Machine Learning

Indah Agustien Siradjuddin

Artificial Neural Networks ¶

Semester Gasal 2019-2020

Single layer Perceptron

Single layer Perceptron / *Perceptron* : untuk data yang dapat dipisahkan secara linear, *linearly separable data*, bukan *unlinearly separable data*, seperti data logika XOR e.g. logical data XOR

In [1]:

```
import numpy as np
import matplotlib.pyplot as matPlot
#inisialisasi data
data=np.array([[1,1,1],[1,1,0],[1,0,1],[1,0,0]]) # bias dan data
targetAnd=np.array([1,0,0,0])
targetOr=np.array([1,1,1,0])
targetXor=np.array([0,1,1,0])
```

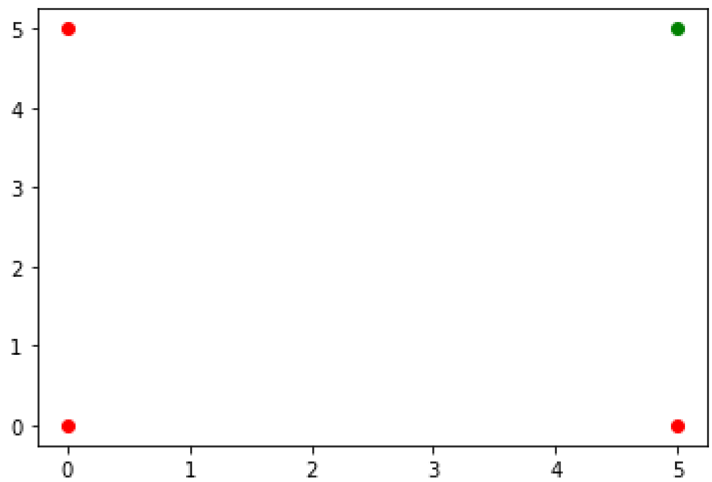
Linearly Separable Data

In [2]:

```
import perceptron
```

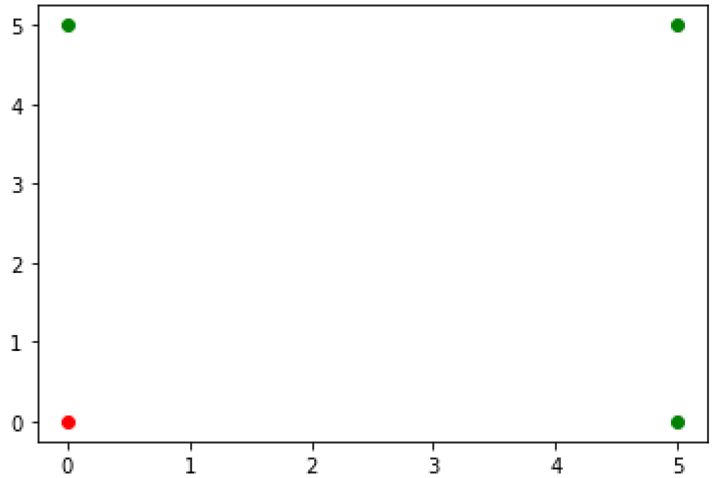
In [3]:

```
perceptron.plotting(data,targetAnd)
```



In [4]:

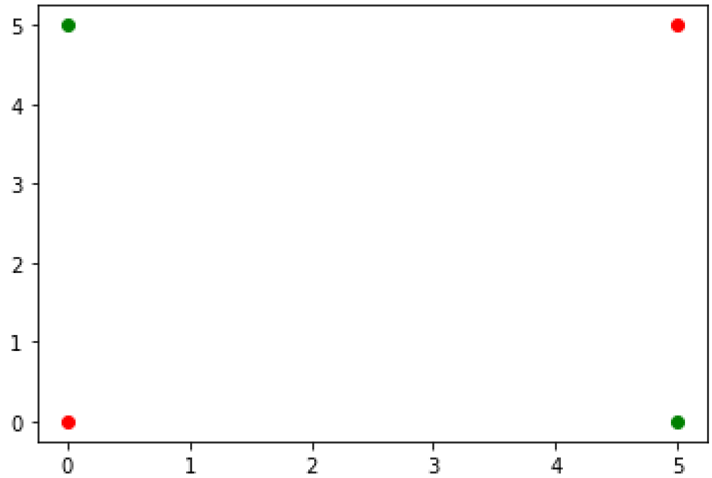
```
perceptron.plotting(data,targetOr)
```



Unlinearly Separable Data

In [5]:

```
perceptron.plotting(data,targetXor)
```



Multilayer Perceptron

Berdasarkan grafik data XOR, maka dibutuhkan lebih dari satu garis untuk memisahkan data XOR. Logika XOR dapat juga didefinisikan sebagai berikut :

x_1 XOR x_2 = (x_1 AND (NOT x_2)) OR (x_2 AND (NOT x_1))

Latihan :

1. Dengan menggunakan tabel kebenaran, buktikan ekivalensi logika XOR tersebut
2. Buat Model XOR dengan menggunakan code perceptron

