Praktikum 3
Perceptron

Tujuan

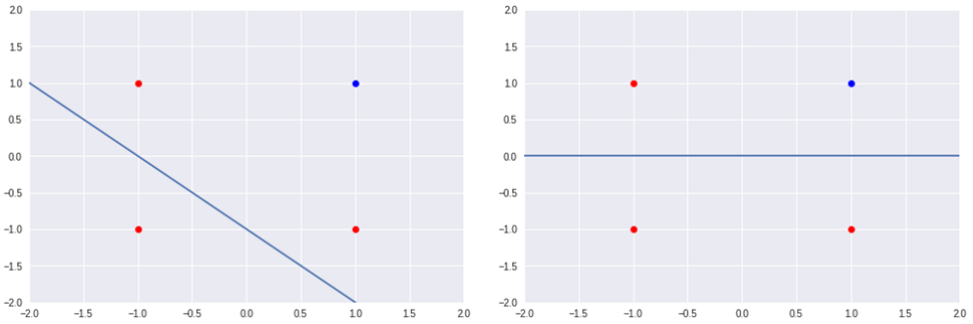
1. Praktikan mampu memahami algoritma Perceptron
2. Praktikan mampu mengaplikasikan algoritma Perceptron untuk melakukan klasifikasi
3. Praktikan mampu menjelaskan keterbatasan algoritma Perceptron

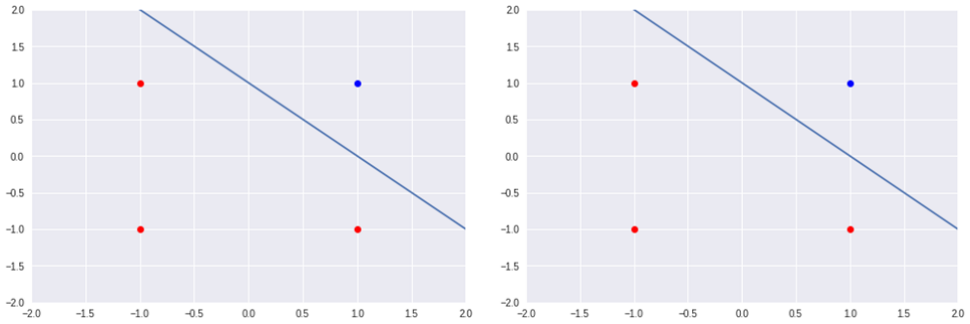
Dasar Teori

Learning Rate

Pada suatu proses pelatihan, dilakukan pencarian nilai-nilai bobot yang dapat menyelesaikan suatu kasus dengan baik. Pencarian tersebut dilakukan secara otomatis berdasarkan data latih yang dimasukkan. Pada suatu kasus klasifikasi, maka nilai-nilai bobot tersebut merepresentasikan *decision boundary* yang dapat memisahkan data latih dari kelas yang berbeda.

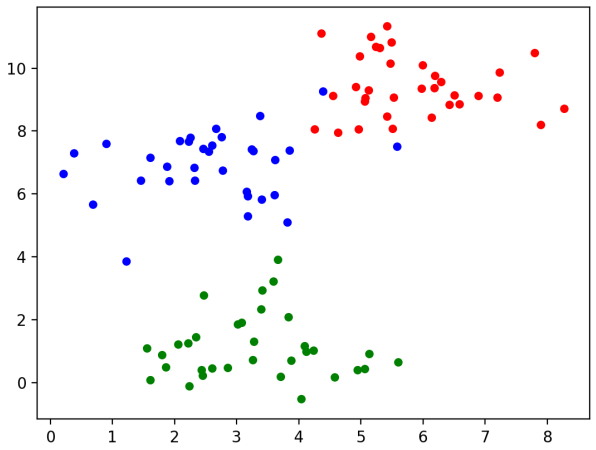
Pada salah satu contoh kasus aplikasi algoritma pelatihan Hebb *rule* pada Bab 2, digunakan data sederhana berupa data logika AND. Kemudian Hebb *rule* dijalankan untuk mencari *decision boundary* yang dapat memisahkan antara data yang termasuk pada kelas *true* dan kelas *false*. Pada setiap langkah pelatihan, nilai-nilai bobot pada jaringan dapat berubah. Karena nilai-nilai bobot tersebut merepresentasikan garis *decision boundary*, maka artinya *decision boundary* tersebut bergeser pada setiap langkah pelatihan. Pada contoh kasus logika AND tersebut, pergeseran *decision boundary*-nya digambarkan pada Gambar 3.1.





