Praktikum 1
Konsep JST

Tujuan

Setelah mengikuti praktikum ini, mahasiswa diharapkan mampu:

1. Memahami konsep jaringan saraf tiruan;
2. Mengimplementasikan jaringan saraf tiruan McCulloch-Pitts; dan
3. Mengimplementasikan perhitungan akurasi.

Dasar Teori

Jaringan Saraf Tiruan

Jaringan saraf tiruan adalah sistem pemroses informasi yang memiliki kemiripan dengan jaringan saraf biologis yang ada pada makhluk hidup. Jaringan saraf tiruan dikembangkan sebagai model matematis dengan komponen-komponen dan mekanisme berikut.

1. Pemrosesan informasi dilakukan pada elemen-elemen jaringan saraf tiruan yang disebut dengan neuron.
2. Informasi diteruskan dari satu neuron ke neuron-neuron yang lain melalui suatu penghubung.
3. Penghubung tersebut memiliki bobot tertentu yang memengaruhi seberapa kuat informasi diteruskan.
4. Setiap neuron menggunakan suatu fungsi aktivasi yang memproses sinyal yang input menjadi sinyal *output*.

Karakteristik sebuah jaringan saraf tiruan ditentukan oleh:

1. Susunan atau arsitektur neuron-neuronnya;
2. Mekanisme (algoritma) perubahan nilai bobotnya; dan
3. Fungsi aktivasinya.

Arsitektur Jaringan Saraf Tiruan

Suatu jaringan saraf tiruan tersusun atas elemen-elemen yang disebut dengan neuron atau unit. Setiap neuron terhubung oleh suatu penghubung yang masing-masing memiliki nilai bobot. Nilai-nilai bobot tersebut

merepresentasikan informasi yang digunakan untuk menyelesaikan suatu masalah. Jaringan saraf tiruan dapat diaplikasikan untuk berbagai jenis masalah, antara lain klasifikasi, klasterisasi, dan prediksi.

Neuron-neuron pada suatu jaringan saraf tiruan disusun dan dikelompokkan secara vertikal sehingga membentuk kelompok-kelompok neuron yang disebut dengan *layer*. Neuron-neuron pada suatu *layer* terhubung dengan neuron-neuron pada *layer* berikutnya dengan suatu penghubung yang masing-masing memiliki nilai bobot. Bagaimana kita menyusun neuron-neuron menjadi *layer-layer* sehingga menjadi jaringan saraf tiruan yang utuh disebut dengan arsitektur atau topologi jaringan saraf tiruan.

Gambar 1.1 memberikan contoh jaringan saraf tiruan sederhana dengan tiga neuron X_1, X_2, dan X_3 pada *layer* pertama dan satu neuron Y pada *layer* kedua. Sinyal akan masuk ke jaringan tersebut melalui *layer* pertama dan diterima oleh neuron-neuron X_1, X_2, dan X_3. Sinyal tersebut kemudian akan diteruskan ke neuron Y dengan melewati w_1, w_2, dan w_3. Karena itu, *layer* pertama disebut juga dengan *input layer* dan *layer* kedua disebut dengan *output layer*.

Jumlah *input neuron* disesuaikan dengan panjang vektor dari data input yang digunakan. Sedangkan jumlah *output neuron* disesuaikan dengan kasus yang ingin diselesaikan. Pada kasus klasifikasi, misalnya, jumlah *output neuron* adalah sama dengan jumlah kelasnya. Pada kasus prediksi dengan satu nilai prediksi, *output neuron* berjumlah satu.

Jika sinyal yang masuk ke neuron-neuron X_1, X_2, ..., X_n adalah x_1, x_2, ..., x_n, maka perhitungan yang dilakukan disebut dengan *weighted sum* dan akan menghasilkan y_in:

$$y\_in = \sum_{i=1}^{n} w_i x_i$$

Sehingga perhitungan yang dilakukan pada jaringan saraf tiruan pada Gambar 1.1 adalah:

