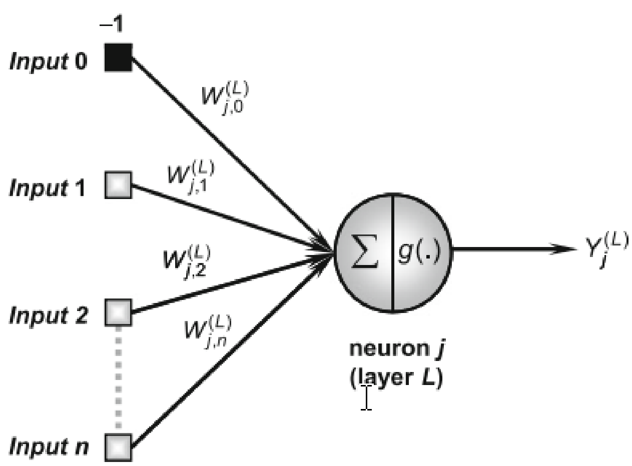
Forward Propagation

1. Sinyal $\{x_1, x_2, \ldots, x_n\}$ dari data training diinputkan ke jaringan
2. Sinyal merambat pada tiap layer hingga menghasilkan output

Backward Propagation

1. Respon yang dihasilkan dari output jaringan dibandingkan dengan respon yang diharapkan (target)
2. Error yang dihitung untuk menyesuaikan bobot dan threshold dari semua neuron





- $\mathbf{W}_{ji}^{(L)}$ adalah bobot sinapsis yang menghubungkan neuron ke j dari layer L dengan input ke i
- $\mathbf{I}_{j}^{(L)}$ adalah vektor dengan elemen yang merupakan perkalian antara input x_i dengan W_{ji} pada layer L:

$$
\begin{aligned}
\mathbf{I}_j^{(1)} &= \sum_{i=0}^{n} W_{ji}^{(1)}.x_i \\
\mathbf{I}_j^{(1)} &= W_{1,0}^{(1)}.x_0 + W_{1,1}^{(1)}.x_1 + \cdots + W_{1,n}^{(1)}.x_n
\end{aligned}
$$

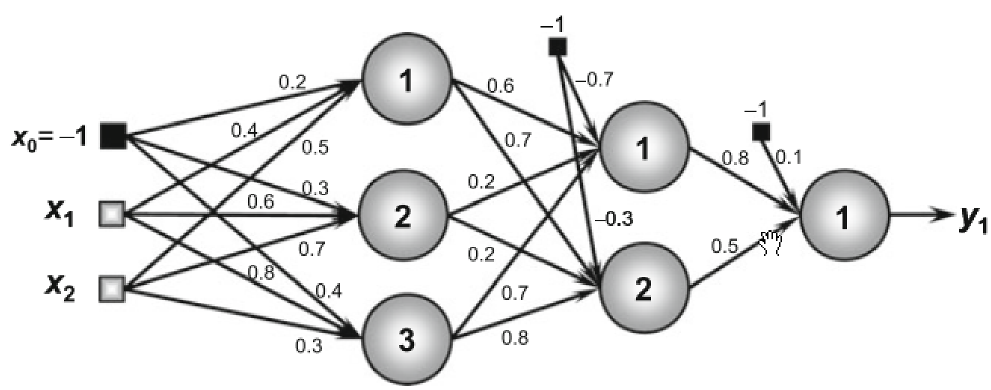
$$
\begin{aligned}
\mathbf{I}_j^{(2)} &= \sum_{i=0}^{n_1} W_{ji}^{(2)}.Y_i^{(1)} \\
\mathbf{I}_j^{(1)} &= W_{1,0}^{(1)}.Y_0^{(1)} + W_{1,1}^{(1)}.Y_1^{(1)} + \cdots + W_{1,n}^{(1)}.Y_{n_1}^{(1)}
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{I}_j^{(3)} &= \sum_{i=0}^{n_2} W_{ji}^{(3)}.Y_i^{(2)} \\
\mathbf{I}_j^{(3)} &= W_{1,0}^{(3)}.Y_0^{(2)} + W_{1,1}^{(3)}.Y_1^{(2)} + \cdots + W_{1,n}^{(3)}.Y_{n_2}^{(2)}
\end{aligned}
$$

$\mathbf{Y}_j^{(L)}$ adalah vektor yang tiap elemennya adalah output dari neuron ke j, dengan g adalah fungsi aktivasi sebagai berikut:

$$
\begin{aligned}
Y_j^{(1)} &= g\big(I_j^{(1)}\big) \\
Y_j^{(2)} &= g\big(I_j^{(2)}\big) \\
Y_j^{(3)} &= g\big(I_j^{(3)}\big)
\end{aligned}
$$

Contoh: MLP dengan 2 input x_1 dan x_2



Contoh MLP

1: Penghitungan aktual output dari jaringan

$$\mathbf{W}_{ji}^{(1)} = \begin{bmatrix} 0.2 & 0.4 & 0.5 \\ 0.3 & 0.6 & 0.7 \\ 0.4 & 0.8 & 0.3 \end{bmatrix}$$

$$\mathbf{W}_{ji}^{(2)} = \begin{bmatrix} -0.7 & 0.6 & 0.2 & 0.7 \\ -0.3 & 0.7 & 0.2 & 0.8 \end{bmatrix}$$

$$\mathbf{W}_{ji}^{(2)} = \begin{bmatrix} 0.1 & 0.8 & 0.5 \end{bmatrix}$$

Dengan input $x_1 = 0.3$ dan $x_2 = 0.7$, dan fungsi aktivasi hyperbolic tangen (tanh) maka nilai $\mathbf{I}_j^{(1)}$:

$$I_j^{(1)} = \begin{bmatrix} I_1^{(1)} \\ I_2^{(1)} \\ I_3^{(1)} \end{bmatrix} = \begin{bmatrix} W_{1,0}^{(1)} \cdot x_0 + W_{1,1}^{(1)} \cdot x_1 + W_{1,2}^{(1)} \cdot x_2 \\ W_{2,0}^{(1)} \cdot x_0 + W_{2,1}^{(1)} \cdot x_1 + W_{2,2}^{(1)} \cdot x_2 \\ W_{3,0}^{(1)} \cdot x_0 + W_{3,1}^{(1)} \cdot x_1 + W_{3,2}^{(1)} \cdot x_2 \end{bmatrix} = \begin{bmatrix} 0.2 \cdot (-1) + 0.4 \cdot 0.3 + 0.5 \cdot 0.7 \\ 0.3 \cdot (-1) + 0.6 \cdot 0.3 + 0.7 \cdot 0.7 \\ 0.4 \cdot (-1) + 0.8 \cdot 0.3 + 0.3 \cdot 0.7 \end{bmatrix} = \begin{bmatrix} 0.27 \\ 0.37 \\ 0.05 \end{bmatrix}$$