In [6]:

```
import numpy as np
import matplotlib.pyplot as matPlot
#inisialisasi data
data=np.array([[1,1,1],[1,1,0],[1,0,1],[1,0,0]]) # bias dan data
targetAndNot1=np.array([0,1,0,0])
targetAndNot2=np.array([0,0,1,0])
targetOr=np.array([1,1,1,0])
```

In [7]:

```
import perceptron
```

In [8]:

```
w=perceptron.perceptronLearning(data,targetAndNot1)
```

```
data= [[1 1 1]
 [1 1 0]
 [1 0 1]
 [1 0 0]]
target= [0 1 0 0]
bobot= [[0.1]
 [0.2]
 [0.3]]
bobot= [[0.1]
 [0.2]
 [0.3]]
jumlah epoch5
epoch =  0 : [[ 0.1 -0.9  0.1  0.1 -0.9]
 [ 0.2 -0.8  0.2  0.2  0.2]
 [ 0.3 -0.7 -0.7 -0.7 -0.7]]
epoch =  1 : [[ 0.1 -0.9  0.1  0.1 -0.9 -0.9  0.1  0.1 -0.9]
 [ 0.2 -0.8  0.2  0.2  0.2  0.2  1.2  1.2  1.2]
 [ 0.3 -0.7 -0.7 -0.7 -0.7 -0.7 -0.7 -0.7 -0.7]]
epoch =  2 : [[ 0.1 -0.9  0.1  0.1 -0.9 -0.9  0.1  0.1 -0.9 -0.9 -0.9 -0.9
-0.9]
 [ 0.2 -0.8  0.2  0.2  0.2  0.2  1.2  1.2  1.2  1.2  1.2  1.2  1.2]
 [ 0.3 -0.7 -0.7 -0.7 -0.7 -0.7 -0.7 -0.7 -0.7 -0.7 -0.7 -0.7 -0.7]]
epoch =  3 : [[ 0.1 -0.9  0.1  0.1 -0.9 -0.9  0.1  0.1 -0.9 -0.9 -0.9 -0.9
-0.9 -0.9
  -0.9 -0.9 -0.9]
 [ 0.2 -0.8  0.2  0.2  0.2  0.2  1.2  1.2  1.2  1.2  1.2  1.2  1.2  1.2
   1.2  1.2  1.2]
 [ 0.3 -0.7 -0.7 -0.7 -0.7 -0.7 -0.7 -0.7 -0.7 -0.7 -0.7 -0.7 -0.7 -0.7
  -0.7 -0.7 -0.7]]
epoch =  4 : [[ 0.1 -0.9  0.1  0.1 -0.9 -0.9  0.1  0.1 -0.9 -0.9 -0.9 -0.9
-0.9 -0.9
  -0.9 -0.9 -0.9 -0.9 -0.9 -0.9 -0.9]
 [ 0.2 -0.8  0.2  0.2  0.2  0.2  1.2  1.2  1.2  1.2  1.2  1.2  1.2  1.2
   1.2  1.2  1.2  1.2  1.2  1.2  1.2]
 [ 0.3 -0.7 -0.7 -0.7 -0.7 -0.7 -0.7 -0.7 -0.7 -0.7 -0.7 -0.7 -0.7 -0.7
  -0.7 -0.7 -0.7 -0.7 -0.7 -0.7 -0.7]]
```

In [9]:

```
w1=w[:,-1]
print('w and not 1=',w1)
```

```
w and not 1= [-0.9  1.2 -0.7]
```

In [10]:

```
w=perceptron.perceptronLearning(data,targetAndNot2)
```

```
data= [[1 1 1]
 [1 1 0]
 [1 0 1]
 [1 0 0]]
target= [0 0 1 0]
bobot= [[0.1]
 [0.2]
 [0.3]]
bobot= [[0.1]
 [0.2]
 [0.3]]
jumlah epoch5
epoch =  0 : [[ 0.1 -0.9 -0.9  0.1 -0.9]
 [ 0.2 -0.8 -0.8 -0.8 -0.8]
 [ 0.3 -0.7 -0.7  0.3  0.3]]
epoch =  1 : [[ 0.1 -0.9 -0.9  0.1 -0.9 -0.9 -0.9  0.1 -0.9]
 [ 0.2 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8]
 [ 0.3 -0.7 -0.7  0.3  0.3  0.3  0.3  1.3  1.3]]
epoch =  2 : [[ 0.1 -0.9 -0.9  0.1 -0.9 -0.9 -0.9  0.1 -0.9 -0.9 -0.9 -0.9
-0.9]
 [ 0.2 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8]
 [ 0.3 -0.7 -0.7  0.3  0.3  0.3  0.3  1.3  1.3  1.3  1.3  1.3  1.3]]
epoch =  3 : [[ 0.1 -0.9 -0.9  0.1 -0.9 -0.9 -0.9  0.1 -0.9 -0.9 -0.9 -0.9
-0.9 -0.9
  -0.9 -0.9 -0.9]
 [ 0.2 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8
  -0.8 -0.8 -0.8]
 [ 0.3 -0.7 -0.7  0.3  0.3  0.3  0.3  1.3  1.3  1.3  1.3  1.3  1.3  1.3
   1.3  1.3  1.3]]
epoch =  4 : [[ 0.1 -0.9 -0.9  0.1 -0.9 -0.9 -0.9  0.1 -0.9 -0.9 -0.9 -0.9
-0.9 -0.9
  -0.9 -0.9 -0.9 -0.9 -0.9 -0.9 -0.9]
 [ 0.2 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8
  -0.8 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8]
 [ 0.3 -0.7 -0.7  0.3  0.3  0.3  0.3  1.3  1.3  1.3  1.3  1.3  1.3  1.3
   1.3  1.3  1.3  1.3  1.3  1.3  1.3]]
```