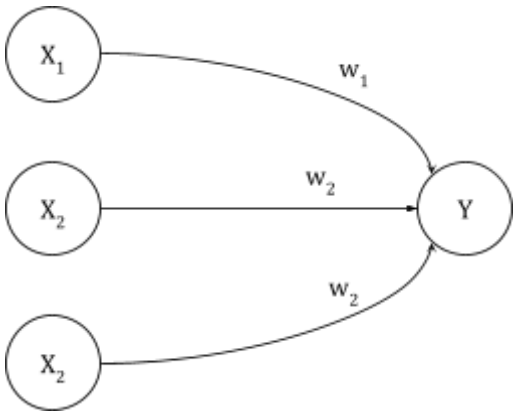
$$y\_in = w_1 x_1 + w_2 x_2 + w_3 x_3$$

Dengan perhitungan tersebut, jika x_1, x_2, ..., x_n bernilai sama, nilai yang dihasilkan oleh ketiga neuron dapat berbeda jika w_1, w, ..., w_n memiliki nilai

yang berbeda-beda. Nilai y_in kemudian melewati suatu fungsi aktivasi dan menghasilkan nilai y yang merupakan *output* dari jaringan saraf tiruan.

Nilai-nilai x_1, x_2, ..., x_n berasal dari data, sedangkan nilai-nilai w_1, w, ..., w_n dapat kita tentukan sendiri atau dapat berasal dari suatu algoritma *learning*, seperti Perceptron dan *backpropagation*.

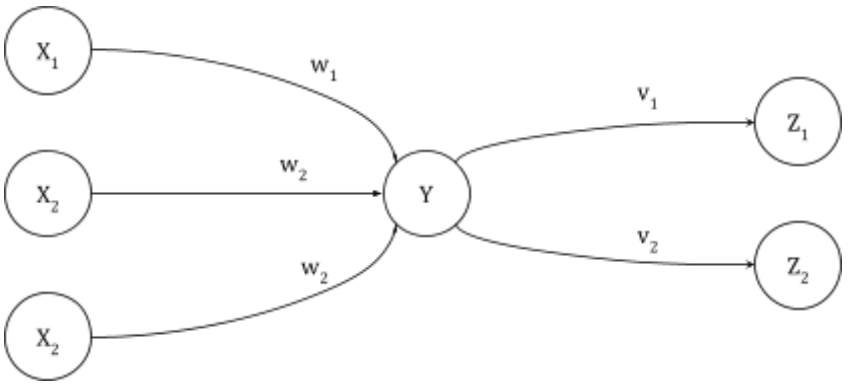
Input layer, pada umumnya, tidak dianggap sebagai sebuah *layer* karena pada *input neuron* tidak dilakukan proses aktivasi. Nilai yang masuk ke input neuron juga merupakan nilai aktivasinya. Sehingga arsitektur jaringan saraf tiruan pada Gambar 1.1 disebut dengan arsitektur *single layer*.



Gambar 1.1 Jaringan saraf tiruan sederhana

Jika kita menambahkan *layer* pada jaringan saraf tiruan di atas, maka hasilnya adalah suatu jaringan saraf tiruan yang dapat menyelesaikan masalah yang lebih kompleks. Gambar 1.2 memberikan contoh jaringan saraf tiruan dengan tambahan satu *layer*. *Layer-layer* yang berada di antara *input layer* dan *output layer* disebut dengan *hidden layer* karena *layer* tersebut "tersembunyi dari dunia luar". Jika suatu jaringan saraf tiruan memiliki satu atau lebih *hidden layer*, maka arsitekturnya disebut dengan *multi layer*.

Pada prinsipnya, semakin banyak *hidden layer* pada suatu jaringan saraf tiruan, maka jaringan saraf tiruan tersebut akan dapat menyelesaikan masalah yang semakin kompleks. Begitu pula dengan jumlah neuron pada *hidden layer*. Semakin banyak jumlah neuron pada *hidden layer* berarti semakin banyak variabel yang tersedia pada jaringan saraf tiruan sehingga kapasitas jaringan saraf tiruan menjadi semakin besar untuk menyelesaikan masalah yang kompleks. Namun hal itu tentu membuat komputasi menjadi lebih lama dan juga berpotensi mengakibatkan terjadinya kondisi *overfitting* jika arsitektur yang digunakan terlalu kompleks.



Gambar 1.2 Jaringan saraf tiruan dengan satu *hidden layer*

Fungsi Aktivasi

Fungsi aktivasi melakukan suatu perhitungan terhadap sinyal input yang masuk ke suatu neuron, menghasilkan nilai aktivasi dari neuron tersebut. Pada umumnya, suatu jaringan saraf tiruan menggunakan fungsi aktivasi yang sama pada semua neuron-neuronnya. Ada beberapa fungsi aktivasi yang dapat digunakan.

