MODUL PRAKTIKUM KECERDASAN BUATAN

PERTEMUAN 6 – JARINGAN SYARAF TIRUAN

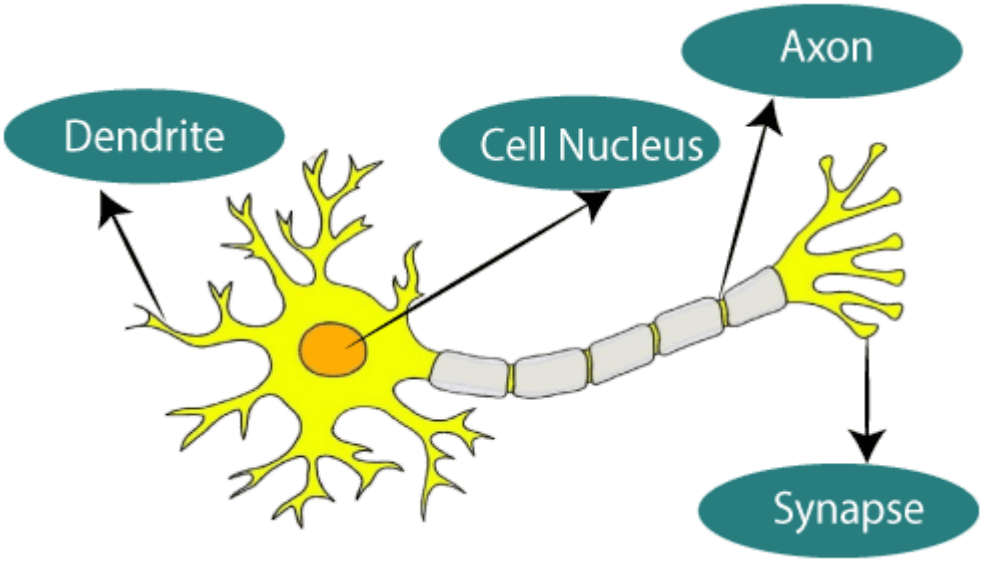
Tools : Jupyter notebook

Bahasa Pemrograman : Python

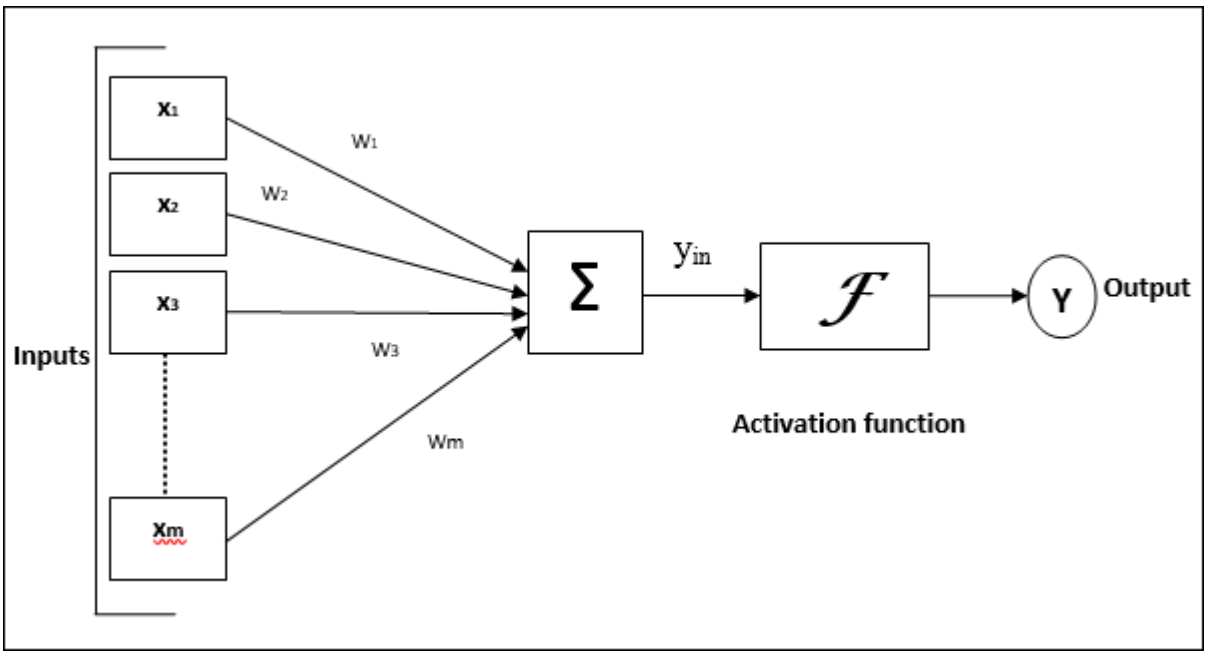
Dalam praktikum kali ini, kita akan belajar tentang bagaimana cara membangun Jaringan Syaraf Tiruan menggunakan Python.

