Modul Praktikum Artificial Neural Network

Kita akan menerapkan jaring saraf kita dengan lapisan tersembunyi langkah demi langkah dengan Python, mari kita kode:

Import required libraries

In [1]:
```
import numpy as np
```

Tentukan fitur masukan

Selanjutnya, kita mengambil nilai input yang ingin kita latih jaringan saraf kita. Kita dapat melihat bahwa kita telah mengambil dua fitur input. Pada kumpulan data nyata, nilai fitur input sebagian besar tinggi.

In [2]:
```
input_features = np.array([[0,0],[0,1],[1,0],[1,1]])
print (input_features.shape)
input_features
```

```
(4, 2)
```

Out[2]:
```
array([[0, 0],
       [0, 1],
       [1, 0],
       [1, 1]])
```

Tentukan nilai keluaran target

Untuk fitur input, kita ingin memiliki output khusus untuk fitur input tertentu. Ini disebut keluaran sasaran. Kita akan melatih model yang memberi kita output target untuk fitur input.

In [3]:
```
target_output = np.array([[0,1,1,1]])

target_output = target_output.reshape(4,1)

print(target_output.shape)

target_output
```

```
(4, 1)
```

Out[3]:
```
array([[0],
       [1],
       [1],
       [1]])
```

Tetapkan bobot acak

Selanjutnya, kita akan menetapkan bobot acak ke fitur input. Perhatikan bahwa model kita akan memodifikasi nilai bobot ini menjadi optimal. Pada titik ini, kita mengambil nilai-nilai ini secara acak. Di sini kita memiliki dua lapisan, jadi kita harus menetapkan bobot untuk mereka secara terpisah.

Variabel lainnya adalah learning rate. Kita akan menggunakan tingkat pembelajaran (LR) dalam algoritma penurunan gradien untuk memperbarui nilai bobot. Umumnya, kita menjaga LR serendah mungkin sehingga kita dapat mencapai error rate minimal.

In [4]:
```
#Tentukan berat:
#6 untuk lapisan tersembunyi
#3 untuk lapisan keluaran
#9 total

weight_hidden = np.array([[0.1,0.2,0.3],
                          [0.4,0.5,0.6]])

weight_output = np.array([[0.7],[0.8],[0.9]])

#Learning Rate :
lr = 0.05
```

Fungsi sigmoid

Setelah kita memiliki nilai bobot dan fitur input, kita akan mengirimkannya ke fungsi utama yang memprediksi output. Perhatikan bahwa fitur input dan nilai bobot kita bisa apa saja, tapi di sini kita ingin mengklasifikasikan data, jadi kita membutuhkan output antara 0 dan 1. Untuk output seperti itu, kita akan menggunakan fungsi sigmoid.

In [5]:
```
def sigmoid(x):
    return 1/(1+np.exp(-x))
```

Fungsi sigmoid derivative

Dalam algoritma penurunan gradien, kita membutuhkan turunan dari fungsi sigmoid.

In [6]:
```
def sigmoid_der(x):
    return sigmoid(x)*(1-sigmoid(x))
```

Logika utama untuk memprediksi keluaran dan memperbarui nilai bobot

Kita akan memahami kode berikut langkah demi langkah.

In [9]:
```
for epoch in range(200000):
    #Input for hidden layer:
    input_hidden = np.dot(input_features, weight_hidden)

    #Output from hidden layer:
    output_hidden = sigmoid(input_hidden)

    #Input for output layer:
    input_op = np.dot(output_hidden, weight_output)

    #Output from output layer:
    output_op = sigmoid(input_op)

    #Phase 1

    #Menghitung Mean Squared Error :
    error_out = ((1/2)*(np.power((output_op-target_output),2)))
    print(error_out.sum())

    #Derivatives for phase 1 :
    derror_douto = output_op - target_output
    douto_dino = sigmoid_der(input_op)
    dino_dwo = output_hidden

    derror_dwo = np.dot(dino_dwo.T, derror_douto*douto_dino)


    #Phase 2

    #derror_w1 = derror_douth*douth_dinh*dinh_dw1
    #derror_douth = derror_dino*dino_outh

    #Derivatives for phase 2 :
    derror_dino = derror_douto*douto_dino
    dino_douth = weight_output
    derror_douth = np.dot(derror_dino, dino_douth.T)
    douth_dinh = sigmoid_der(input_hidden)
    dinh_dwh = input_features
    derror_wh = np.dot(dinh_dwh.T, douth_dinh*derror_douth)

    #Update Weights
    weight_hidden -= lr*derror_wh
    weight_output -= lr*derror_dwo
```

```
0.3404001841459817
0.3403562662703295
0.34031247662694225
0.34026881418213206
0.34022527791082136
0.3401818667965103
0.34013857983124396
0.3400954160155779
0.34005237435854424
0.3400094538776164
0.33996665359867234
0.3399239725559594
0.33988140979205617
0.33983896435783534
0.33979663531242466
0.3397544217231687
0.3397123226655893
0.33967033722334516
0.33962846448819184
```

Nilai bobot akhir

Di bawah ini, kita menunjukkan nilai bobot yang diperbarui untuk kedua lapisan — prediksi kita didasarkan pada nilai-nilai ini.

In [10]:
```
#Final hidden layer weight values :

print(weight_hidden)
```

```
[[-3.82266736  2.36350436  3.13872805]
 [-3.79487903  2.45206336  3.18546752]]
```

In [11]:
```
#Final output layer weight values :

print(weight_output)
```

```
[[-13.22687155]
 [  1.74413064]
 [  3.50403463]]
```

Membuat prediksi

Prediction for (1,1). Target output = 1

Pertama-tama, kita akan mengambil nilai input yang ingin kita prediksi outputnya. Variabel "result1" menyimpan nilai produk titik dari variabel input dan bobot lapisan tersembunyi. Kita memperoleh output dengan menerapkan fungsi sigmoid, hasilnya disimpan dalam variabel result2. Itulah fitur input untuk lapisan output. Kita menghitung input untuk lapisan output dengan mengalikan fitur input dengan bobot lapisan output. Untuk mencari nilai keluaran akhir, kita ambil nilai sigmoidnya.

In [12]:
```
#Taking inputs :
single_point = np.array([1,1])

#1st step :
result1 = np.dot(single_point, weight_hidden)

#2nd step :
result2 = sigmoid(result1)

#Print final result
#print(result2)

result3 = np.dot(result2,weight_output)
result4 = sigmoid(result3)
print(result4)
```

```
[0.99462911]
```

Perhatikan bahwa output yang diprediksi sangat dekat dengan 1. Jadi kami telah berhasil membuat prediksi yang akurat.

Prediction for (0,0). Target output = 0

In [13]:
```
#Taking inputs :
single_point = np.array([0,0])

#1st step :
result1 = np.dot(single_point, weight_hidden)

#2nd step :
result2 = sigmoid(result1)

#Print final result
#print(result2)

result3 = np.dot(result2,weight_output)
result4 = sigmoid(result3)
print(result4)
```

```
[0.01817523]
```

Perhatikan bahwa output yang diprediksi sangat mendekati 0, yang menunjukkan tingkat keberhasilan model.