Pemilihan fungsi aktivasi dapat memengaruhi kemampuan jaringan saraf tiruan dalam menyelesaikan masalah. Untuk kasus-kasus sederhana, fungsi aktivasi linear dapat digunakan dan akan dapat menyelesaikan masalah dengan baik. Namun, untuk kasus-kasus yang lebih kompleks dengan data yang sifatnya non-linear, kita perlu memberikan sifat *nonlinearity* pada jaringan saraf tiruan dengan menggunakan fungsi aktivasi yang non-linear.

1. Fungsi identitas

   Fungsi ini memberikan hasil yang sama dengan inputnya tanpa melakukan perhitungan apapun. Sehingga, pada esensinya, ini sama dengan tidak menggunakan fungsi aktivasi sama sekali. Fungsi ini memiliki bentuk:

   $$f(x) = x$$

2. Fungsi *step* biner

   Fungsi ini menghasilkan nilai aktivasi berdasarkan perbandingan nilai input terhadap nilai batas (*threshold*) tertentu yang disimbolkan dengan θ. Nilai yang dihasilkan adalah 0 atau 1 (bulat, tidak ada nilai di antaranya) berdasarkan perbandingan tersebut. Fungsi ini termasuk fungsi linear dan memiliki bentuk:

   $$f(x) = \begin{cases} 1 & if x \geq \theta \\ 0 & if x < \theta \end{cases}$$

3. Fungsi *step* bipolar

Fungsi ini mirip dengan fungsi step biner dengan perbedaan pada nilai *output*-nya yaitu -1 atau 1 (bulat, tidak ada nilai di antaranya). Persamaan dari fungsi ini adalah:

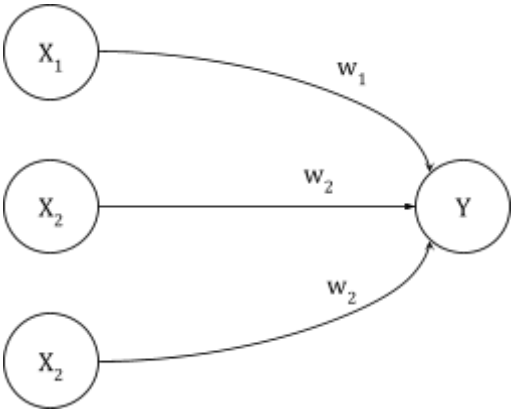
$$f(x) = \begin{cases} 1 & if x \geq \theta \\ -1 & if x < \theta \end{cases}$$

4. Fungsi *sigmoid* biner
   Fungsi ini merupakan fungsi non-linear dengan nilai aktivasi berupa bilangan pecahan antara 0 dan 1. Fungsi ini disebut juga dengan fungsi *logistic*. Bentuk dari fungsi ini adalah:
   $$f(x) = \frac{1}{1+exp(-\sigma x)}$$
5. Fungsi *sigmoid* bipolar
   Fungsi ini mirip dengan fungsi sigmoid biner tetapi dengan nilai *output* antara -1 dan 1. Bentuk dari fungsi ini adalah:
   $$f(x) = \frac{2}{1+exp(-\sigma x)} - 1$$
6. Fungsi tangen hiperbolik (tanh)
   Fungsi ini dapat menjadi alternatif dari fungsi sigmoid. Pada beberapa kasus, fungsi ini lebih baik daripada fungsi sigmoid. *Output* dari fungsi ini adalah antara -1 dan 1. Fungsi ini memiliki bentuk:
   $$f(x) = \frac{e^{2x}-1}{e^{2x}+1}$$
7. Fungsi ReLU (*rectified linear unit*)
   Fungsi ini memiliki kemampuan yang hampir sama baiknya dengan fungsi non-linear tetapi dengan komputasi yang lebih sederhana. Sehingga fungsi ini cukup banyak digunakan pada *deep learning* yang melibatkan banyak sekali neuron. Bentuk dari fungsi ini adalah:
   $$f(x) = max(0, x)$$