In [11]:

```
w2=w[:,-1]
print('w and not 2=',w2)
```

```
w and not 2= [-0.9 -0.8  1.3]
```

In [12]:

```
w=perceptron.perceptronLearning(data,targetOr)
```

```
data= [[1 1 1]
 [1 1 0]
 [1 0 1]
 [1 0 0]]
target= [1 1 1 0]
bobot= [[0.1]
 [0.2]
 [0.3]]
bobot= [[0.1]
 [0.2]
 [0.3]]
jumlah epoch5
epoch =  0 : [[ 0.1  0.1  0.1  0.1 -0.9]
 [ 0.2  0.2  0.2  0.2  0.2]
 [ 0.3  0.3  0.3  0.3  0.3]]
epoch =  1 : [[ 0.1  0.1  0.1  0.1 -0.9  0.1  0.1  0.1 -0.9]
 [ 0.2  0.2  0.2  0.2  0.2  1.2  1.2  1.2  1.2]
 [ 0.3  0.3  0.3  0.3  0.3  1.3  1.3  1.3  1.3]]
epoch =  2 : [[ 0.1  0.1  0.1  0.1 -0.9  0.1  0.1  0.1 -0.9 -0.9 -0.9 -0.9
-0.9]
 [ 0.2  0.2  0.2  0.2  0.2  1.2  1.2  1.2  1.2  1.2  1.2  1.2  1.2]
 [ 0.3  0.3  0.3  0.3  0.3  1.3  1.3  1.3  1.3  1.3  1.3  1.3  1.3]]
epoch =  3 : [[ 0.1  0.1  0.1  0.1 -0.9  0.1  0.1  0.1 -0.9 -0.9 -0.9 -0.9
-0.9 -0.9
  -0.9 -0.9 -0.9]
 [ 0.2  0.2  0.2  0.2  0.2  1.2  1.2  1.2  1.2  1.2  1.2  1.2  1.2  1.2
   1.2  1.2  1.2]
 [ 0.3  0.3  0.3  0.3  0.3  1.3  1.3  1.3  1.3  1.3  1.3  1.3  1.3  1.3
   1.3  1.3  1.3]]
epoch =  4 : [[ 0.1  0.1  0.1  0.1 -0.9  0.1  0.1  0.1 -0.9 -0.9 -0.9 -0.9
-0.9 -0.9
  -0.9 -0.9 -0.9 -0.9 -0.9 -0.9 -0.9]
 [ 0.2  0.2  0.2  0.2  0.2  1.2  1.2  1.2  1.2  1.2  1.2  1.2  1.2  1.2
   1.2  1.2  1.2  1.2  1.2  1.2  1.2]
 [ 0.3  0.3  0.3  0.3  0.3  1.3  1.3  1.3  1.3  1.3  1.3  1.3  1.3  1.3
   1.3  1.3  1.3  1.3  1.3  1.3  1.3]]
```

In [13]:

```
w3=w[:,-1]
print('w or =',w3)
```

```
w or = [-0.9  1.2  1.3]
```

Learning/Training

Learning/Training adalah proses untuk membangun sebuah model, yaitu arsitektur dan bobotnya (*weights*), termasuk bias, dengan menggunakan dataset pelatihan/pembelajaran.
Jenis Pembelajaran :

1. Supervised learning : data pelatihan (input/feature dan target). Target data yang berperan sebagai 'pengawas', dengan cara memperbaharui bobot (dan bias) dengan tujuan meminimalkan error antara target (output yang diinginkan) dengan output yang sebenarnya.
2. Unsupervised learning : data pelatihan hanya terdiri dari data input/feature saja
3. Reinforcement learning : mekanisme *reward* and *punishment*

Aturan Pembelajaran

Bobot di-*update* dengan cara meminimalkan error antara target dengan output yang sebenarnya

Gradient Descent Learning Rule

Fungsi Error : *Sum Squared Error*

$$E = \sum_{p=1}^{n} (t_p - y_p)^2$$

Dimana :

- t_i : target/output yang diinginkan dari input data-i
- y_i : output yang sebenarnya dari input data-i
- n : jumlah data pelatihan

Gradient descent untuk meminimalkan error :

Sumber :

Sebagaimana yang terdapat pada algoritma perceptron-3, bobot di-*update* dengan menggunakan : $w_i(t+1) = w_i(t) + \Delta x_i(t)$, dimana Δ adalah selisih antara target dengan output.