$$Y_j^{(1)} = \begin{bmatrix} Y_1^{(1)} \\ Y_2^{(1)} \\ Y_3^{(1)} \end{bmatrix} = \begin{bmatrix} g\left(I_1^{(1)}\right) \\ g\left(I_2^{(1)}\right) \\ g\left(I_3^{(1)}\right) \end{bmatrix} = \begin{bmatrix} \tanh(0.27) \\ \tanh(0.37) \\ \tanh(0.05) \end{bmatrix} = \begin{bmatrix} 0.26 \\ 0.35 \\ 0.05 \end{bmatrix} \xrightarrow{Y_0^{(1)}=-1} Y_j^{(1)} = \begin{bmatrix} Y_0^{(1)} \\ Y_1^{(1)} \\ Y_2^{(1)} \\ Y_3^{(1)} \end{bmatrix} = \begin{bmatrix} -1 \\ 0.26 \\ 0.35 \\ 0.05 \end{bmatrix},$$

$$I_j^{(2)} = \begin{bmatrix} I_1^{(2)} \\ I_2^{(2)} \end{bmatrix} = \begin{bmatrix} W_{1,0}^{(2)} \cdot Y_0^{(1)} + W_{1,1}^{(2)} \cdot Y_1^{(1)} + W_{1,2}^{(2)} \cdot Y_2^{(1)} + W_{1,3}^{(2)} \cdot Y_3^{(1)} \\ W_{2,0}^{(2)} \cdot Y_0^{(1)} + W_{2,1}^{(2)} \cdot Y_1^{(1)} + W_{2,2}^{(2)} \cdot Y_2^{(1)} + W_{2,3}^{(2)} \cdot Y_3^{(1)} \end{bmatrix} = \begin{bmatrix} 0.96 \\ 0.59 \end{bmatrix}$$

$$Y_j^{(2)} = \begin{bmatrix} Y_1^{(2)} \\ Y_2^{(2)} \end{bmatrix} = \begin{bmatrix} g(I_1^{(2)}) \\ g(I_2^{(2)}) \end{bmatrix} = \begin{bmatrix} \tanh(0.96) \\ \tanh(0.59) \end{bmatrix} = \begin{bmatrix} 0.74 \\ 0.53 \end{bmatrix} \xrightarrow{Y_0^{(2)}=-1} Y_j^{(2)} = \begin{bmatrix} Y_0^{(2)} \\ Y_1^{(2)} \\ Y_2^{(2)} \end{bmatrix} = \begin{bmatrix} -1 \\ 0.74 \\ 0.53 \end{bmatrix}$$

$$I_j^{(3)} = \begin{bmatrix} I_1^{(3)} \end{bmatrix} = \begin{bmatrix} W_{1,0}^{(3)} \cdot Y_0^{(2)} + W_{1,1}^{(3)} \cdot Y_1^{(2)} + W_{1,2}^{(3)} \cdot Y_2^{(2)} \end{bmatrix} = [0.76]$$

$$Y_j^{(3)} = \begin{bmatrix} Y_1^{(3)} \end{bmatrix} = \begin{bmatrix} g(I_1^{(3)}) \end{bmatrix} = [\tanh(0.76)] = [0.64]$$

- Karena terletak pada layer terakhir, maka variabel $Y_0^{(3)} = -1$ tidak dimasukkan
- Nilai $Y_1^{(3)}$ adalah output aktual dari jaringan MLP di atas

2. Penentuan fungsi untuk menghitung perkiraan error :

- Tujuan: mengukur penyimpangan antara aktual output dengan output yang diharapkan
- Dengan asumsi p adalah jumlah sampel input dan k adalah nomer sampel, error bisa dihitung dengan rumus MSE:

$$E_{\mathrm{M}} = \frac{1}{p}\sum_{k=1}^{p} E(k),$$

- Penerapan pada contoh di atas: $$E(k) = \frac{1}{2}\sum_{j=1}^{n_3}\left(d_j(k) - Y_j^{(3)}(k)\right)^2,$$

3. Penyesuaian bobot

---

1. **Penyesuaian bobot pada layer output** Tujuan: menyesuaikan bobot matriks $\mathbf{W}_{ji}^{(3)}$ untuk meminimalisasi error Menggunakan aturan seperti pada ADALINE, dengan menerapkan definisi gradien dan penggunaan chain rule dari turunan fungsi, didapatkan:

$$\Delta E^{(3)} = \frac{\delta E}{\delta W_{ji}^{(3)}} = \frac{\delta E}{\delta Y_j^{(3)}}.\frac{\delta Y_j^{(3)}}{\delta I_j^{(3)}}.\frac{\delta I_j^{(3)}}{\delta W_{ji}^{(3)}}$$

Dengan $\frac{\delta I_j^{(3)}}{\delta W_{ji}^{(3)}} = Y_i^{(2)}$, $\frac{\delta Y_j^{(3)}}{\delta I_j^{(3)}} = g'\left(I_j^{(3)}\right)$, dan $\frac{\delta E}{\delta Y_j^{(3)}} = -\left(d_j - Y_j^{(3)}\right)$ maka:

$$\frac{\partial E}{\partial W_{ji}^{(3)}} = -\left(d_j - Y_j^{(3)}\right) \cdot g'\left(I_j^{(3)}\right) \cdot Y_i^{(2)}$$

. * g' adalah turunan pertama dari fungsi aktivasi * d adalah nilai target yang diharapkan