McCulloch-Pitts Neuron

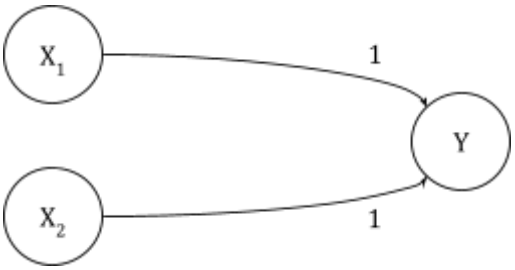
McCulloch-Pitts neuron diperkirakan merupakan jaringan saraf tiruan yang pertama kali dibuat. Dirancang oleh McCulloch dan Pitts (1943), jaringan saraf tiruan ini memperkenalkan beberapa aspek penting. Jaringan saraf tiruan ini menggunakan fungsi aktivasi *step* biner dan tidak memiliki mekanisme *learning*. Oleh karena itu, nilai-nilai bobotnya harus ditentukan secara manual. Begitu pula dengan nilai *threshold*-nya yang juga harus ditentukan secara manual.

Gambar 1.3 memberikan satu contoh McCulloch-Pitts neuron dengan arsitektur *single layer*. McCulloch-Pitts neuron juga dapat dibuat dengan arsitektur *multi layer*.



Gambar 1.3 Arsitektur McCulloch-Pitts neuron *single layer*

Sebagai contoh kasus, klasifikasi biner (dua kelas, *true* dan *false*) dengan data logika AND dapat diselesaikan dengan arsitektur dua *input neuron* dan satu *output neuron*, seperti diilustrasikan pada Gambar 1.4. Karena data input memiliki panjang vektor 2, maka *input neuron*-nya juga berjumlah 2. Kemudian karena kasusnya adalah klasifikasi biner, maka *output neuron*-nya berjumlah 1 yang dapat bernilai 1 atau 0.



Gambar 1.4 Arsitektur McCulloch-Pitts neuron untuk data logika AND

Untuk membentuk logika AND, nilai bobot yang harus digunakan adalah $w_1 = 1$ dan $w_2 = 1$. Sehingga, untuk data input $x_1 = 1$ dan $x_2 = 1$, maka:

$$\begin{aligned} y\_in &= \sum_{i=1}^{n} w_i x_i \\ &= 1 \cdot 1 + 1 \cdot 1 \\ &= 2 \end{aligned}$$

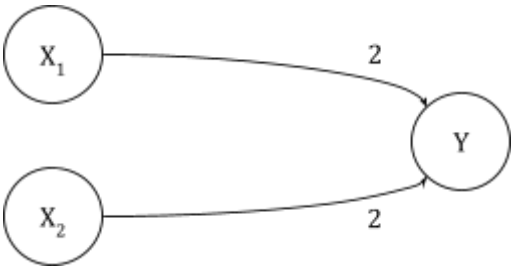
Nilai *threshold* (θ) untuk fungsi aktivasi *step* biner yang harus digunakan untuk membentuk logika AND adalah 2. Karena $y\_in = 2$ dan $\theta = 2$, maka

$y\_in \geq \theta$ sehingga $y = 1$. Dengan perhitungan yang sama, tabel hasil perhitungan untuk semua baris data adalah seperti pada Tabel 1.1.

Tabel 1.1 Perhitungan data logika AND

x_1	x_2	y_in	y
1	1	2	1
1	0	1	0
0	1	1	0
0	0	0	0

Untuk logika OR dan AND NOT, arsitektur McCulloch-Pitts neuron yang digunakan sama seperti logika AND, yaitu dengan dua *input neuron* dan satu *output neuron*. Yang membedakan hanya nilai-nilai bobot dan *threshold*-nya. Untuk logika OR, nilai-nilai bobot yang digunakan adalah 2 untuk masing-masing w_1 dan $w_2$dan nilai *threshold* 2 seperti dijelaskan pada Gambar 1.5 sehingga hasil perhitungannya ada pada Tabel 1.2.

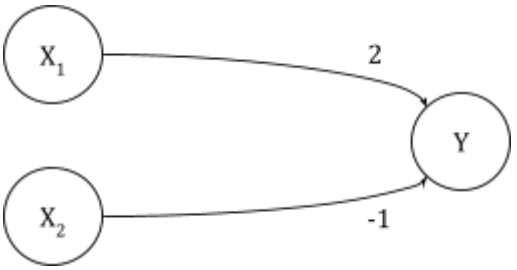


Gambar 1.5 Arsitektur McCulloch-Pitts neuron untuk data logika OR

Tabel 1.2 Perhitungan data logika OR

x_1	x_2	y_in	y
1	1	4	1
1	0	2	1
0	1	2	1
0	0	0	0

Untuk logika AND NOT, nilai-nilai bobot yang digunakan adalah $w_1 = 2$ dan $w_2 = -1$ dengan nilai *threshold* 2. Sehingga arsitekturnya adalah seperti pada Gambar 1.6 dan hasil perhitungannya ada pada Tabel 1.3.