Gambar 3.1 Pergeseran *decision boundary* (dari kiri atas hingga kanan bawah) selama proses pelatihan dengan Hebb rule menggunakan data logika AND

Perhatikan pada Gambar 3.1 bahwa pergeseran *decision boundary* terjadi dengan langkah pergeseran yang besar. Pada data sederhana seperti yang digunakan pada contoh tersebut, pergeseran seperti itu tidak menimbulkan masalah. Namun, akan beda halnya jika data yang digunakan lebih kompleks dengan jumlah data dan dimensi data yang tinggi dan data pada kedua kelas yang harus dipisahkan memiliki jarak yang kecil. Gambar 3.2 memberikan satu contoh data yang memiliki kerapatan yang cukup tinggi.



Gambar 3.2 Data yang memiliki kerapatan yang tinggi

Dalam kondisi seperti itu, pergeseran *decision boundary* tidak dapat dilakukan dengan langkah pergeseran yang besar karena akan berpotensi mengakibatkan posisi *decision boundary* yang tidak baik. Maka, dibuatlah satu parameter yang mengatur seberapa besar langkah pergeseran *decision boundary* selama proses pelatihan berjalan. Parameter ini disebut dengan *learning rate* yang disimbolkan dengan alpha (α).

Nilai *learning rate* berkisar antara 0 hingga 1 ($0 < \alpha \leq 1$). Penentuan nilai *learning rate* ini krusial dan sangat memengaruhi proses maupun hasil

pelatihan. Jika nilainya terlalu rendah, maka perubahan nilai-nilai bobot atau pergeseran *decision boundary* akan terjadi dengan sangat perlahan pada setiap iterasinya. Akibatnya, proses pelatihan akan berlangsung dengan sangat lama. Sebaliknya, jika *learning rate* bernilai terlalu tinggi, maka pergeseran *decision boundary* akan menjadi terlalu besar sehingga ketelitiannya buruk dan sangat berpotensi tidak akan menemukan solusi yang baik. *Learning rate* dengan nilai yang baik –tidak terlalu tinggi dan tidak terlalu rendah– akan dapat membawa proses pelatihan menemukan solusi yang baik dengan jumlah iterasi pelatihan yang tidak terlalu banyak sehingga tidak memakan waktu yang terlalu lama.

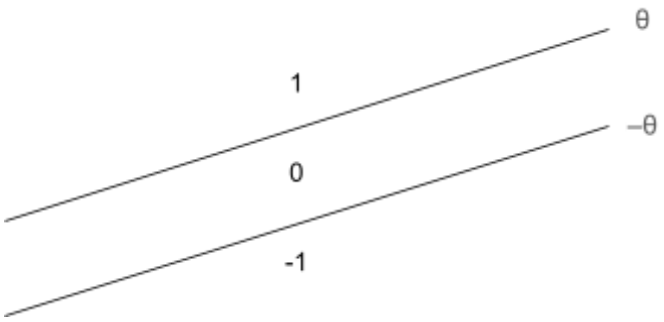
Perceptron

Pada awalnya, Perceptron (Rosenblatt, 1958) ditujukan untuk ditanamkan pada sebuah mesin. Perceptron diimplementasikan pertama kali pada komputer IBM 704 kemudian pada sebuah mesin yang diberi nama "Mark 1 Perceptron", sebuah mesin yang didesain untuk melakukan *image recognition*.

Perceptron merupakan suatu *linear classifier*. Artinya, Perceptron hanya dapat menyelesaikan kasus atau data yang bersifat *linearly separable*. Ini merupakan hal yang menjadi kritik oleh Minsky dan Papert (1969) yang menunjukkan bahwa Perceptron tidak bisa menyelesaikan XOR yang merupakan kasus yang sederhana namun bersifat *nonlinearly-separable*. Hal tersebut menyebabkan penurunan minat para peneliti dan pendanaannya untuk mengembangkan jaringan saraf tiruan selama lebih dari sepuluh tahun yang disebut dengan periode *AI winter* yang terjadi hingga tahun 1980-an.

Namun begitu, Perceptron tetap merupakan algoritma yang sangat penting pada jaringan saraf tiruan. Perceptron memiliki algoritma yang lebih baik daripada Hebb net. Perceptron melakukan proses pelatihan secara iteratif dan dapat berjalan dalam banyak *epoch*. Perceptron dapat diatur laju pelatihannya dengan menentukan parameter *learning rate*.