Jaringan Syaraf Tiruan

Jaringan syaraf tiruan merupakan bagian dari kecerdasan buatan yang mengadopsi jaringan syaraf pada otak manusia (jaringan syaraf biologi). Mirip dengan otak manusia yang memiliki neuron-neuron yang saling terhubung satu sama lain, jaringan syaraf tiruan juga memiliki neuron-neuron yang saling terhubung satu sama lain di berbagai lapisan jaringan. Neuron dalam jaringan syaraf tiruan dikenal sebagai node. Adapun desain jaringan syaraf manusia dan jaringan syaraf tiruan dapat dilihat pada Gambar 6.1 dan Gambar 6.2.



Gambar 6.1. Jaringan syaraf biologi (https://www.javatpoint.com/artificial-neural-network)



Gambar 6.2. Jaringan syaraf tiruan

(https://www.tutorialspoint.com/artificial_neural_network/artificial_neural_network_basic_concepts.htm)

Hubungan antara jaringan syaraf tiruan dan jaringan syaraf biologi dapat dilihat pada Tabel 6.1.

Tabel 6.1. Hubungan kesamaan antara jaringan syaraf tiruan dengan jaringan syaraf biologi

Biological Neural Network	Artificial Neural Network
Dendrites	Inputs
Cell nucleus	Nodes
Synapse	Weights
Axon	Output

Arsitektur Jaringan Syaraf Tiruan

Secara garis besar, jaringan syaraf tiruan terdiri atas tiga lapisan:

1. Lapisan input (input layer)

   Lapisan input berfungsi untuk menerima berbagai format input dari programmer.

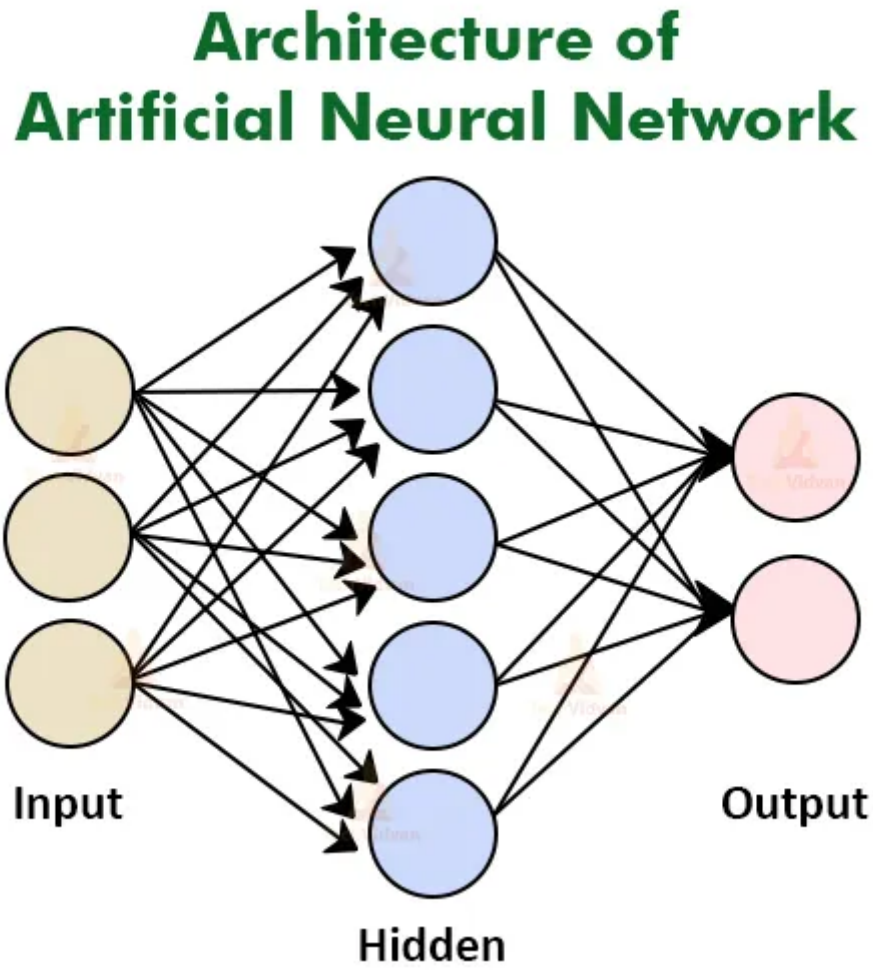
2. Lapisan tersembunyi (hidden layer)