Gambar 1.6 Arsitektur McCulloch-Pitts neuron untuk data logika AND NOT

Tabel 1.3 Perhitungan data logika AND NOT

x_1	x_2	y_{in}	y
1	1	1	0
1	0	2	1
0	1	-1	0
0	0	0	0

Praktikum

1. Buka Google Colaboratory melalui tautan ini.
2. Tulis kode berikut ke dalam setiap *cell* pada *notebook* tersebut.
    a. Fungsi *Step* Biner

```
def binstep(x, th=0):
    return 1 if x >= th else 0
```

    b. Fungsi McCulloch-Pitts (MCP) Neuron

```
import numpy as np
```

```
def MCP(x, w, th):
    y_in = np.dot(x, w)
    y_out = binstep(y_in, th)

    return y_out
```

c. Fungsi Hitung Akurasi

```
def calc_accuracy(a, b):
    s = [1 if a[i] == b[i] else 0 for i in range(len(a))]

    return sum(s) / len(a)
```

d. Fungsi Logika AND

```
def AND(X):
    w = 1, 1
    th = 2
    y = [MCP(i, w, th) for i in X]

    return y
```

e. Eksekusi Logika AND

```
data = (0, 0), (0, 1), (1, 0), (1, 1)
output = AND(data)
true = 0, 0, 0, 1
accuracy = calc_accuracy(output, true)

print('Output:', output)
print('True:', true)
print('Accuracy:', accuracy)
```

f. Fungsi Logika OR
   Buat fungsi baru untuk logika OR berdasarkan fungsi logika AND dengan mengubah nilai bobot menjadi [2, 2].

g. Eksekusi Logika OR

```
data = (0, 0), (0, 1), (1, 0), (1, 1)
```

```
output = OR(data)
true = 0, 1, 1, 1
accuracy = calc_accuracy(output, true)

print('Output:', output)
print('True:', true)
print('Accuracy:', accuracy)
```

h. Logika AND NOT
   Buat fungsi baru untuk logika OR berdasarkan fungsi logika AND dengan mengubah nilai bobot menjadi [2, -1].

i. Eksekusi Logika AND NOT

```
data = (0, 0), (0, 1), (1, 0), (1, 1)
output = ANDNOT(data)
true = 0, 0, 1, 0
accuracy = calc_accuracy(output, true)

print('Output:', output)
print('True:', true)
print('Accuracy:', accuracy)
```

j. Logika XOR

```
def XOR(X):
    X_flip = [(i[1], i[0]) for i in X]
    y1 = ANDNOT(X)
    y2 = ANDNOT(X_flip)
    y = zip(y1, y2)
    z = OR(y)

    return z
```

k. Eksekusi Logika XOR

```
data = (0, 0), (0, 1), (1, 0), (1, 1)
output = XOR(data)
true = 0, 1, 1, 0
accuracy = calc_accuracy(output, true)

print('Output:', output)
print('True:', true)
print('Accuracy:', accuracy)
```

Analisis

1. Jalankan semua *cell* (`Ctrl+F9`). Amati *output* dan akurasi yang dihasilkan.
2. Pada kode a, apa maksud dari *statement* `return 1 if x >= th else 0`?
3. Pada kode b:
    a. Apakah perbedaan antara variabel `y_in` dan `y_out`?
    b. Apakah yang dilakukan oleh fungsi `np.dot()`?
4. Pada kode c, apakah kegunaan dari variabel-variabel `w` dan `th`?
5. Pada kode j, apakah tujuan dari variabel `X_flip`?
6. Pada hal apa saja terdapat kesamaan antara neuron biologis dan neuron tiruan?
7. Jelaskan kelebihan dan kekurangan yang dimiliki McCulloch-Pitts neuron.
8. Mengapa logika XOR lebih rumit sehingga membutuhkan kombinasi beberapa logika yang lain?