Praktikum 2
Hebb Net

Tujuan

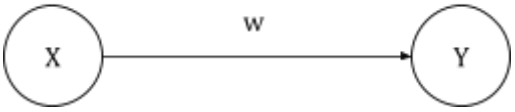
1. Praktikan mampu memahami algoritma pembelajaran Hebb *rule*
2. Praktikan mampu mengimplementasikan Hebb Net

Dasar Teori

Bias

Bias adalah salah satu hal yang penting pada jaringan saraf tiruan sehingga digunakan pada hampir semua arsitektur jaringan saraf tiruan. Bias adalah bobot dari neuron yang ditambahkan ke suatu jaringan saraf tiruan, di luar neuron-neuron yang kita tetapkan. Neuron tambahan tersebut tidak melakukan proses aktivasi dan memiliki nilai aktivasi yang konstan, yaitu 1. Sedangkan *bias* sama dengan bobot-bobot yang lain yang nilainya dinamis dan dapat dicari oleh suatu algoritma *learning*.

Jika kita merancang jaringan saraf tiruan yang hanya memiliki satu *input neuron* dan satu *output neuron*, maka arsitekturnya adalah seperti pada Gambar 2.1.



Gambar 2.1 Jaringan saraf tiruan sederhana dengan satu *input neuron* dan satu *output neuron*

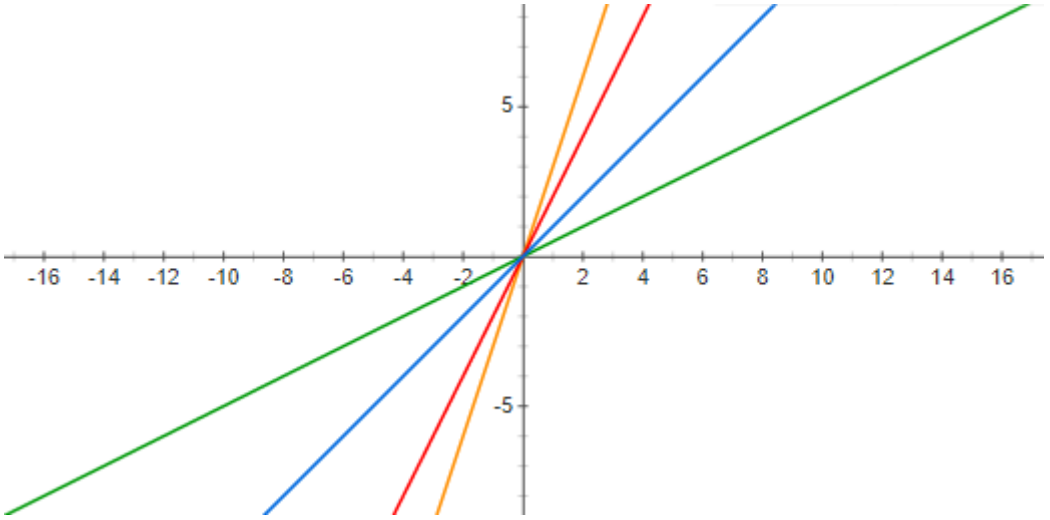
Dengan arsitektur sederhana tersebut, perhitungan yang dilakukan adalah:

$$y\_in = \sum_{i=1}^{n} w_i x_i$$
$$= w_1 x_1$$

Dari persamaan tersebut, terlihat bahwa kita menghasilkan suatu persamaan garis. Garis yang dihasilkan dapat memiliki kemiringan/gradien yang berbeda-beda tetapi selalu melewati titik (0, 0), seperti diilustrasikan pada Gambar 2.2. Ini terjadi karena pada persamaan di atas, hanya terdapat variabel w_1 yang mengatur gradien garisnya saja. Dengan bentuk garis tersebut, jaringan saraf tiruan akan kurang memiliki fleksibilitas dan *coverage* dalam memodelkan data yang ada. Di sisi lain, suatu persamaan garis memiliki bentuk umum:

$$y = mx + c$$

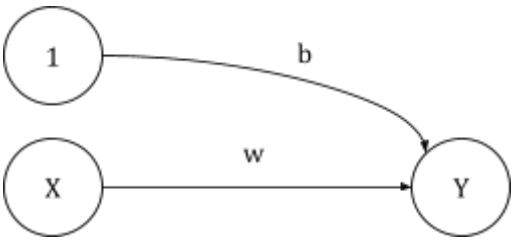
sehingga garis yang terbentuk dapat memiliki gradien (m) dan pergeseran (c) yang berbeda-beda.