   Lapisan tersembunyi merupakan lapisan yang berada diantara lapisan input dan output. Lapisan ini berfungsi untuk menemukan fitur dan pola tersembunyi.melalui sejumlah perhitungan.

3. Lapisan output (output layer)

   Lapisan output berfungsi untuk mendapatkan output dari serangkaian proyes yang terjadi pada lapisan sebelumnya.

Arsitektur jaringan syaraf tiruan digambarkan dalam Gambar 6.3.



Gambar 6.3. Arsitektur jaringan syaraf tiruan
(https://blog.knoldus.com/architecture-of-artificial-neural-network/)

Penerapan Jaringan Syaraf Tiruan

Pada sesi kali ini, kita akan membuat jaringan syaraf tiruan yang sangat sederhana dengan satu lapisan *input* dan satu lapisan *output* dari awal menggunakan pustaka `numpy` pada Python. Jaringan syaraf tiruan ini mampu mengklasifikasikan data yang dapat dipisahkan secara linear. Langkah-langkah yang perlu dilakukan antara lain:

1. Mendefinisikan variabel bebas (*independent variables*) dan variabel terikat (*dependent variable*)
2. Mendefinisikan nilai dari *hyperparameter*
3. Menentukan fungsi aktivasi dan turunannya
4. Melatih model

5. Membuat prediksi

Langkah 1: Mendefinisikan variabel bebas (independent variables) dan variabel terikat (dependent variable)

```
In [2]: #Define independent variables and dependent variable
        import numpy as np

        input_set = np.array([[0,1,0],
                              [0,0,1],
                              [1,0,0],
                              [1,1,0],
                              [1,1,1],
                              [0,1,1],
                              [0,1,0]])#Dependent variable
        labels = np.array([[1,
                            0,
                            0,
                            1,
                            1,
                            0,
                            1]])
        labels = labels.reshape(7,1) #to convert labels to vector
```

Input terdiri dari tujuh *record* sehingga didefinisikan pula nilai label atau target dari ketujuh *reord* tersebut yang nantinya akan diprediksi. Hal tersebut dapat digambarkan pad Tabel 6.2.

Tabel 6.2 Nilai inut dan nilai label

No	Input	Label
1	0,1,0	1
2	0,0,1	0
3	1,0,0	0
4	1,1,0	1
5	1,1,1	1
6	0,1,1	0
7	0,1,0	1

Langkah 2: Mendefinisikan nilai dari hyperparameter

Dalam praktikum ini, kita akan menggunakan fungsi `random.seed` dari `numpy` untuk mendapatkan nilai acak. Selanjutnya, kita menginisialisasi bobot dengan angka acak yang terdistribusi

normal. Karena kita memiliki tiga fitur di input dan vektor dengan tiga bobot sehingga kita menginisialisasi nilai bias dengan nomor acak lain. Terakhir, kita menetapkan kecepatan pembelajaran ke 0,05.

```
In [3]: #Define Hyperparameters
        np.random.seed(42)
        weights = np.random.rand(3,1)
        bias = np.random.rand(1)
        lr = 0.05 #learning rate
```