Pada *Gradient Descent Learning*, bobot di-*update* dengan menggunakan persamaan berikut:

$$\Delta x_i(t) = \eta\big(-\frac{\partial E}{\partial w_i}\big)$$
$$\frac{\partial E}{\partial w_i} = -2(t_p - y_p)\frac{\partial f}{\partial net_p} x_{i,p}$$

dimana :

- η adalah **learning rate**

Akan tetapi, turunan dari fungsi aktivasi sangat sulit didapatkan jika fungsi aktivasi berbentuk diskontinyu, seperti fungsi *ramp* dan *stepwise*.

Widrow Hoff Learning Rule

Aturan pembelajaran juga sering disebut sebagai *Least Mean Square* (LMS). Aturan ini digunakan untuk fungsi aktivasi diskontinyu, dengan asumsi $f = net_p$. Oleh karena itu turunan dari fungsi tersebut adalah $\frac{\delta f}{\delta net_p}$ is 1.

$$\frac{\partial E}{\partial w_i} = -2(t_p - y_p)x_{i,p}$$

Bobot di-*update* berdasarkan persamaan berikut:

$$w_i(t+1) = w_i(t) + 2\eta(t_p - y_p)x_{i,p}$$

Updating Weights

1. Stochastic/online Learning : setiap iterasi
2. Batch/offline learning : setiap epoch

Backpropagation

MLP yang sering digunakan berdasarkan algoritma pembelajaran supervised dengan aturan Gradient Descent adalah **Backpropagation Neural Network**. Tahapan *Backpropagation*.:

1. **Feedforward Pass**, menghitung semua output pada semua neuron di setiap *layer*
2. **Backward Propagation**, error (selisih antara target dan output) dikirim kembali dari output layer ke input layer, sehingga setiap bobot dapat di-update

   Sumber :

- neuron pada layer input $x_0, x_1, \ldots, x_i$, dimana jumlah neuron (fitur) adalah d
- neuron pada hidden layer $y_0, y_1, \ldots, y_j$, dimana jumlah neuron pada hidden layer adalah j
- neuron pada output layer $O_0, O_1, \ldots, O_k$, dimana jumlah neuron pada output layer adalah k
- w_{ij} bobot antara neuron pada input layer x_i dan neuron pada hidden layer y_j
- v_{jk} bobot antara neuron pada hidden layer y_j dan neuron pada output layer O_k

Data Iris:
Bobot antara input dan hidden layer :

Bobot antara hidden dan output layer

Feedforward Pass

Output neuron pada hidden dan output layer dihitung berdasarkan fungsi aktivasi (sigmoid):

$$\begin{aligned} y_j &= f_{yj}(net_{yj}) \\ &= \frac{1}{1+e^{-net_{yj}}} \\ O_k &= f_{Ok}(net_{Ok}) \\ &= \frac{1}{1+e^{-net_{Ok}}} \end{aligned}$$

dimana

$$\begin{aligned} net_{yj} &= \sum_{i=0}^{d}(x_i w_{ij}) \\ net_{Ok} &= \sum_{j=0}^{J}(y_j v_{jk}) \end{aligned}$$

- y_j adalah output atau nilai aktivasi dari neuron pada hidden layer
- O_k adalah output atau nilai aktivasi dari neuron pada output layer

Neuron Hidden Layer

Net input

$$
\begin{aligned}
net_{yj} &= \sum_{i=0}^{4}(x_i w_{ij}) \\
net_{y1} &= x_0 \times w_{01} + x_1 \times w_{11} + x2_{\times} w_{21} + x_3 \times w_{w31} + x_4 \times w_{41} \\
&= 1 \times 0.1 + 5.1 \times 0.2 + 3.5 \times 0.1 + 1.4 \times 0.15 + 0.2 \times 0.2 = 1.72 \\
net_{y2} &= x_0 \times w_{02} + x_1 \times w_{12} + x2_{\times} w_{22} + x_3 \times w_{w32} + x_4 \times w_{42} \\
&= 1 \times 0.3 + 5.1 \times 0.4 + 3.5 \times 0.1 + 1.4 \times 0.3 + 0.2 \times 0.2 = 3.15 \\
net_{y3} &= x_0 \times w_{03} + x_1 \times w_{13} + x2_{\times} w_{23} + x_3 \times w_{w33} + x_4 \times w_{43} \\
&= 1 \times 0.4 + 5.1 \times 0.15 + 3.5 \times 0.25 + 1.4 \times 0.4 + 0.2 \times 0.5 = 2.7
\end{aligned}
$$

Dengan menggunakan matriks :

$$
\begin{bmatrix} y_1 & y_2 & y_3 & y_4 \end{bmatrix} = \begin{bmatrix} x_0 & x_1 & x_2 & x_3 & x_4 \end{bmatrix} \begin{bmatrix} w_{01} & w_{02} & w_{03} \\ w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix}
$$

Nilai aktivasi

$$
\begin{aligned}
y_j &= f_{yj}(net_{yj}) \\
&= \frac{1}{1 + e^{-net_{yj}}} \\
y_1 &= \frac{1}{1 + e^{-1.72}} = 0.85 \\
y_2 &= \frac{1}{1 + e^{-3.15}} = 0.96 \\
y_3 &= \frac{1}{1 + e^{-2.7}} = 0.94
\end{aligned}
$$