Gambar 2.2 Garis-garis yang dapat dihasilkan dengan persamaan $y = mx$

Maka, untuk memberikan pergeseran pada garis yang dihasilkan oleh jaringan saraf tiruan tersebut, satu neuron ditambahkan pada *input layer* beserta bobotnya yang disimbolkan dengan b sehingga arsitekturnya menjadi seperti pada Gambar 2.3.

Contoh tersebut hanya merupakan ilustrasi penambahan neuron beserta *bias*-nya saja. McCulloch-Pitts neuron, pada rancangan aslinya, tidak menggunakan neuron tambahan dan *bias*.

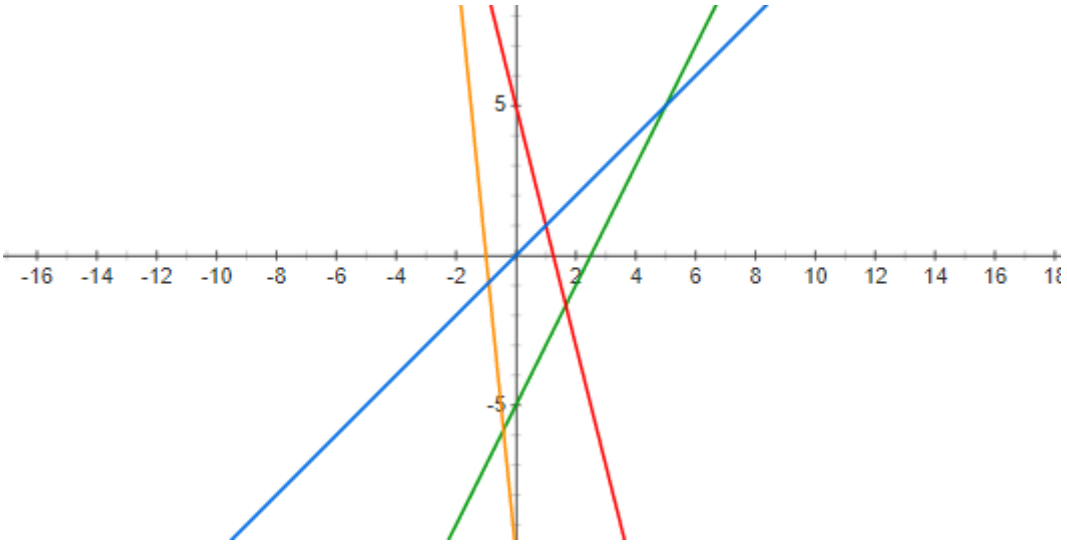


Gambar 2.3 Jaringan saraf tiruan sederhana dengan penambahan neuron dan *bias*

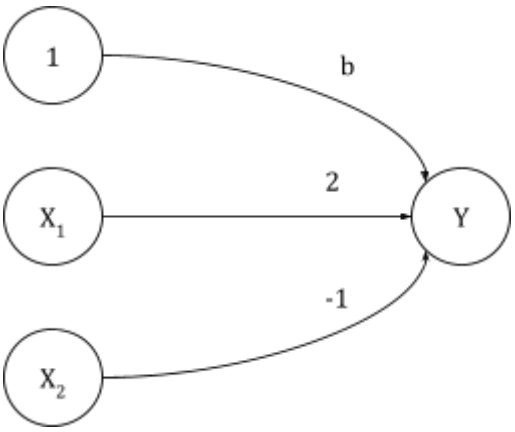
Dengan penambahan bias b, maka perhitungan y_in menjadi:

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

Dengan begitu, persamaan tersebut memiliki variabel yang mengatur pergeseran garis dan dapat menghasilkan bentuk-bentuk garis yang lebih bervariasi, seperti yang diilustrasikan pada Gambar 2.4. Perhatikan bahwa garis-garis tersebut memiliki posisi yang lebih fleksibel daripada garis-garis pada Gambar 2.2 karena tidak harus melewati titik (0, 0).



Gambar 2.4 Garis-garis yang dapat dihasilkan dengan penambahan neuron dan *bias*



Gambar 2.3 McCulloch-Pitts neuron

Decision Boundary

Proses pelatihan pada suatu kasus klasifikasi, tujuan yang ingin dicapai adalah membuat suatu pemisah antara kelas-kelas yang ingin diklasifikasikan. Jika divisualisasikan, pemisah ini dapat berupa garis atau kurva. Garis atau kurva pemisah ini disebut dengan *decision boundary*.