Langkah 3: Mendefinisikan fungsi aktivasi dan turunannya

Fungsi aktivasi yang kita gunakan adalah fungsi sigmoid.

```
In [4]: #Define Activation Function and it's derivative: Our activation function is the sigmoid function.
        def sigmoid(x):
            return 1/(1+np.exp(-x))
```

```
In [5]: #Define a function that calculates the derivative of the sigmoid function.
        def sigmoid_derivative(x):
            return sigmoid(x)*(1-sigmoid(x))
```

Langkah 4: Melatih model

Pada tahap ini, kita akan mendefinisikan nilai *epoch*. *Epoch* menjelaskan berapa kali kita ingin melatih algoritma pada kumpulan data. Kita akan melatih algoritma sebanyak 25.000 kali sehingga nilai *epoch* menjadi 25.000. Nilai *epoch* tersebut bisa diatur sesuai kebutuhan.

```
In [6]: #Train our ANN model
        for epoch in range(25000):
            inputs = input_set
            XW = np.dot(inputs, weights)+ bias
            z = sigmoid(XW)
            error = z - labels
            print(error.sum())
            dcost = error
            dpred = sigmoid_derivative(z)
            z_del = dcost * dpred
            inputs = input_set.T
            weights = weights - lr*np.dot(inputs, z_del)

            for num in z_del:
                bias = bias - lr*num
```

Output

```
1.8806216715619812
1.8525640899325237
1.8243107340899896
1.7958847686438375
1.7673099081827413
1.7386103353186555
1.70981061515949
1.6809356068569574
1.6520103729182125
1.623060087002853
1.5941099409498616
1.5651850517915373
1.5363103695131097
1.5075105863073248
1.4788100480531448
1.4502326687170073
1.4218018483346397
1.3935403951819836
1.3654704526863837
```

Langkah 5: Membuat prediksi

Pada tahap ini, kita akan melakukan prediksi. Kita mencoba untuk melakukan prediksi label atas nilai input [1,0,0] .

```
In [7]: #Make predictions
        single_pt = np.array([1,0,0])
        result = sigmoid(np.dot(single_pt, weights) + bias)
        print(result)
```

Output

```
[0.02262959]
```

Nilai prediksi [0.02262959] menunjukkan bahwa nilai tersbut mendekati 0 daripada 1, sehingga nilai [1,0,0] diklasifikasikan sebagai 0.

Selanjutnya kita akan mencoba melakukan prediksi label atas input [0,1,0] .

```
In [9]: single_pt = np.array([0,1,0])
        result = sigmoid(np.dot(single_pt, weights) + bias)
        print(result)
```

Output

```
[0.98778682]
```

Nilai prediksi [0.98778682] menunjukkan bahwa nilai tersbut mendekati 1 daripada 0, sehingga nilai [0,1,0] diklasifikasikan sebagai 1.

Latihan: Uji Coba Menggunakan Berbagai Jenis Fungsi Aktivasi (*Membership Function*)

1. Gunakan fungsi aktivasi linear dan hyperbolic tangent untuk data yang sama dan tunjukkan hasilnya!
2. Apakah terjadi perbedaan hasil pelatihan dan prediksi pada saat menggunakan fungsi aktivasi yang berbeda? Tunjukkan dan jelaskan! (dapat menggunakan tabel untuk menunjukkan nilai perbedaannya)
3. Gunakan variasi *epoch* (2500, 3500, 4500) dan tunjukkan hasil pelatihan dan hasil prediksinya! (dapat menggunakan tabel untuk menunjukkan nilai perbedaannya)

Referensi:

https://www.kdnuggets.com/2019/11/build-artificial-neural-network-scratch-part-1.html
https://www.geeksforgeeks.org/implementation-of-neural-network-from-scratch-using-numpy/
https://www.javatpoint.com/artificial-neural-network
https://www.tutorialspoint.com/artificial_neural_network/artificial_neural_network_basic_concepts.htm
https://www.ibm.com/topics/neural-networks