Dengan η adalah learning rate, maka

$$\Delta W_{ji}^{(3)} = -\eta.\frac{\delta E}{\delta W_{ji}^{(3)}}$$
$$= \eta.\delta_j^{(3)}.Y_i^{(2)}$$

dengan $\delta_j^{(3)}$ adalah lgradien lokal yang terkait dengan neuron ke j dari layer output, dengan rumus:

$$\delta_j^{(3)} = \left(d_j - Y_j^{(3)}\right).g'\left(I_j^{(3)}\right)$$

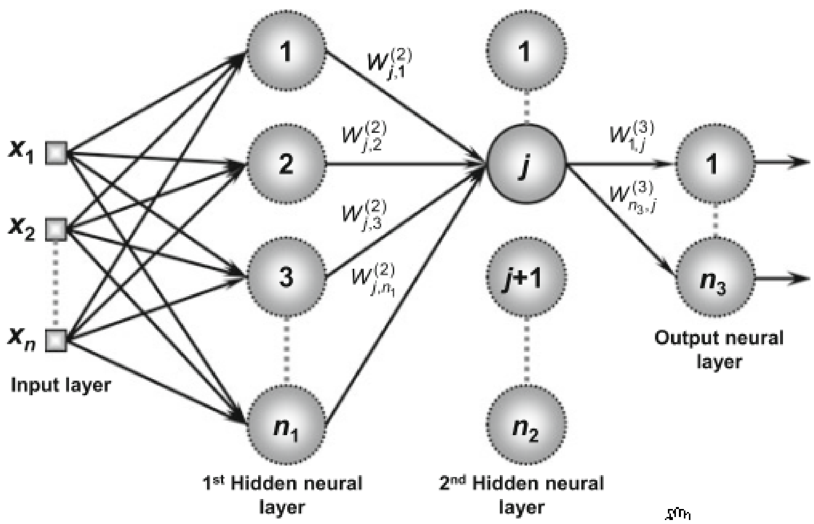
Maka :

$$W_{ji}^{(3)}(t+1) = W_{ji}^{(3)}(t) + \eta.\delta_j^{(3)}.Y_i^{(2)}$$

dengan bentuk rumus algoritma menjadi:

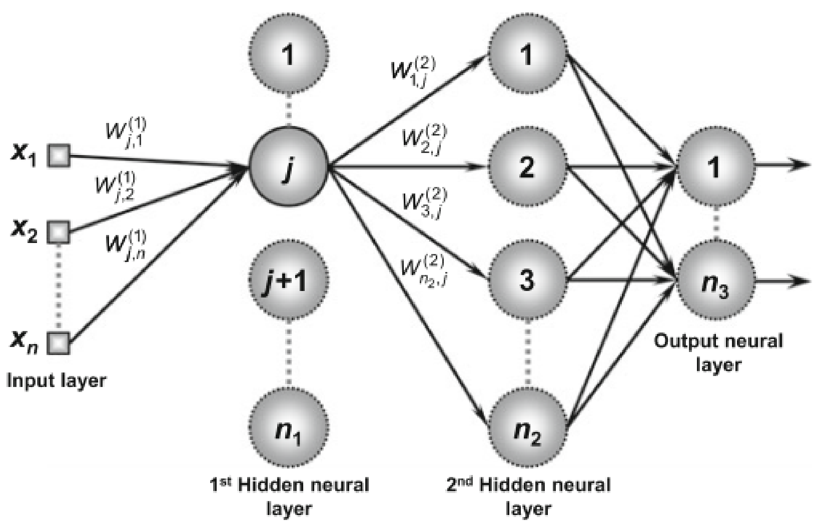
$$W_{ji}^{(3)} \leftarrow W_{ji}^{(3)} + \eta \cdot \delta_j^{(3)} \cdot Y_i^{(2)}$$

2. **Penyesuaian bobot pada layer tengah kedua**



$$W_{ji}^{(2)}(t+1) = W_{ji}^{(2)}(t) + \eta \cdot \delta_j^{(2)} \cdot Y_i^{(1)}$$

$$W_{ji}^{(2)} \leftarrow W_{ji}^{(2)} + \eta \cdot \delta_j^{(2)} \cdot Y_i^{(1)}$$



$$W_{ji}^{(1)}(t+1) = W_{ji}^{(1)}(t) + \eta \cdot \delta_j^{(1)} \cdot x_i$$

Analog dengan penyesuaian bobot pada layer output, rumusnya menjadi:
Atau:

3. **Penyesuaian bobot pada layer tengah pertama**

Analog dengan penyesuaian bobot pada layer sebelumnya, rumusnya menjadi:
Atau:

$$W_{ji}^{(1)} \leftarrow W_{ji}^{(1)} + \eta \cdot \delta_j^{(1)} \cdot x_i$$

1.5.1 Implementasi dalam Algoritma Pemrograman

Begin {MLP Algorithm – Training Phase}

<1> Obtain the set of training samples $\{\boldsymbol{x}^{(k)}\}$;

<2> Associate the vector with the desired output $\{\boldsymbol{d}^{(k)}\}$ for each training sample;

<3> Initialize $W_{ji}^{(1)}$, $W_{ji}^{(2)}$ and $W_{ji}^{(3)}$ with small random values;

<4> Specify the learning rate $\{\eta\}$ and the required precision $\{\varepsilon\}$;

<5> Initialize the epoch counter $\{epoch \leftarrow 0\}$;

<6> Repeat:

<6.1> $E_M^{previous} \leftarrow E_M$; {according to (5.8)}

<6.2> For all train samples $\{\boldsymbol{x}^{(k)}, \boldsymbol{d}^{(k)}\}$, do:

<6.2.1> Obtain $I_j^{(1)}$ and $Y_j^{(1)}$; {according to (5.1) and (5.4)}

<6.2.2> Obtain $I_j^{(2)}$ and $Y_j^{(2)}$; {according to (5.2) and (5.5)}

<6.2.3> Obtain $I_j^{(3)}$ and $Y_j^{(3)}$; {according to (5.3) and (5.6)}

Forward steps

<6.2.4> Determine $\delta_j^{(3)}$; {according to (5.15)}

<6.2.5> Adjust $W_{ji}^{(3)}$; {according to (5.17)}

<6.2.6> Determine $\delta_j^{(2)}$; {according to (5.26)}

<6.2.7> Adjust $W_{ji}^{(2)}$; {according to (5.28)}

<6.2.8> Determine $\delta_j^{(1)}$; {according to (5.37)}

<6.2.9> Adjust $W_{ji}^{(1)}$; {according to (5.39)}

Backward steps

<6.3> Obtain the adjusted $Y_j^{(3)}$; {according to <6.2.1>, <6.2.2> and <6.2.3>}

<6.4> $E_M^{current} \leftarrow E_M$; {according to (5.8)}

<6.5> $epoch \leftarrow epoch + 1$;