Pada praktikum sebelumnya, pembentukan logika AND, OR, AND NOT, dan XOR sebenarnya merupakan permasalahan klasifikasi biner (dua kelas) yang memisahkan antara kelas *true* dan *false*. Sebagai contoh, pada kasus logika AND dengan McCulloch-Pitts neuron, berdasarkan nilai-nilai bobot dan *threshold* yang digunakan ($w_1 = 1; w_2 = 1; \theta = 2$), kita dapat mencari *decision boundary*-nya seperti berikut.

Epoch

Pada *machine learning*, *epoch* berarti suatu fase pemrosesan semua baris atau *instance* data sebanyak satu kali. Sebagai contoh, jika kita memiliki seratus baris data, maka setelah data keseratus selesai diproses, maka pemrosesan telah selesai dilakukan sebanyak satu *epoch*. Jika proses dilakukan kembali dari baris pertama, maka proses memasuki *epoch* berikutnya.

Beberapa algoritma pelatihan jaringan saraf tiruan melakukan proses pelatihan sebanyak lebih dari satu *epoch* hingga suatu kondisi berhenti tercapai. Kondisi berhenti yang umum digunakan adalah saat *error* (selisih antara *output* dan target) telah mencapai kurang dari suatu batas nilai tertentu atau saat proses pelatihan telah berjalan sebanyak sekian *epoch*.

Hebb Net

Algoritma pelatihan Hebb net disebut dengan Hebb rule. Hebb rule hanya berjalan sebanyak satu *epoch*. Artinya, setiap data latih hanya diproses sebanyak satu kali selama proses pelatihan.

Praktikum

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

```
def bipstep(y, th=0):
    return 1 if y >= th else -1
```

    b. Fungsi *Training* Hebb

```
def hebb_fit(train, target, verbose=False, draw=False,
draw_padding=1):
    w = np.zeros(len(train[0]) + 1)
    bias = np.ones((len(train), 1))
    train = np.hstack((bias, train))

    for r, row in enumerate(train):
        w = [w[i] + row[i] * target[r] for i in range(len(row))]

        if verbose:
            print('Bobot:', w)

        if draw:
            plot(line(w, 0), train, target, draw_padding)

    return w
```

    c. Fungsi *Testing* Hebb

```
def hebb_predict(X, w):
    Y = []

    for x in X:
        y_in = w[0] + np.dot(x, w[1:])
```

```
        y = bipstep(y_in)

        Y.append(y)

    return Y
```

d. 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)
```

e. Logika AND

```
from sklearn.metrics import accuracy_score

train = (1, 1), (1, -1), (-1, 1), (-1, -1)
target = 1, -1, -1, -1
model = hebb_fit(train, target, verbose=False, draw=False)
output = hebb_predict(train, model)
accuracy = accuracy_score(output, target)

print('Output:', output)
print('Target:', target)
print('Accuracy:', accuracy)
```

f. Logika OR

```
train = (1, 1), (1, -1), (-1, 1), (-1, -1)
target = 1, 1, 1, -1
model = hebb_fit(train, target, verbose=True, draw=True)
output = hebb_predict(train, model)
accuracy = accuracy_score(output, target)

print('Output:', output)
print('Target:', target)
print('Accuracy:', accuracy)
```

g. Logika AND NOT

```
train = (1, 1), (1, -1), (-1, 1), (-1, -1)
target = -1, 1, -1, -1
```

```
model = hebb_fit(train, target, verbose=True, draw=True)
output = hebb_predict(train, model)
accuracy = accuracy_score(output, target)

print('Output:', output)
print('Target:', target)
print('Accuracy:', accuracy)
```

h. Logika XOR

```
train = (1, 1), (1, -1), (-1, 1), (-1, -1)
target = -1, 1, 1, -1
model = hebb_fit(train, target, verbose=True, draw=True)
output = hebb_predict(train, model)
accuracy = accuracy_score(output, target)

print('Output:', output)
print('Target:', target)
print('Accuracy:', accuracy)
```

Analisis

1. Pada klasifikasi menggunakan logika XOR, mengapa akurasi yang didapatkan tidak mencapai 1 (100%)?
2. Lakukan proses training dan testing menggunakan data berikut.
   Training: `(-1, .5), (.5, .3), (1, 1.5), (3, 1.9)`
   Target: `(-1, -1, 1, 1)`
   Berapakah akurasi yang didapatkan? Mengapa tidak dapat mencapai akurasi 1 (100%)?