In [14]:

```
import numpy as np
```

In [15]:

```
#Fungsi Aktivasi
def sigmoidFn(x):
    return 1/(1+np.exp(-x))
```

Hidden Layer

In [16]:

```
inp=np.array([1,5.1,3.5,1.4,0.2])
w=np.array([[0.1,0.3,0.4],[0.2,0.4,0.15],[0.1,0.1,0.25],[0.15,0.3,0.4],[0.2,0.2,0.5]])
net_y=inp.dot(w)
print('NetInput Hidden=',net_y)
y=sigmoidFn(net_y)
print('OutputHidden=',y)
```

```
NetInput Hidden= [1.72 3.15 2.7 ]
OutputHidden= [0.84812884 0.95890872 0.93702664]
```

Output Layer
Net Input

$$
\begin{aligned}
net_{Ok} &= \sum_{j=0}^{2}(y_j v_{jk}) \\
net_{O1} &= y_0 \times v_{01} + y_1 \times v_{11} + y_2 \times v_{21} + y_3 \times v_{31} \\
&= 1 \times 0.3 + 0.85 \times 0.05 + 0.96 \times 0.1 + 0.94 \times 0.4 = 0.81 \\
net_{O2} &= y_0 \times v_{02} + y_1 \times v_{12} + y_2 \times v_{22} + y_3 \times v_{32} \\
&= 1 \times 0.2 + 0.85 \times 0.1 + 0.96 \times 0.3 + 0.94 \times 0.4 = 0.95
\end{aligned}
$$

Persamaan Matriks :

$$
\begin{bmatrix} O_1 & O_2 \end{bmatrix} = \begin{bmatrix} y_0 & y_1 & y_2 & y_3 \end{bmatrix} \begin{bmatrix} v_{01} & v_{02} \\ v_{11} & v_{12} \\ v_{21} & v_{22} \\ v_{31} & v_{32} \end{bmatrix}
$$

nilai aktivasi/output dari Backpropagation :

$$
\begin{aligned}
O_k &= f_{O_k}(net_{O_k}) \\
&= \frac{1}{1 + e^{-net_{O_k}}} \\
O_1 &= \frac{1}{1 + e^{-0.81}} = 0.69 \\
O_2 &= \frac{1}{1 + e^{-0.95}} = 0.72
\end{aligned}
$$

Output Layer

In [17]:

```
dataHidden=np.ones((1,4))
dataHidden[0,1:]=y
v=np.array([[0.3,0.2],[0.05,0.1],[0.1,0.3],[0.4,0.4]])
net_O=dataHidden.dot(v)
o=sigmoidFn(net_O)
print('NetInput Output=',net_O)
print('Output=',o)
```

```
NetInput Output= [[0.81310797 0.94729616]]
Output= [[0.6927714  0.72057109]]
```

Backward Propagation

Bobot di-*update* untuk meminimalkan fungsi error.
Error dihitung dan dipropagasi ke seluruh layer.
Fungsi error yang digunakan *sum of squarred errors* (SSE) :

$$E_p = \frac{1}{2}\Big(\frac{\sum_{k=1}^{K}(t_{k,p} - O_{k,p})^2}{K}\Big)$$

dimana :

- K : jumlah class/label/output neurons
- p : pola-p

Update bobot antara hidden dan output layer:

$$v_{jk}(t) = v_{jk}(t-1) + \Delta v_{jk}(t)$$

Update bobot antara input dan hidden layer:

$$w_{ij}(t) = w_{ij}(t-1) + \Delta w_{ij}(t)$$

Update bobot between hidden and ouput layer

$$\Delta v_{jk} = \eta\Big(-\frac{\partial E}{\partial v_{jk}}\Big)$$

$$\frac{\partial E}{\partial v_{jk}} = \frac{\partial E}{\partial O_k}\frac{\partial O_k}{\partial v_{jk}}$$

Derivasi Error berdasarkan O_k

$$\frac{\partial E}{\partial O_k} = \frac{\partial\Big(\frac{\frac{1}{2}\big(\sum_{k=1}^{K}(t_{k,p}-O_{k,p})^2\big)}{K}\Big)}{\partial O_k}$$
$$= -(t_k - O_{k,p})$$

Derivasi O_k berdasarkan v_jk :

$$\frac{\partial O_k}{\partial v_{jk}} = \frac{\partial O_k}{\partial net_{Ok}} \frac{\partial net_{Ok}}{\partial v_{jk}}$$

Derivasi O_k berdasarkan net_{Ok}

$$\begin{aligned}
\frac{\partial O_k}{\partial net_{Ok}} &= \frac{f_{Ok}}{\partial net_{Ok}} \\
&= \frac{\partial\left(\frac{1}{1+e^{-net_{ok}}}\right)}{\partial net_{Ok}} \\
&= -1\left(1+e^{-net_{ok}}\right)^{-2}. -1.e^{-net_{ok}} \\
&= \frac{e^{-net_{ok}}}{\left(1+e^{-net_{ok}}\right)^2} \\
&= \frac{1}{\left(1+e^{-net_{ok}}\right)} \frac{e^{-net_{ok}}}{\left(1+e^{-net_{ok}}\right)} \\
&= O_k(1-O_k)
\end{aligned}$$