Until: $\left|E_M^{current} - E_M^{previous}\right| \leq \varepsilon$

End {MLP Algorithm – Training Phase}

Begin {MLP Algorithm – Operation Phase}

<1> Obtain a sample $\{\boldsymbol{x}\}$;

<2> Assume $W_{ji}^{(1)}$, $W_{ji}^{(2)}$ and $W_{ji}^{(3)}$ already adjusted in the training stage;

<3> Execute the following instructions:

<3.1> Obtain $I_j^{(1)}$ and $Y_j^{(1)}$; {according to (5.1) and (5.4)}

<3.2> Obtain $I_j^{(2)}$ and $Y_j^{(2)}$; {according to (5.2) and (5.5)}

<3.3> Obtain $I_j^{(3)}$ and $Y_j^{(3)}$; {according to (5.3) and (5.6)}

Forward steps

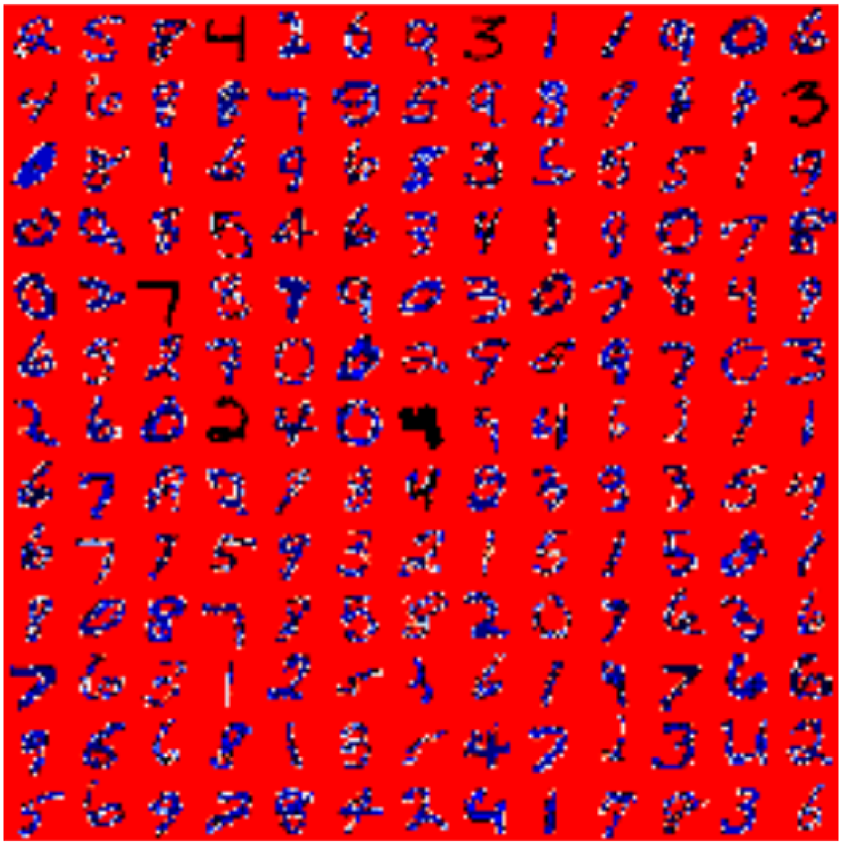
<4> Publish the outputs of the network, which are given by the elements of $Y_j^{(3)}$.

End {MLP Algorithm – Operation Phase}

1.6 Contoh implementasi dalam python untuk pengenalan tulisan tangan

1.6.1 Data

Dari database MNIST



Membaca Data

```
In [4]: %matplotlib inline
        import numpy as np
        import matplotlib.pyplot as plt
        image_size = 28 # width and length
        no_of_different_labels = 10 #  i.e. 0, 1, 2, 3, ..., 9
        image_pixels = image_size * image_size
        data_path = "data/"
        train_data = np.loadtxt(data_path + "mnist_train.csv",delimiter=",")
        test_data = np.loadtxt(data_path + "mnist_test.csv",delimiter=",")
```

1.6.2 Data ke array dengan nilai 0 < data < 1

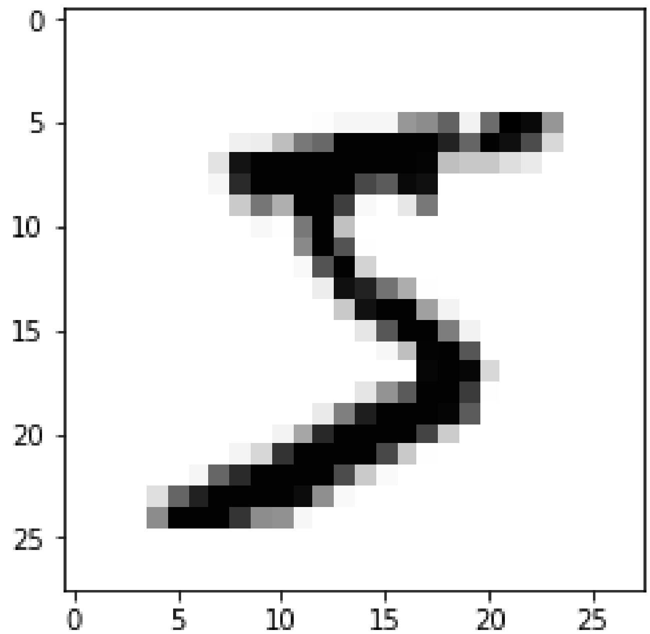
```
In [5]: fac = 255  *0.99 + 0.01
        train_imgs = np.asfarray(train_data[:, 1:]) / fac
        test_imgs = np.asfarray(test_data[:, 1:]) / fac
        train_labels = np.asfarray(train_data[:, :1])
        test_labels = np.asfarray(test_data[:, :1])
```