Algoritma Perceptron menggunakan fungsi aktivasi dengan dua batas nilai, yaitu θ dan $-\theta$ sehingga dapat menghasilkan tiga *output*, yaitu 1, 0, dan -1. Jika nilai θ dibuat sama, maka Perceptron menggunakan arsitektur *single layer* dengan *bias* pada *input layer*.



Gambar 3.3 *Decision boundary* pada Perceptron

Jika data latih berisi dua kelas yang bersifat *linearly separable*, maka algoritma Perceptron menjamin bahwa kedua kelas tersebut akan dapat dipisahkan. Namun jika data latihnya bersifat *nonlinearly-separable*, maka proses pelatihan algoritma Perceptron tidak akan pernah selesai.

Perceptron hanya dapat menyelesaikan kasus yang bersifat *linearly separable* karena hanya memiliki satu *output neuron*. Permasalahan yang sifatnya *nonlinearly-separable* sebenarnya dapat diselesaikan dengan mengembangkan arsitektur Perceptron menjadi *multilayer* yang disebut dengan *multilayer* Perceptron (MLP). Namun, algoritma pelatihan jaringan saraf tiruan *multilayer* lebih kompleks daripada algoritma pelatihan jaringan saraf tiruan *single layer*.

Algoritma Pelatihan Perceptron

Berikut adalah algoritma pelatihan Perceptron.

1. Inisialisasi nilai-nilai bobot (termasuk *bias*) dan tentukan *learning rate* (α).
2. Untuk setiap data latih *s* dan target *t*, lakukan langkah 3–5.
3. Set nilai aktivasi setiap *input neuron*:
$$x_i = s_i$$
4. Hitung nilai aktivasi *output neuron*:
5. Jika $y \neq t$:
$w_i' = w_i + \alpha t x$
$b' = b + \alpha t$
Jika $y = t$, maka nilai semua bobot dan *bias* tidak diubah.
6. Setelah semua data selesai diproses, jika tidak ada perubahan nilai bobot, hentikan pelatihan. Jika ada perubahan nilai bobot, lanjutkan pelatihan ke *epoch* berikutnya (kembali ke langkah 2).

Praktikum

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

```
def percep_step(input, th=0):
    return 1 if input > th else -1 if input < -th else 0
```

    b. Fungsi *Training* Perceptron

```
def percep_fit(X, target, th=0, a=1, max_epoch=-1, verbose=False,
draw=False):
    w = np.zeros(len(X[0]) + 1)
    bias = np.ones((len(X), 1))
    X = np.hstack((bias, X))
    stop = False
    epoch = 0

    while not stop and (max_epoch == -1 or epoch < max_epoch):
        stop = True
        epoch += 1

        if verbose:
            print('\nEpoch', epoch)

        for r, row in enumerate(X):
            y_in = np.dot(row, w)
            y = percep_step(y_in, th)

            if y != target[r]:
                stop = False

                w = [w[i] + a * target[r] * row[i] for i in
range(len(row))]

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

            if draw:
                plot(line(w, th), line(w, -th), X, target)

    return w, epoch
```

    c. Fungsi *Testing* Perceptron

```
def percep_predict(X, w, th=0):
    Y = []

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

        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

```
train = (1, 1), (1, -1), (-1, 1), (-1, -1)
target = 1, -1, -1, -1
th = .2
model, epoch = percep_fit(train, target, th, verbose=True,
draw=True)
output = percep_predict(train, model)
accuracy = calc_accuracy(output, target)

print('Epochs:', epoch)
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
th = .2
model, epoch = percep_fit(train, target, th, verbose=True,
draw=True)
output = percep_predict(train, model)
accuracy = calc_accuracy(output, target)
```

```
print('Epochs:', epoch)
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
th = .2
model, epoch = percep_fit(train, target, th, verbose=True,
draw=True)
output = percep_predict(train, model)
accuracy = calc_accuracy(output, target)

print('Epochs:', epoch)
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
th = .2
model, epoch = percep_fit(train, target, th, max_epoch=50,
verbose=True, draw=False)
output = percep_predict(train, model)
accuracy = accuracy_score(output, target)

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

Analisis

1. Mengapa Perceptron gagal dalam melakukan proses *training* menggunakan data logika XOR? Jelaskan.
2. Lakukan pelatihan data logika AND dengan *learning rate* yang berbeda-beda. Amati jumlah *epoch* yang dilakukan. Bagaimanakah efeknya pada proses pelatihan?