Derivasi net_{Ok} berdasarkan v_{jk}:

$$\begin{aligned}
\frac{\partial net_{Ok}}{\partial v_{jk}} &= \frac{\partial\left(\sum_{j=0}^{J} v_{jk} y_j\right)}{\partial v_{jk}} \\
&= y_j
\end{aligned}$$

Sehingga,

$$\begin{aligned}
\frac{\partial O_k}{\partial v_{jk}} &= O_k(1-O_k) y_j \\
&= f'_{Ok} y_j
\end{aligned}$$

Bobot antara hidden dan output layer di-*update* dengan menggunakan persamaan berikut :

$$\begin{aligned}
\Delta v_{jk} &= \eta\left(\frac{-E}{\partial v_{jk}}\right) \\
&= -\eta \frac{\partial E}{\partial O_k} \frac{\partial O_k}{\partial v_{jk}} \\
&= -\eta. -(t_k - O_{kp}) f'_{Ok} y_j \\
&= \eta \delta O k y_j
\end{aligned}$$

Bobot antara Output dan hidden

Hitung *delta weights* antara output dan hidden layer (Δv).
Iterasi pertama (dari epoch pertama) , target data adalah $t_1 = 0$ and $t_2 = 0$. Asusmsi laju pembelajaran $\eta = 0.05$, maka *delta weights*:

$$\begin{aligned}
\delta Ok &= (t_k - O_k) \times O_k \times (1 - O_k) \\
\delta O1 &= (t_1 - O_1) \times O_1 \times (1 - O_1) \\
&= (0 - 0.69) \times 0.69 \times (1 - 0.69) = -0.148 \\
\delta O2 &= (t_2 - O_2) \times O_2 \times (1 - O_2) \\
&= (0 - 0.72) \times 0.72 \times (1 - 0.72) = -0.145 \\
\Delta v_{jk} &= \eta \delta Oky_j \\
\Delta v_{01} &= \eta \delta O1y_0 \\
&= 0.05 \times (-0.148) \times 1 = -0.0074 \\
\Delta v_{11} &= \eta \delta O1y_1 \\
&= 0.05 \times (-0.148) \times 0.85 = -0.0063 \\
\Delta v_{21} &= \eta \delta O1y_2 \\
&= 0.05 \times (-0.148) \times 0.96 = -0.0071 \\
\Delta v_{31} &= \eta \delta O1y_3 \\
&= 0.05 \times (-0.148) \times 0.94 = -0.0069 \\
\Delta v_{02} &= \eta \delta O2y_0 \\
&= 0.05 \times (-0.145) \times 1 = -0.0072 \\
\Delta v_{12} &= \eta \delta O2y_1 \\
&= 0.05 \times (-0.145) \times 0.85 = -0.0062 \\
\Delta v_{22} &= \eta \delta O2y_2 \\
&= 0.05 \times (-0.145) \times 0.96 = -0.0069 \\
\Delta v_{32} &= \eta \delta O2y_3 \\
&= 0.05 \times (-0.145) \times 0.94 = -0.0068
\end{aligned}$$

update bobot antara hidden dan output layer

$$\begin{aligned}
v_{jk}(2) &= v_{jk}(1) + \Delta v_{jk}(1) \\
v_{01}(2) &= v_{01}(1) + \Delta v_{01}(1) = 0.3 - 0.0074 = 0.2926 \\
v_{11}(2) &= v_{11}(1) + \Delta v_{11}(1) = 0.05 - 0.0063 = 0.0437 \\
v_{21}(2) &= v_{21}(1) + \Delta v_{21}(1) = 0.1 - 0.0071 = 0.0929 \\
v_{31}(2) &= v_{31}(1) + \Delta v_{31}(1) = 0.4 - 0.0069 = 0.3931 \\
v_{02}(2) &= v_{02}(1) + \Delta v_{02}(1) = 0.2 - 0.0072 = 0.1928 \\
v_{12}(2) &= v_{12}(1) + \Delta v_{12}(1) = 0.1 - 0.0062 = 0.0938 \\
v_{22}(2) &= v_{22}(1) + \Delta v_{22}(1) = 0.3 - 0.0069 = 0.2931 \\
v_{32}(2) &= v_{32}(1) + \Delta v_{32}(1) = 0.4 - 0.0068 = 0.3932
\end{aligned}$$

In [18]:

```
#hitung delta bobot antara hidden dg output
sigmaOk=np.zeros((1,2))
deltaV=np.zeros((4,2))
target=np.array([[0,0]])
learningRate=0.05

sigmaOk[0,0]=(target[0,0]-o[0,0])*o[0,0]*(1-o[0,0])
sigmaOk[0,1]=(target[0,1]-o[0,1])*o[0,1]*(1-o[0,1])

print('sigmaOk=\n',sigmaOk)

for i in range(4):
    for j in range(2):
        deltaV[i,j]=learningRate*sigmaOk[0,j]*dataHidden[0,i]
#update bobot antara hidden dg output
print('deltaV=\n',deltaV)
v=v+deltaV
print('updated v=\n', v)
```

```
sigmaOk=
 [[-0.1474489  -0.14508583]]
deltaV=
 [[-0.00737245 -0.00725429]
 [-0.00625278 -0.00615257]
 [-0.0070695  -0.0069562 ]
 [-0.00690818 -0.00679746]]
updated v=
 [[0.29262755 0.19274571]
 [0.04374722 0.09384743]
 [0.0929305  0.2930438 ]
 [0.39309182 0.39320254]]
```