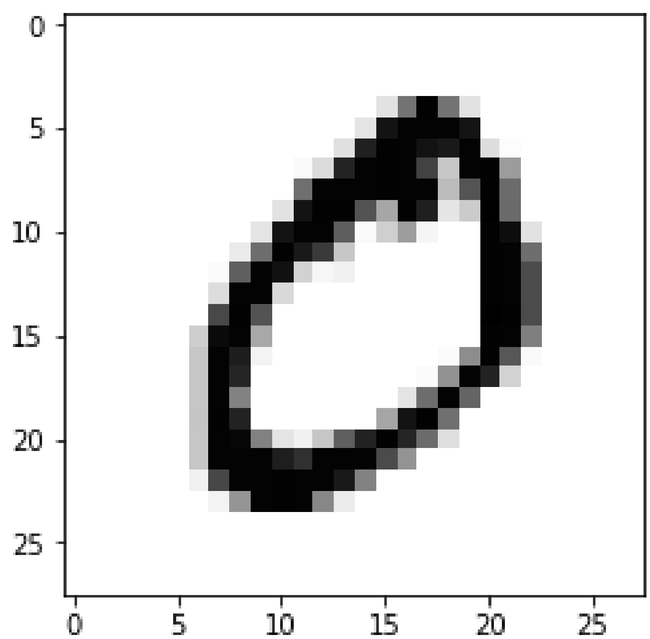
In [6]:
```
lr = np.arange(10)
for label in range(10):
    one_hot = (lr==label).astype(np.int)
    print("label: ", label, " in one-hot representation: ", one_hot)
```

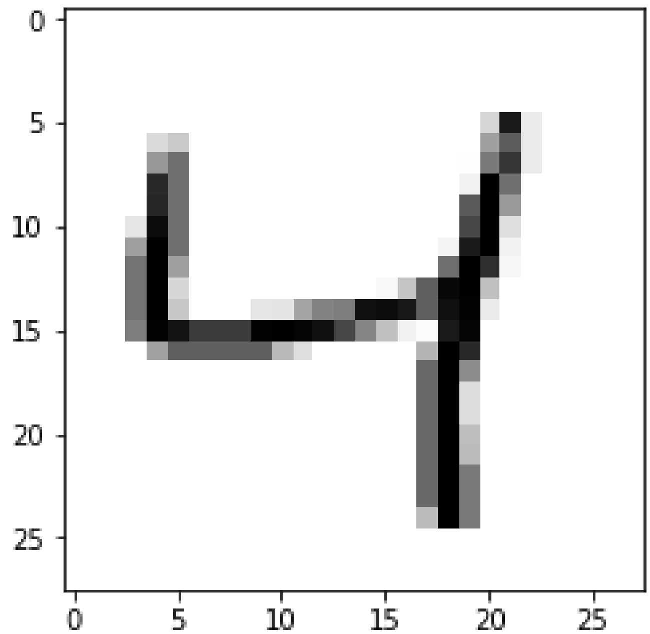
```
label:  0  in one-hot representation:  [1 0 0 0 0 0 0 0 0 0]
label:  1  in one-hot representation:  [0 1 0 0 0 0 0 0 0 0]
label:  2  in one-hot representation:  [0 0 1 0 0 0 0 0 0 0]
label:  3  in one-hot representation:  [0 0 0 1 0 0 0 0 0 0]
label:  4  in one-hot representation:  [0 0 0 0 1 0 0 0 0 0]
label:  5  in one-hot representation:  [0 0 0 0 0 1 0 0 0 0]
label:  6  in one-hot representation:  [0 0 0 0 0 0 1 0 0 0]
label:  7  in one-hot representation:  [0 0 0 0 0 0 0 1 0 0]
label:  8  in one-hot representation:  [0 0 0 0 0 0 0 0 1 0]
label:  9  in one-hot representation:  [0 0 0 0 0 0 0 0 0 1]
```

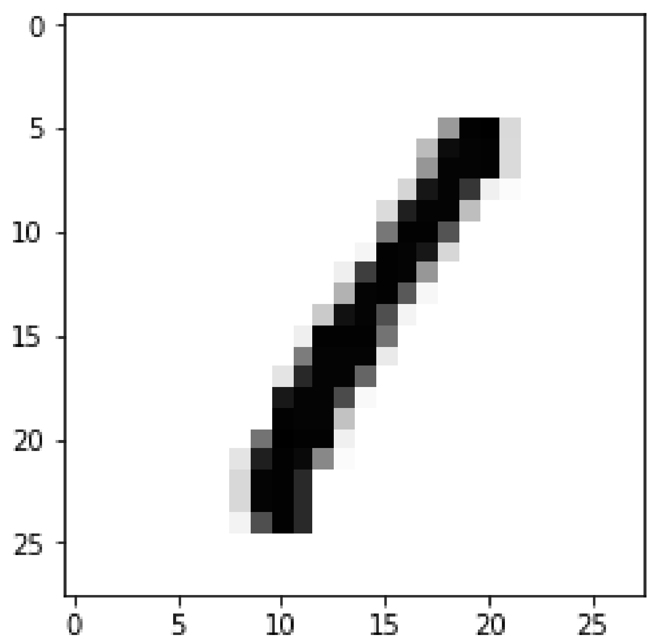
In [7]:
```
lr = np.arange(no_of_different_labels)
# transform labels into one hot representation
train_labels_one_hot = (lr==train_labels).astype(np.float)
test_labels_one_hot = (lr==test_labels).astype(np.float)
# we don't want zeroes and ones in the labels neither:
train_labels_one_hot[train_labels_one_hot==0] = 0.01
train_labels_one_hot[train_labels_one_hot==1] = 0.99
test_labels_one_hot[test_labels_one_hot==0] = 0.01
test_labels_one_hot[test_labels_one_hot==1] = 0.99
```

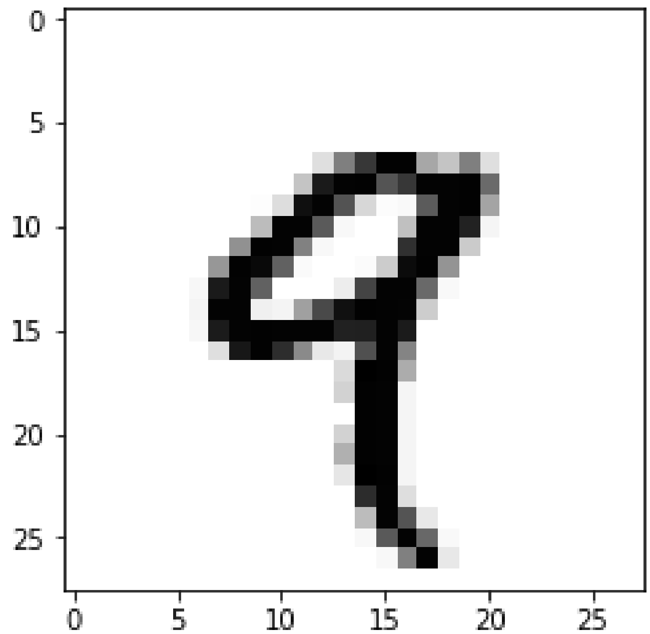
In [8]:
```
for i in range(10):
    img = train_imgs[i].reshape((28,28))
    plt.imshow(img, cmap="Greys")
    plt.show()
```

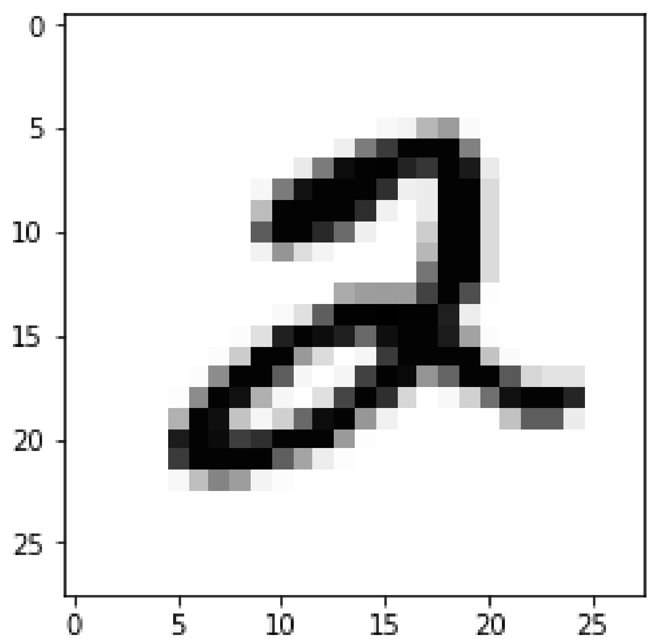

0
5
10
15
20
25
0
5
10
15
20
25

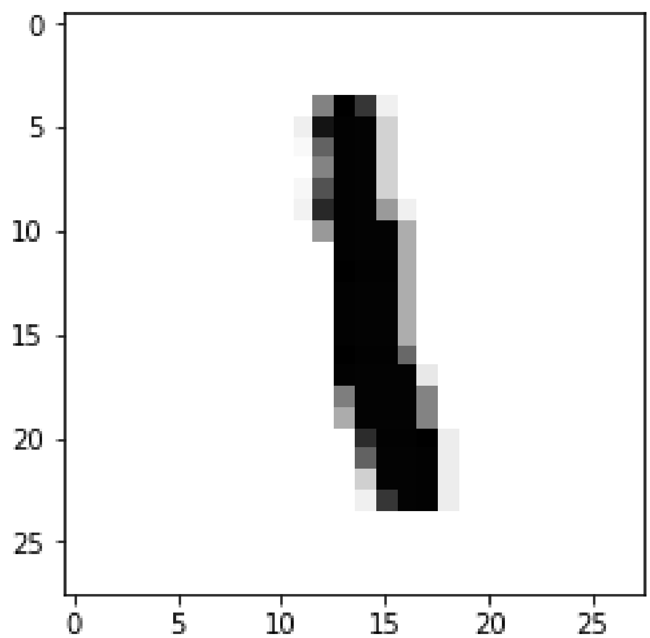

0
5
10
15
20
25
0
5
10
15
20
25

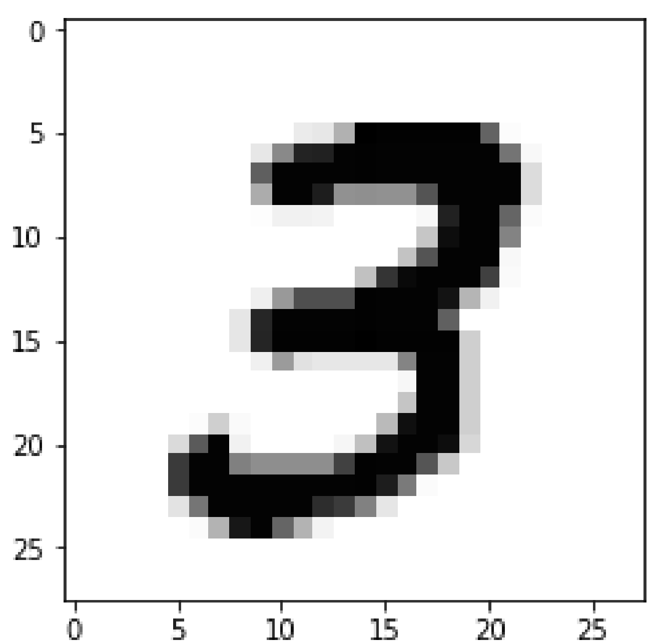

0
5
10
15
20
25
0
5
10
15
20
25

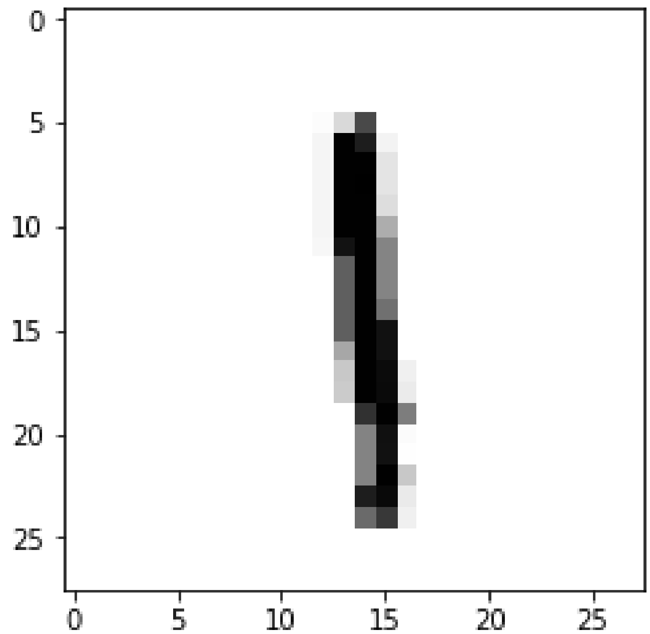

0
5
10
15
20
25
0
5
10
15
20
25

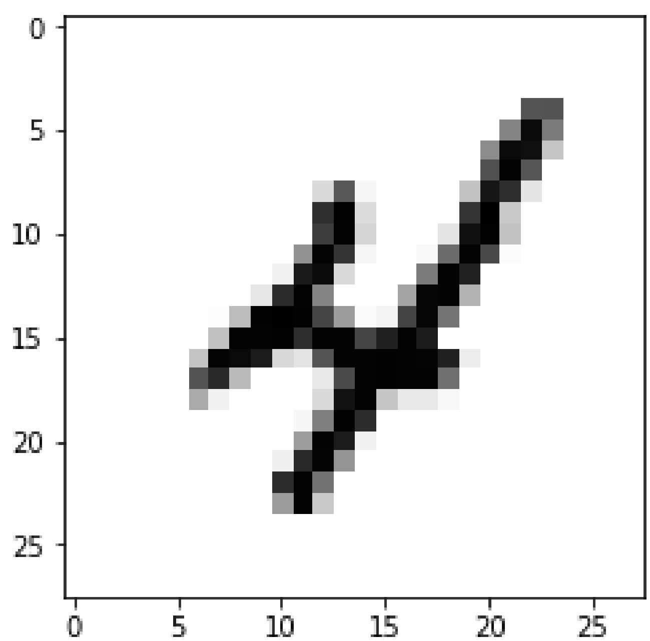

0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25






```
In [9]: import pickle
        with open("data/pickled_mnist.pkl", "bw") as fh:
            data = (train_imgs,
```

```
                test_imgs,
                train_labels,
                test_labels,
                train_labels_one_hot,
                test_labels_one_hot)
        pickle.dump(data, fh)
```

In [10]:
```
import pickle
with open("data/pickled_mnist.pkl", "br") as fh:
    data = pickle.load(fh)
train_imgs = data[0]
test_imgs = data[1]
train_labels = data[2]
test_labels = data[3]
train_labels_one_hot = data[4]
test_labels_one_hot = data[5]
image_size = 28 # width and length
no_of_different_labels = 10 #  i.e. 0, 1, 2, 3, ..., 9
image_pixels = image_size * image_size
```

In [11]:
```
import numpy as np
@np.vectorize
def sigmoid(x):
    return 1 / (1 + np.e ** -x)
activation_function = sigmoid
from scipy.stats import truncnorm
def truncated_normal(mean=0, sd=1, low=0, upp=10):
    return truncnorm((low - mean) / sd,
                     (upp - mean) / sd,
                     loc=mean,
                     scale=sd)
class NeuralNetwork:

    def __init__(self,
                 no_of_in_nodes,
                 no_of_out_nodes,
                 no_of_hidden_nodes,
                 learning_rate):
        self.no_of_in_nodes = no_of_in_nodes
        self.no_of_out_nodes = no_of_out_nodes
        self.no_of_hidden_nodes = no_of_hidden_nodes
        self.learning_rate = learning_rate
        self.create_weight_matrices()

    def create_weight_matrices(self):
        """
        A method to initialize the weight
        matrices of the neural network
```

```
        """
        rad = 1 / np.sqrt(self.no_of_in_nodes)
        X = truncated_normal(mean=0,
                             sd=1,
                             low=-rad,
                             upp=rad)
        self.wih = X.rvs((self.no_of_hidden_nodes,
                                       self.no_of_in_nodes))
        rad = 1 / np.sqrt(self.no_of_hidden_nodes)
        X = truncated_normal(mean=0, sd=1, low=-rad, upp=rad)
        self.who = X.rvs((self.no_of_out_nodes,
                                         self.no_of_hidden_nodes))


    def train(self, input_vector, target_vector):
        """
        input_vector and target_vector can
        be tuple, list or ndarray
        """

        input_vector = np.array(input_vector, ndmin=2).T
        target_vector = np.array(target_vector, ndmin=2).T

        output_vector1 = np.dot(self.wih,
                                input_vector)
        output_hidden = activation_function(output_vector1)

        output_vector2 = np.dot(self.who,
                                output_hidden)