Update bobot antara input dan hidden layer

$$\Delta w_{ij} = \eta(-\frac{\partial E}{\partial w_{ij}})$$

dimana,

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial w_i j}$$

derivasi E berdasarkan y_j :

$$\frac{\partial E}{\partial y_j} = \frac{\partial E}{\partial O_k}\frac{\partial O_k}{\partial net_{Ok}}\frac{\partial net_{Ok}}{\partial y_j}$$

dimana, $\frac{\partial E}{\partial O_k} = -(t_k - O_{kp})$, dan $\frac{\partial O_k}{\partial net_{Ok}} = O_k(1 - O_k) = f'_{Ok}$, sehingga :

$$\begin{aligned}\frac{\partial E}{\partial O_k}\frac{\partial O_k}{\partial net_{Ok}} &= -(t_k - O_{kp})f'_{Ok} \\ &= -\delta O_k\end{aligned}$$

Derivasi net_{Ok} berdasarkan y_j :

$$\begin{aligned}\frac{\partial net_{Ok}}{\partial y_j} &= \frac{\partial\big(\sum_{j=0}^{J} v_{jk}y_j\big)}{\partial y_j} \\ &= v_{jk}\end{aligned}$$

Sehingga,

$$\begin{aligned}\frac{\partial E}{\partial y_j} &= \frac{\partial E}{\partial O_k}\frac{\partial O_k}{\partial net_{Ok}}\frac{\partial net_{Ok}}{\partial y_j} \\ &= -\delta Ok v_{jk}\end{aligned}$$

Derivasi y berdasarkan w

$$\begin{aligned}\frac{\partial y_j}{\partial w_i j} &= \frac{\partial y_j}{\partial net_{yj}}\frac{\partial net_{yj}}{\partial w_{ij}} \\ &= y_j(1-y_j)x_i \\ &= f'_{yj}x_i\end{aligned}$$

sehingga,

$$\begin{aligned}\Delta w_{ij} &= \eta\big(-\frac{\partial E}{\partial w_{ij}}\big) \\ &= -\eta\frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial w_i j} \\ &= -\eta - \delta Ok v_{jk} f'_{yj} x_i \\ &= \eta\delta_{yj}x_i\end{aligned}$$

Hidden layer dan input layer

Hitung *delta weight* antara input dan hidden layer (Δw) , :

$$
\begin{aligned}
\delta yj &= \delta O_k v_{jk}(y_j(1-y_j)) \\
\delta y_1 &= (\delta O_1 v_{11} + \delta O_2 v_{12})[y_1(1-y_1)] \\
&= (-0.148 \times 0.05 + (-0.145) \times 0.1)[0.85(1-0.85)] = -0.0028 \\
\delta y_2 &= (\delta O_1 v_{21} + \delta O_2 v_{22})[y_2(1-y_2)] \\
&= (-0.148 \times 0.1 + (-0.145) \times 0.3)[0.96(1-0.96)] = -0.0022 \\
\delta y_3 &= (\delta O_1 v_{31} + \delta O_2 v_{12})[y_3(1-y_3)] \\
&= (-0.148 \times 0.4 + (-0.145) \times 0.4)[0.94(1-0.94)] = -0.0066
\end{aligned}
$$

$$
\begin{aligned}
\Delta w_{ij} &= \eta \delta yj x_i \\
\Delta w_{01} &= 0.05 \delta y_1 x_0 = 0.05 \times -0.0028 \times 1 = -0.00014 \\
\Delta w_{02} &= 0.05 \delta y_2 x_0 = 0.05 \times -0.0022 \times 1 = -0.00011 \\
\Delta w_{03} &= 0.05 \delta y_3 x_0 = 0.05 \times -0.0066 \times 1 = -0.00033 \\
\Delta w_{11} &= 0.05 \delta y_1 x_1 = 0.05 \times -0.0028 \times 5.1 = -0.000714 \\
\Delta w_{12} &= 0.05 \delta y_2 x_1 = 0.05 \times -0.0022 \times 5.1 = -0.0000561 \\
\Delta w_{13} &= 0.05 \delta y_3 x_1 = 0.05 \times -0.0066 \times 5.1 = -0.001683 \\
\Delta w_{21} &= 0.05 \delta y_1 x_2 = 0.05 \times -0.0028 \times 3.5 = -0.00049 \\
\Delta w_{22} &= 0.05 \delta y_2 x_2 = 0.05 \times -0.0022 \times 3.5 = -0.000385 \\
\Delta w_{23} &= 0.05 \delta y_3 x_3 = 0.05 \times -0.0066 \times 3.5 = -0.001155 \\
\Delta w_{31} &= 0.05 \delta y_1 x_3 = 0.05 \times -0.0028 \times 1.4 = -0.000196 \\
\Delta w_{32} &= 0.05 \delta y_2 x_3 = 0.05 \times -0.0022 \times 1.4 = -0.000154 \\
\Delta w_{33} &= 0.05 \delta y_3 x_3 = 0.05 \times -0.0066 \times 1.4 = -0.000462 \\
\Delta w_{41} &= 0.05 \delta y_1 x_4 = 0.05 \times -0.0028 \times 0.2 = -0.000028 \\
\Delta w_{42} &= 0.05 \delta y_2 x_4 = 0.05 \times -0.0022 \times 0.2 = -0.000022 \\
\Delta w_{43} &= 0.05 \delta y_3 x_4 = 0.05 \times -0.0066 \times 0.2 = -0.000066
\end{aligned}
$$

update bobot antara hidden dan input

$$
\begin{aligned}
w_{ij}(2) &= w_{ij}(1) + \Delta w_{ij}(1) \\
w_{01}(2) &= w_{01}(1) + \Delta w_{01}(1) = 0.1 - 0.00014 = 0.09986 \\
w_{02}(2) &= w_{02}(1) + \Delta w_{02}(1) = 0.3 - 0.00011 = 0.29989 \\
w_{03}(2) &= w_{03}(1) + \Delta w_{03}(1) = 0.4 - 0.00033 = 0.39967 \\
w_{11}(2) &= w_{11}(1) + \Delta w_{11}(1) = 0.2 - 0.000714 = 0.199286 \\
w_{12}(2) &= w_{12}(1) + \Delta w_{12}(1) = 0.4 - 0.0000561 = 0.3999439 \\
w_{13}(2) &= w_{13}(1) + \Delta w_{13}(1) = 0.15 - 0.001683 = 0.148317 \\
w_{21}(2) &= w_{21}(1) + \Delta w_{21}(1) = 0.1 - 0.00049 = 0.09951 \\
w_{22}(2) &= w_{22}(1) + \Delta w_{22}(1) = 0.1 - 0.000385 = 0.099615 \\
w_{23}(2) &= w_{23}(1) + \Delta w_{23}(1) = 0.25 - 0.001155 = 0.248845 \\
w_{31}(2) &= w_{31}(1) + \Delta w_{31}(1) = 0.15 - 0.000196 = 0.149804 \\
w_{32}(2) &= w_{32}(1) + \Delta w_{32}(1) = 0.3 - 0.000154 = 0.299846 \\
w_{33}(2) &= w_{33}(1) + \Delta w_{33}(1) = 0.4 - 0.000462 = 0.399538 \\
w_{41}(2) &= w_{41}(1) + \Delta w_{41}(1) = 0.2 - 0.000028 = 0.199972 \\
w_{42}(2) &= w_{42}(1) + \Delta w_{42}(1) = 0.2 - 0.000022 = 0.199978 \\
w_{43}(2) &= w_{43}(1) + \Delta w_{43}(1) = 0.5 - 0.000066 = 0.499934
\end{aligned}
$$

In [19]:

```
#hitung delta bobot antara input dg hidden
#hitung deltaNetY

tempV=v[1:4,:]
print(v.shape)

tempV=tempV.T
sigmaY=sigmaOk.dot(tempV)*(y*(1-y))


deltaW=np.zeros((5,3))




for i in range (5):
    for j in range (3):
        deltaW[i,j]=learningRate*sigmaY[0,j]*inp[i]
#update bobot antara input dengan hidden

print('deltaW=\n',deltaW)
w=w+deltaW
print('w=\n',w)
```

```
(4, 2)
deltaW=
 [[-1.29234021e-04 -1.10759264e-04 -3.39321120e-04]
 [-6.59093507e-04 -5.64872249e-04 -1.73053771e-03]
 [-4.52319073e-04 -3.87657426e-04 -1.18762392e-03]
 [-1.80927629e-04 -1.55062970e-04 -4.75049569e-04]
 [-2.58468042e-05 -2.21518529e-05 -6.78642241e-05]]
w=
 [[0.09987077 0.29988924 0.39966068]
 [0.19934091 0.39943513 0.14826946]
 [0.09954768 0.09961234 0.24881238]
 [0.14981907 0.29984494 0.39952495]
 [0.19997415 0.19997785 0.49993214]]
```

Algoritma :

- Inisialisasi bobot secara acak untuk seluruh layer, *learning rate* (η), dan epoch=0.
- while stopping_condition is not True:
  - $\varepsilon_T = 0$
  - untuk setiap data training p:
    - feedforward : $y_{j,p}(\forall j = 0, \ldots, J)$; $O_{k,p}(\forall k = 1, \ldots, K)$
    - Hitung error δ_{ok} dan δ_{yj}
    - backward propagation, update bobot v_{jk} dan w_{ij}
    - Hitung $\varepsilon_T + = [\varepsilon_p = \sum_{k=1}^{K} (t_{k,p} - O_{k,p})^2]$
  - t=t+1

Stopping condition :

- epoch
- ε_T

Referensi