        output_network = activation_function(output_vector2)

        output_errors = target_vector - output_network
        # update the weights:
        tmp = output_errors * output_network \
              * (1.0 - output_network)
        tmp = self.learning_rate  * np.dot(tmp,
                                           output_hidden.T)
        self.who += tmp
        # calculate hidden errors:
        hidden_errors = np.dot(self.who.T,
                               output_errors)
        # update the weights:
        tmp = hidden_errors * output_hidden * \
              (1.0 - output_hidden)
        self.wih += self.learning_rate \
                          * np.dot(tmp, input_vector.T)
```

```
    def run(self, input_vector):
        # input_vector can be tuple, list or ndarray
        input_vector = np.array(input_vector, ndmin=2).T
        output_vector = np.dot(self.wih,
                               input_vector)
        output_vector = activation_function(output_vector)

        output_vector = np.dot(self.who,
                               output_vector)
        output_vector = activation_function(output_vector)

        return output_vector

    def confusion_matrix(self, data_array, labels):
        cm = np.zeros((10, 10), int)
        for i in range(len(data_array)):
            res = self.run(data_array[i])
            res_max = res.argmax()
            target = labels[i][0]
            cm[res_max, int(target)] += 1
        return cm
    def precision(self, label, confusion_matrix):
        col = confusion_matrix[:, label]
        return confusion_matrix[label, label] / col.sum()

    def recall(self, label, confusion_matrix):
        row = confusion_matrix[label, :]
        return confusion_matrix[label, label] / row.sum()


    def evaluate(self, data, labels):
        corrects, wrongs = 0, 0
        for i in range(len(data)):
            res = self.run(data[i])
            res_max = res.argmax()
            if res_max == labels[i]:
                corrects += 1
            else:
                wrongs += 1
        return corrects, wrongs
```

In [12]:
```
ANN = NeuralNetwork(no_of_in_nodes = image_pixels,
                    no_of_out_nodes = 10,
                    no_of_hidden_nodes = 100,
                    learning_rate = 0.1)
```

```
for i in range(len(train_imgs)):
    ANN.train(train_imgs[i], train_labels_one_hot[i])
```

In [13]:
```
for i in range(20):
    res = ANN.run(test_imgs[i])
    print(test_labels[i], np.argmax(res), np.max(res))
```

```
[7.] 7 0.996850895396134
[2.] 2 0.879290068314333
[1.] 1 0.9885382005449624
[0.] 0 0.9598546392323661
[4.] 4 0.9610990523952812
[1.] 1 0.9874147335917179
[4.] 4 0.9960447065142569
[9.] 9 0.9866138343156539
[5.] 6 0.23178749562747264
[9.] 9 0.9289345810333837
[0.] 0 0.9806986301638966
[6.] 6 0.7904147868669377
[9.] 9 0.9965318364360122
[0.] 0 0.9795655851391541
[1.] 1 0.9933031137099365
[5.] 5 0.9385219157487653
[9.] 9 0.9910696854526403
[7.] 7 0.995788882250648
[3.] 3 0.8211988380272226
[4.] 4 0.993765852340242
```

1.6.3 Pengukuran Performa

In [14]:
```
benar, salah = ANN.evaluate(train_imgs, train_labels)
print("akurasi data training: ",benar/(benar + salah))
benar, salah = ANN.evaluate(test_imgs, test_labels)
print("akurasi data testing: ", benar/(benar + salah))
```

```
akurasi data training:  0.9491333333333334
akurasi data testing:  0.9497
```

1.6.4 Training dengan multi epoch

In [15]:
```
import numpy as np
@np.vectorize   #keterangan: dengan dekorasi ini, maka fungsi
def sigmoid(x): #sigmoid bisa digunakan untuk vektor output
    return 1/(1 + np.e ** -x)

fungsi_aktivasi = sigmoid
```

```python
from scipy.stats import truncnorm

def truncated_normal(mean=0,sd=1, low=0, uper=10):
    return truncnorm((low - mean)/sd,
                     (uper - mean)/sd,
                     loc=mean,
                     scale=sd)


class NeuralNetwork:
    '''
    kelas untuk jst.
    '''
    def __init__(self,jumlah_node_input,
                 jumlah_node_output,
                 jumlah_node_hidden,
                 learning_rate):
        self.jumlah_node_input = jumlah_node_input
        self.jumlah_node_output = jumlah_node_output
        self.jumlah_node_hidden = jumlah_node_hidden
        self.learning_rate = learning_rate
        self.init_bobot()

    def init_bobot(self):
        """
        metode untuk inisialisasi bobot
        menggunakan fungsirandom rvs
        """
        rad = 1/np.sqrt(self.jumlah_node_input)
        X = truncated_normal(mean=0,
                             sd=1,
                             low=-rad,
                             uper=rad)
        #Bobot antara node input dengan node hidden
        self.w_ih = X.rvs((self.jumlah_node_hidden,self.jumlah_node_input))

        rad = 1/np.sqrt(self.jumlah_node_hidden)
        X = truncated_normal(mean=0,
                             sd=1,
                             low=-rad,
                             uper=rad)
        #bobot antara node hidden dengan node output
        self.w_ho = X.rvs((self.jumlah_node_output, self.jumlah_node_hidden))

    def train_single(self,input_vector, target_vector):

        output_vectors = []
```

```
        input_vector = np.array(input_vector,ndmin=2).T
        target_vector = np.array(target_vector, ndmin=2).T

        output_vector1 = np.dot(self.w_ih, input_vector)
        output_hidden = activation_function(output_vector1)

        output_vector2 = np.dot(self.w_ho, output_hidden)
        output_network = activation_function(output_vector2)

        output_errors = target_vector -output_network


        #update bobot layer hidden ke output
        tmp = output_errors * output_network * (1.0 - output_network)

        tmp = self.learning_rate * np.dot(tmp,output_hidden.T)

        self.w_ho += tmp

        #hitung error pada node hidden
        hidden_errors = np.dot(self.w_ho.T, output_errors)

        #update bobot input ke hidden
        tmp = hidden_errors * output_hidden * (1.0 - output_hidden)
        self.w_ih += self.learning_rate * np.dot(tmp, input_vector.T)

    def train(self,data_array,label_data_array,
              epochs = 1, intermediate_result=False):
        intermediate_weights = []
        for epoch in range(epochs):
            print("*",end="")
            for i in range(len(data_array)):
                self.train_single(data_array[i],label_data_array[i])
            if intermediate_result:
                intermediate_weights.append((self.w_ih.copy(),
                                             self.w_ho.copy()))
        return intermediate_weights

    def confusion_matrix(self, data_array, labels):
        cm = {}
        for i in range(len(data_array)):
            res = self.run(data_array[i])
            res_max = res.argmax()
            target = labels[i][0]
            if (target, res_max) in cm:
                cm[(target, res_max)] += 1
            else:
                cm[(target, res_max)] = 1
```

```
            return cm

    def run(self, input_vector):
        """ input_vector can be tuple, list or ndarray """

        input_vector = np.array(input_vector, ndmin=2).T
        output_vector = np.dot(self.wih,
                               input_vector)
        output_vector = activation_function(output_vector)

        output_vector = np.dot(self.who,
                               output_vector)
        output_vector = activation_function(output_vector)

        return output_vector


    def evaluate(self, data, labels):
        corrects, wrongs = 0, 0
        for i in range(len(data)):
            res = self.run(data[i])
            res_max = res.argmax()
            if res_max == labels[i]:
                corrects += 1
            else:
                wrongs += 1
        return benar, salah
```

In [16]:
```
train_labels_one_hot
```

Out[16]:
```
array([[0.01, 0.01, 0.01, ..., 0.01, 0.01, 0.01],
       [0.99, 0.01, 0.01, ..., 0.01, 0.01, 0.01],
       [0.01, 0.01, 0.01, ..., 0.01, 0.01, 0.01],
       ...,
       [0.01, 0.01, 0.01, ..., 0.01, 0.01, 0.01],
       [0.01, 0.01, 0.01, ..., 0.01, 0.01, 0.01],
       [0.01, 0.01, 0.01, ..., 0.01, 0.99, 0.01]])
```

In [17]:
```
epochs = 10
NN = NeuralNetwork(jumlah_node_input=image_pixels,
                   jumlah_node_output = 10,
                    jumlah_node_hidden=100,
                    learning_rate=0.15)
bobot = ANN.train(train_imgs, train_labels_one_hot,
                  epochs=epochs,
                 intermediate_result=True)
```

```
TypeError                                 Traceback (most recent call last)

<ipython-input-17-93109359d188> in <module>
  6 bobot = ANN.train(train_imgs, train_labels_one_hot,
  7                    epochs=epochs,
----> 8                   intermediate_result=True)


TypeError: train() got an unexpected keyword argument 'epochs'
```

In [ ]:
```python
rad = 1/np.sqrt(2)
print(rad)
X = truncated_normal(mean=0,
                    sd=1,
                    low=-rad,
                    uper=rad)
print(X)
```

In [ ]:
```python
X.rvs((4,4))
```

In [ ]:
```python
X.rvs(4)
```

In [ ]: