Backpropagation

Randy Cahya Wihandika, S.ST., M.Kom.

Backpropagation

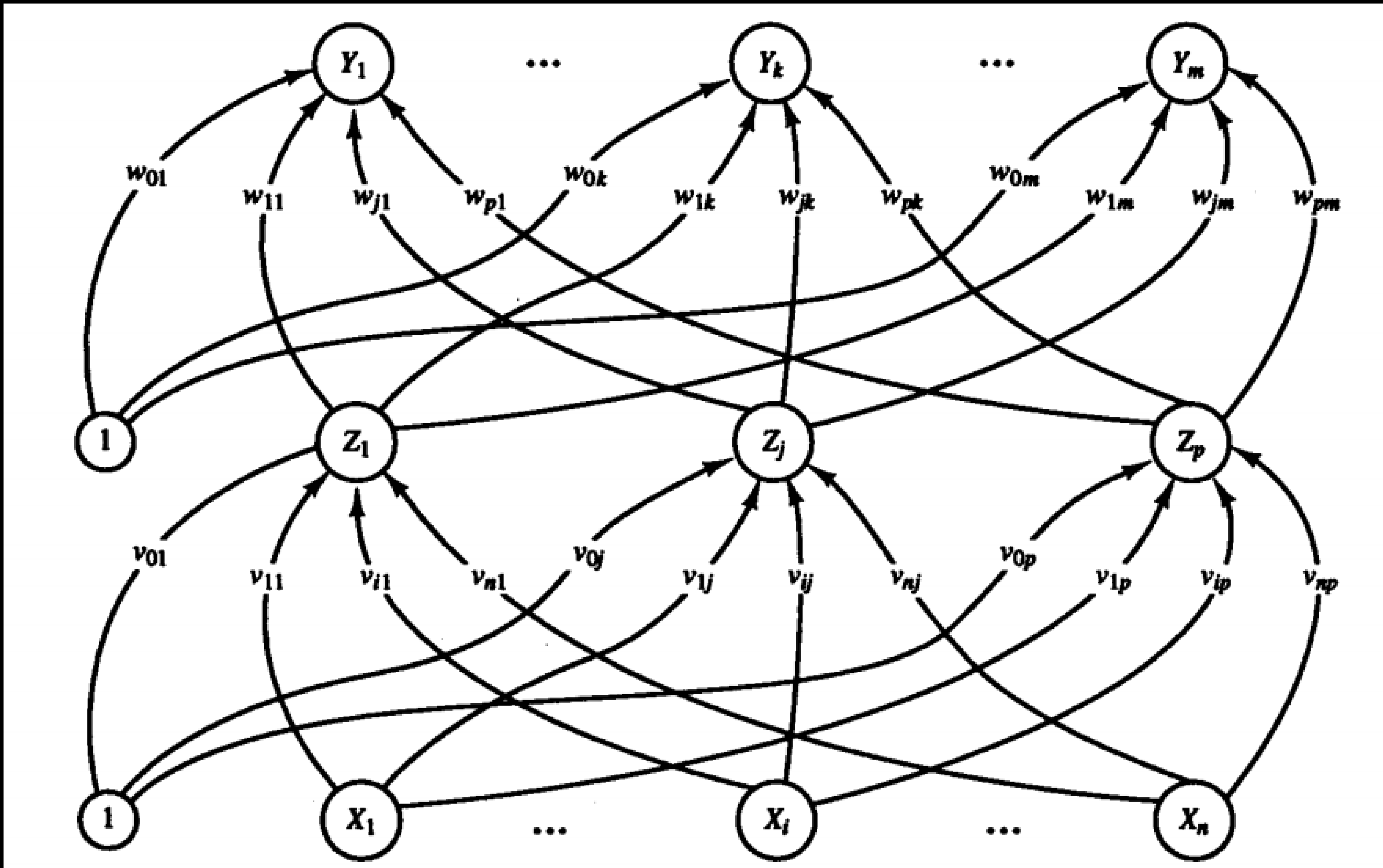
- Jaringan saraf *multilayer*
- Bekerja secara *supervised* –ada *training* dan *testing*– sehingga dapat digunakan untuk klasifikasi dan prediksi

Backpropagation

- Pada *training*, ada tiga tahap: *feedforward*, perhitungan *error* secara mundur (*backpropagation*), dan perubahan nilai bobot
- Pada *testing*, hanya dilakukan *feedforward*
- Dapat menggunakan arsitektur satu atau banyak *hidden layer*
- Pada banyak kasus, satu *hidden layer* cukup

Backpropagation

- Jumlah neuron pada *input layer* sejumlah fitur/ciri/dimensi data
- Jumlah neuron pada *output layer* sejumlah kelas atau pola kelas
- Bias dapat digunakan pada *hidden layer* dan *output layer*



Sumber: Fausett (1994)

Fungsi Aktivasi

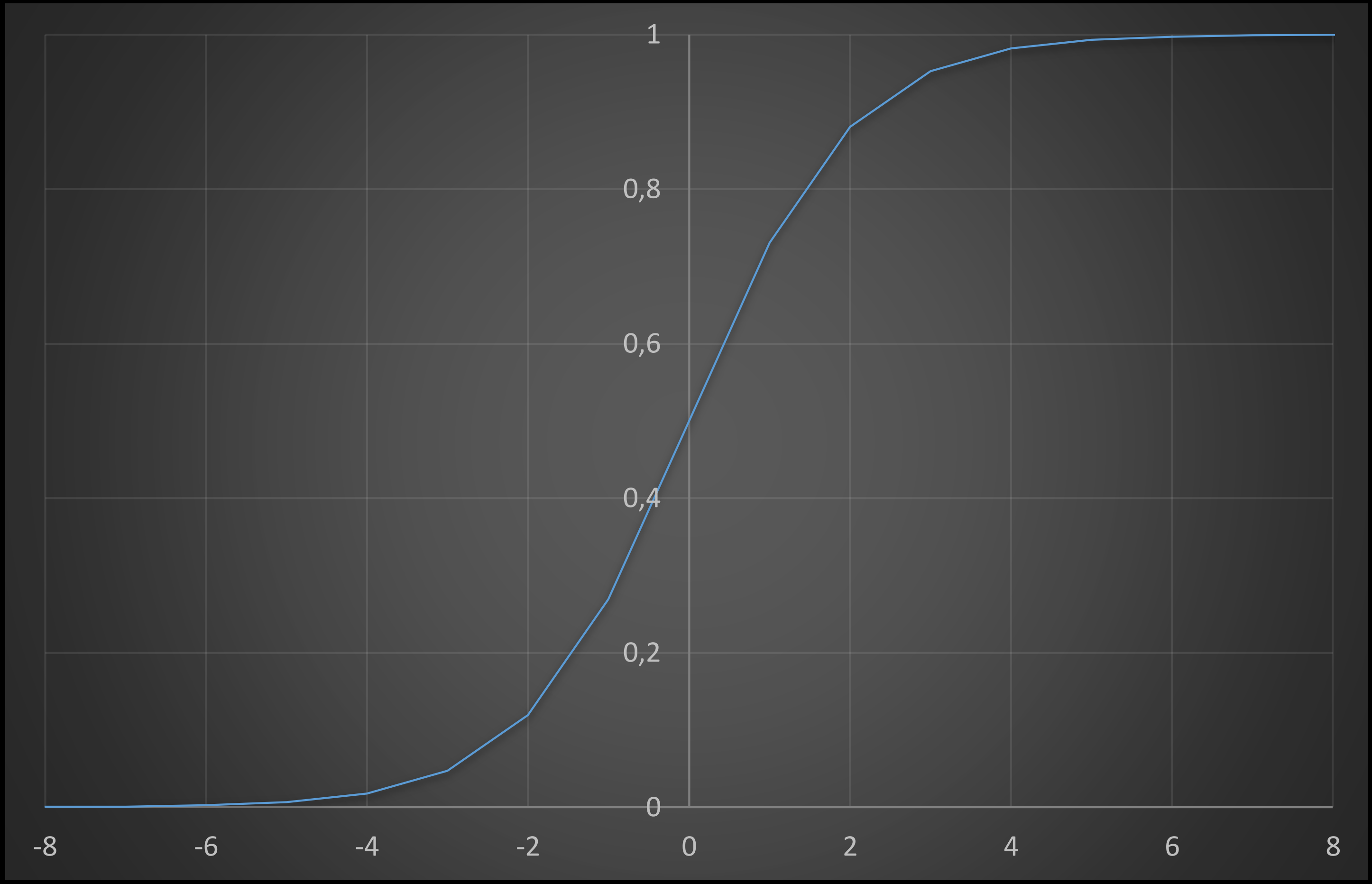
- Fungsi aktivasi yang sering digunakan adalah sigmoid biner:

$$f(x) = \frac{1}{1 + e^{-x}}$$

- Pada *input layer* tidak ada fungsi aktivasi
- Diferensial dari $f(x)$[1]:

$$f'(x) = f(x)[1 - f(x)]$$

[1] https://towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e



Fungsi Sigmoid Biner

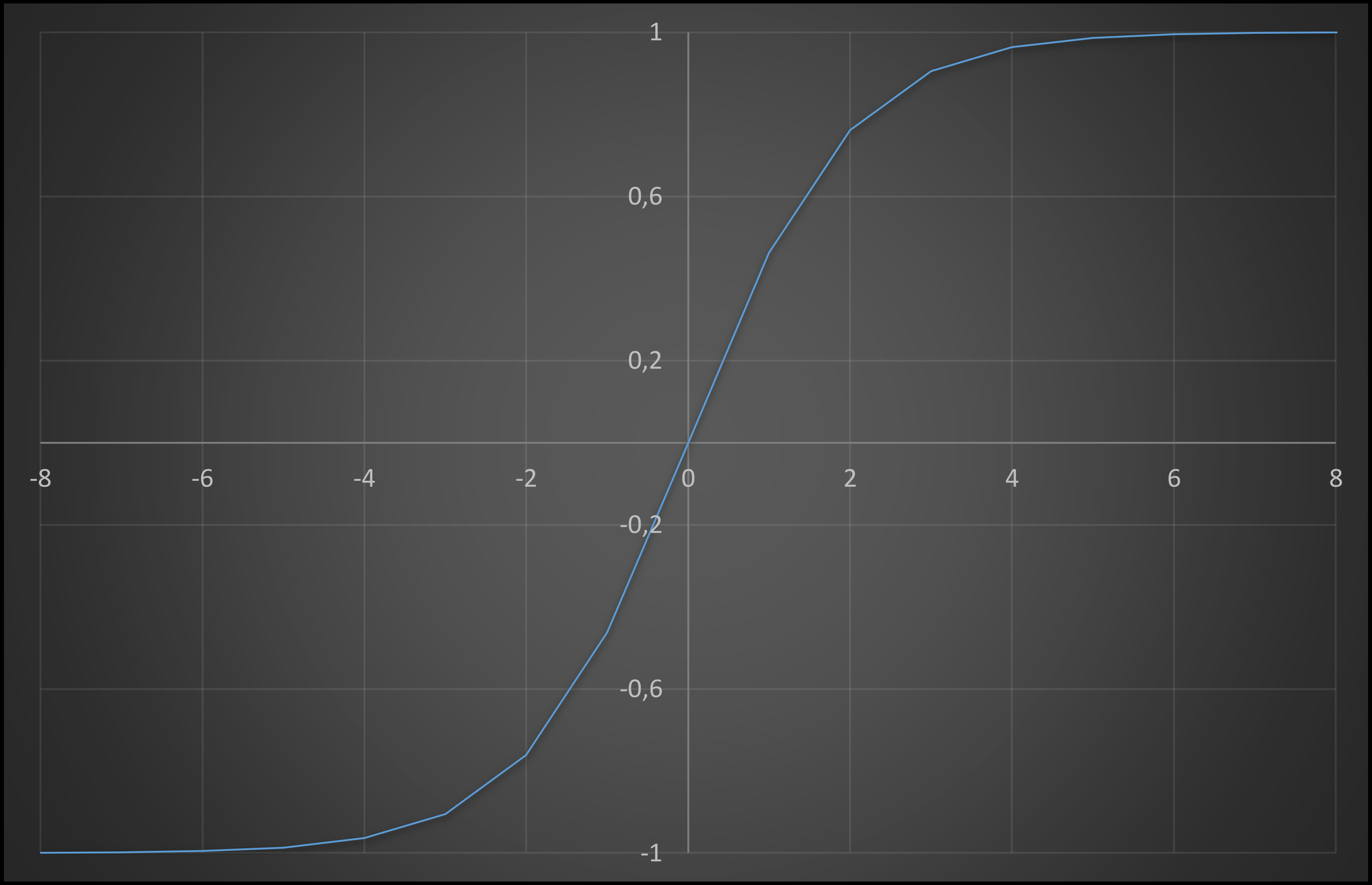
Fungsi Aktivasi

- Fungsi aktivasi yang juga dapat digunakan adalah sigmoid biner:

$$f(x) = \frac{2}{1 + e^{-x}} - 1$$

- Diferensial dari $f(x)$:

$$f'(x) = \frac{1}{2}[1 + f(x)][1 - f(x)]$$



Fungsi Sigmoid Bipolar

Algoritme

1. Inisialisasi nilai bobot; umumnya dengan nilai acak antara 0–1
2. Selama kondisi berhenti belum tercapai, lakukan langkah 3–10
3. Untuk setiap data latih, lakukan langkah 4–9

Algoritme: Feedforward

4. Setiap neuron input menerima data latih x dan meneruskan ke *hidden layer*
5. Setiap neuron *hidden* menjumlahkan (Σ) nilai yang diterima (z_{in}), menghitung nilai aktivasinya $z = f(z_{in})$, dan meneruskan ke *output layer*
6. Setiap neuron *output* menjumlahkan (Σ) nilai yang diterima (y_{in}) dan menghitung nilai aktivasinya $y = f(y_{in})$

Algoritme: Backpropagation

7. Setiap neuron output menghitung nilai *error* (selisih) antara output y dengan target t:

$$\delta_k = (t_k - y_k) f'(y_{in_k})$$

$$\Delta w_{jk} = \alpha \delta_k z_j$$

Algoritme: Backpropagation

8. Setiap neuron *hidden* menjumlahkan nilai δ dari *output layer:*

$$\delta_{in_j} = \sum_{k=1}^{m} \delta_k w_{jk}$$

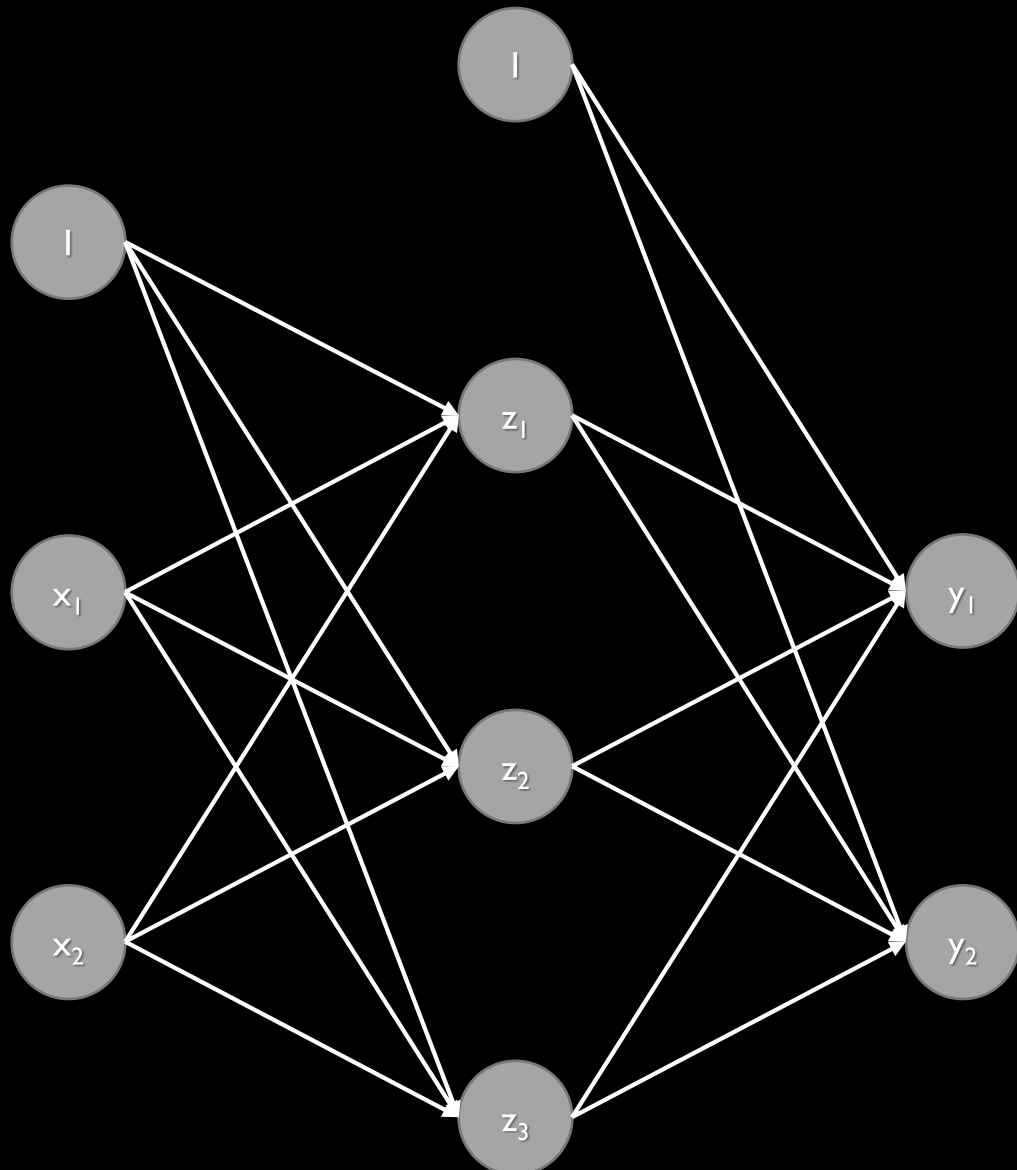
$$\delta_j = \delta_{in_j} f'\left(z_{in_j}\right)$$

$$\Delta v_{ij} = \alpha \delta_j x_i$$

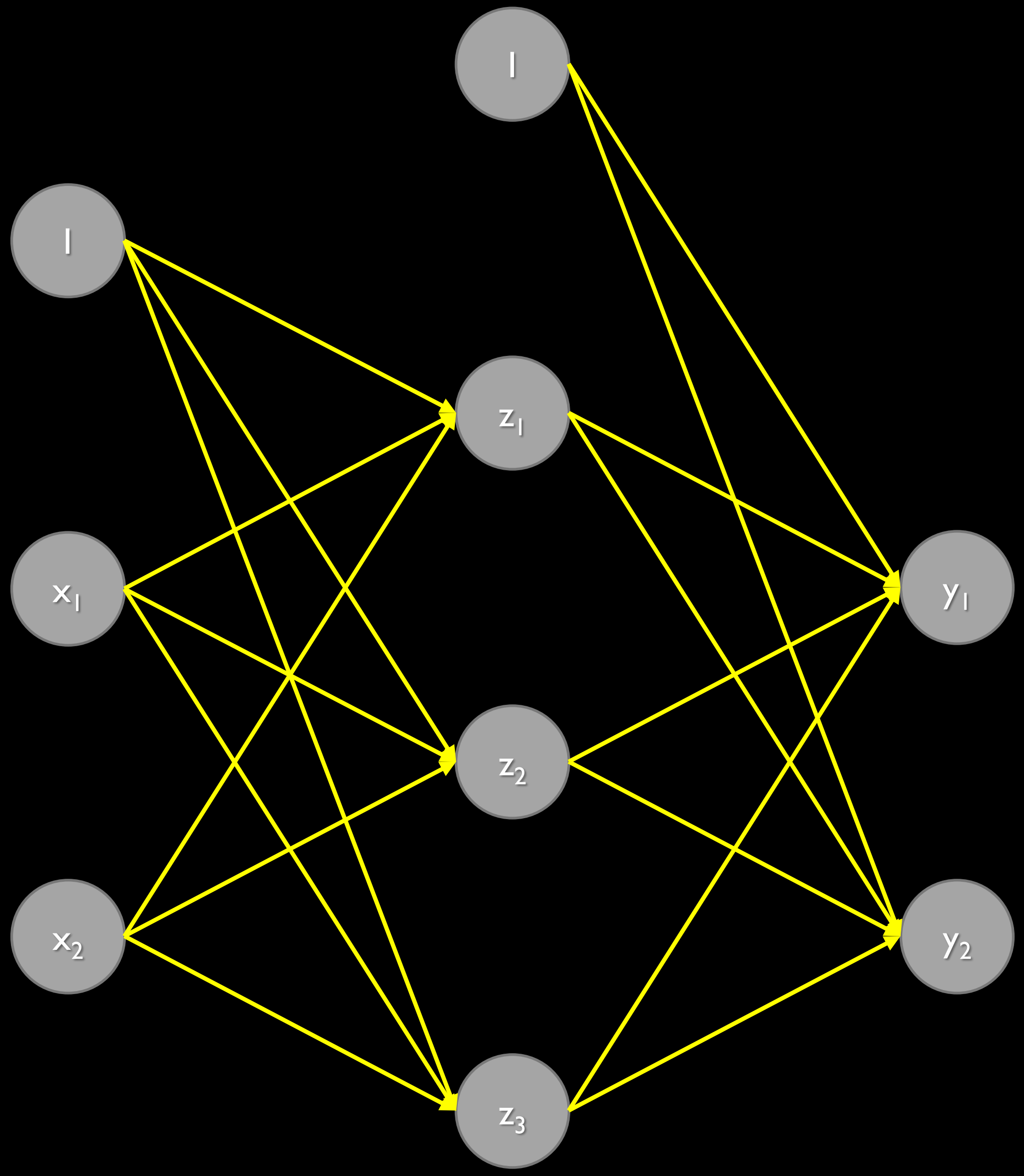
Algoritme: Update Bobot

9. Setiap neuron *output* dan *hidden* meng-*update* nilai bobotnya:

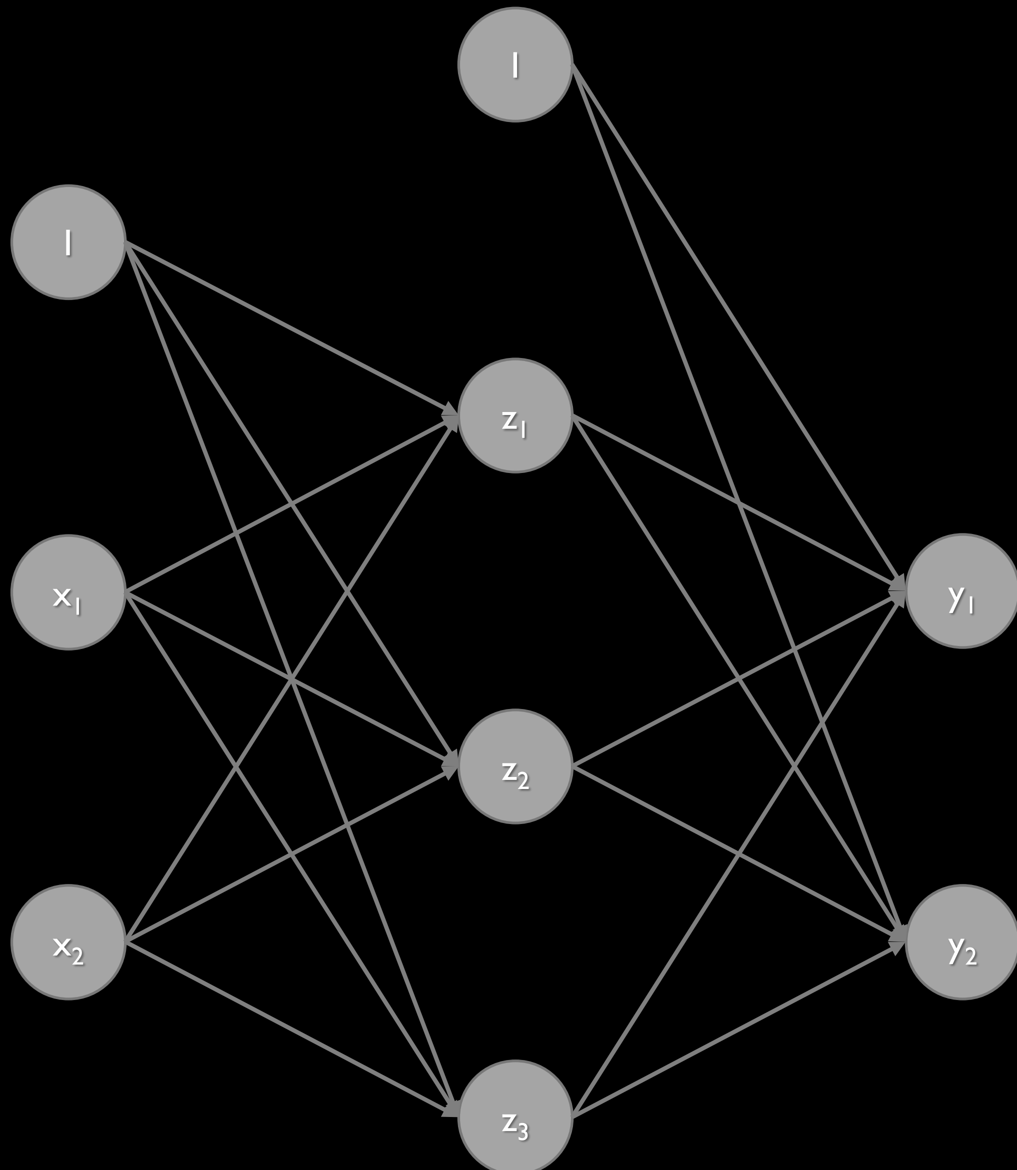
$$w'_{jk} = w_{jk} + \Delta w_{jk}$$
$$v'_{jk} = v_{jk} + \Delta v_{jk}$$



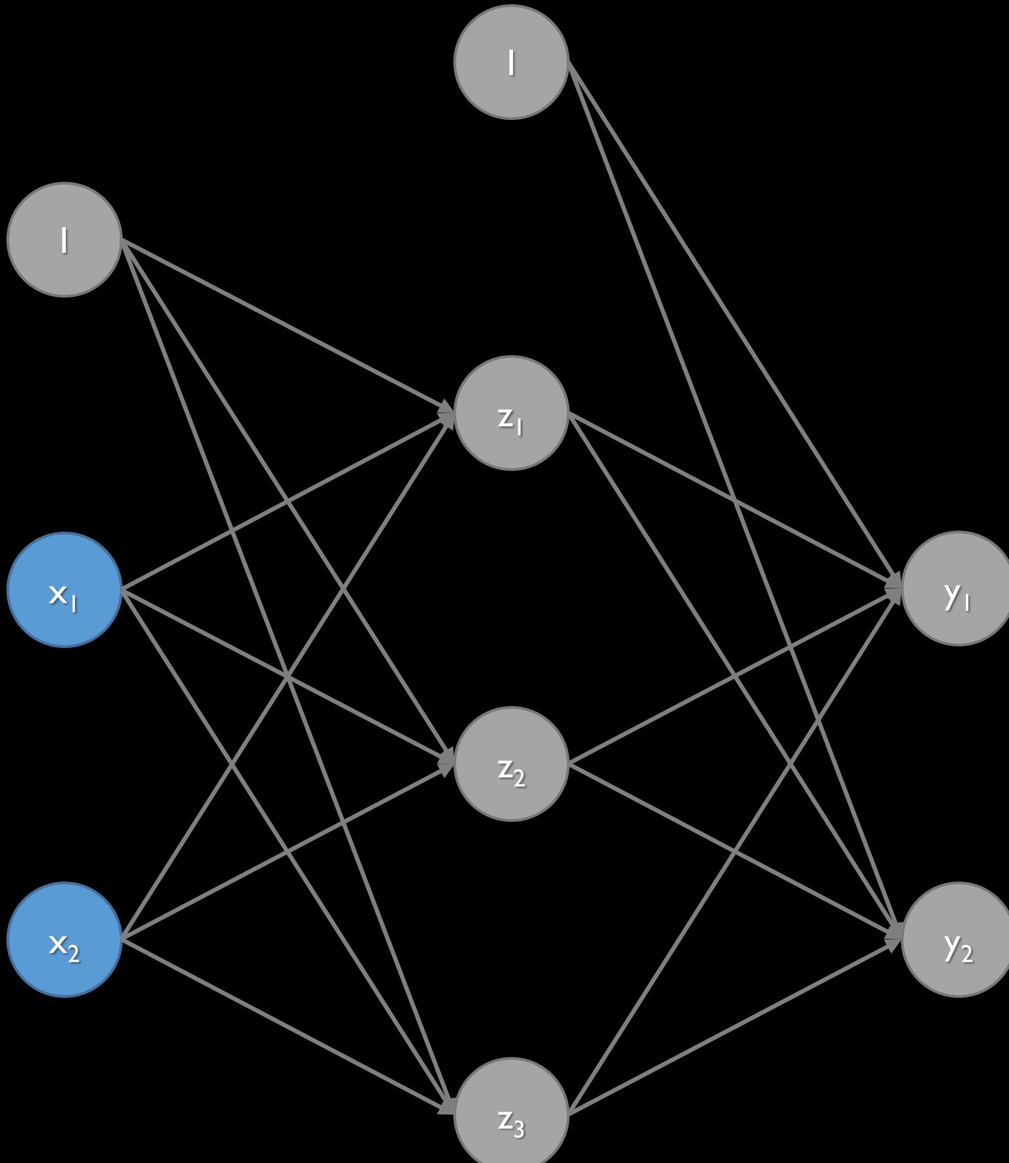
1
1
x_1
x_2
z_1
z_2
z_3
y_1
y_2



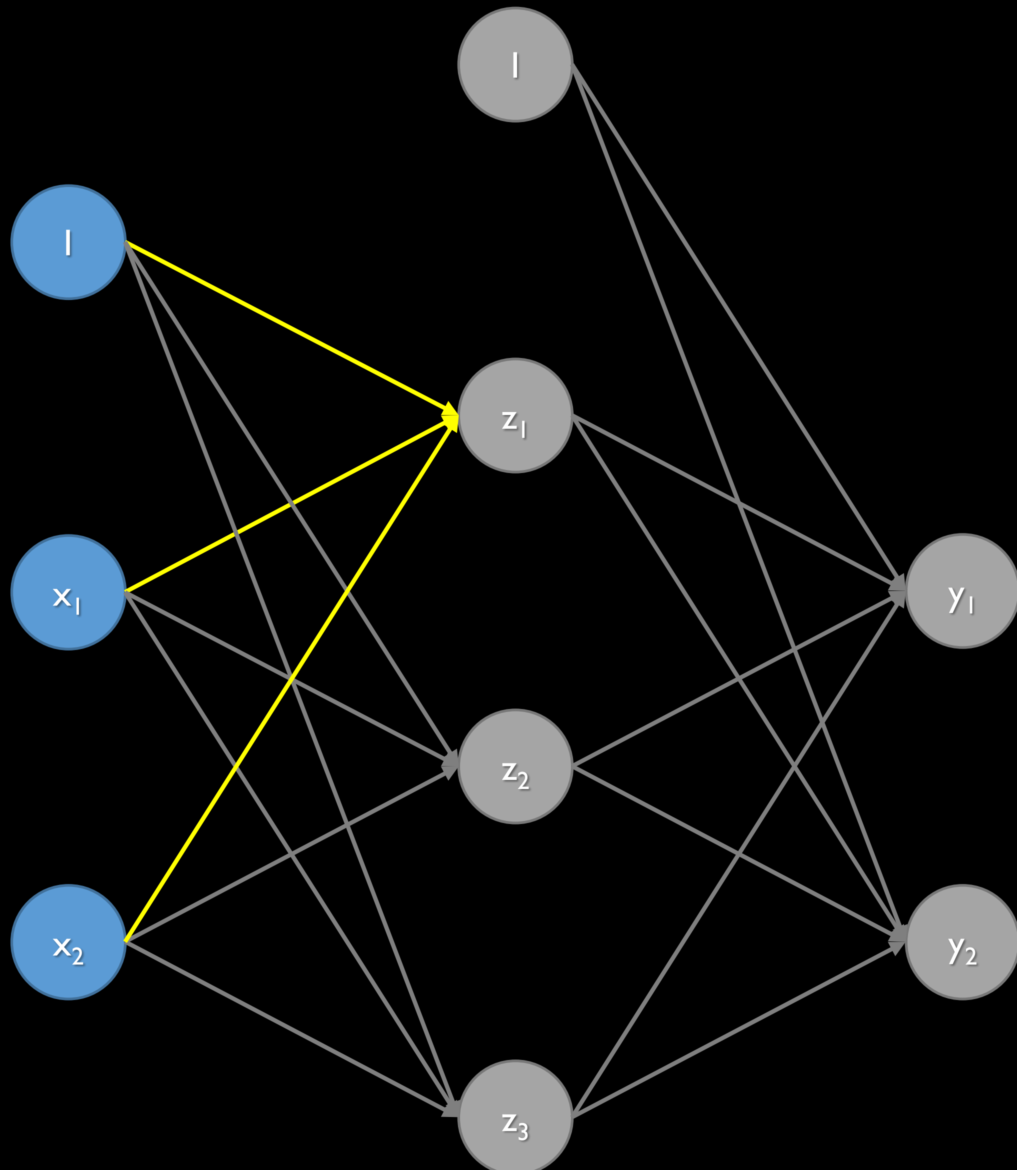
Inisialisasi nilai bobot



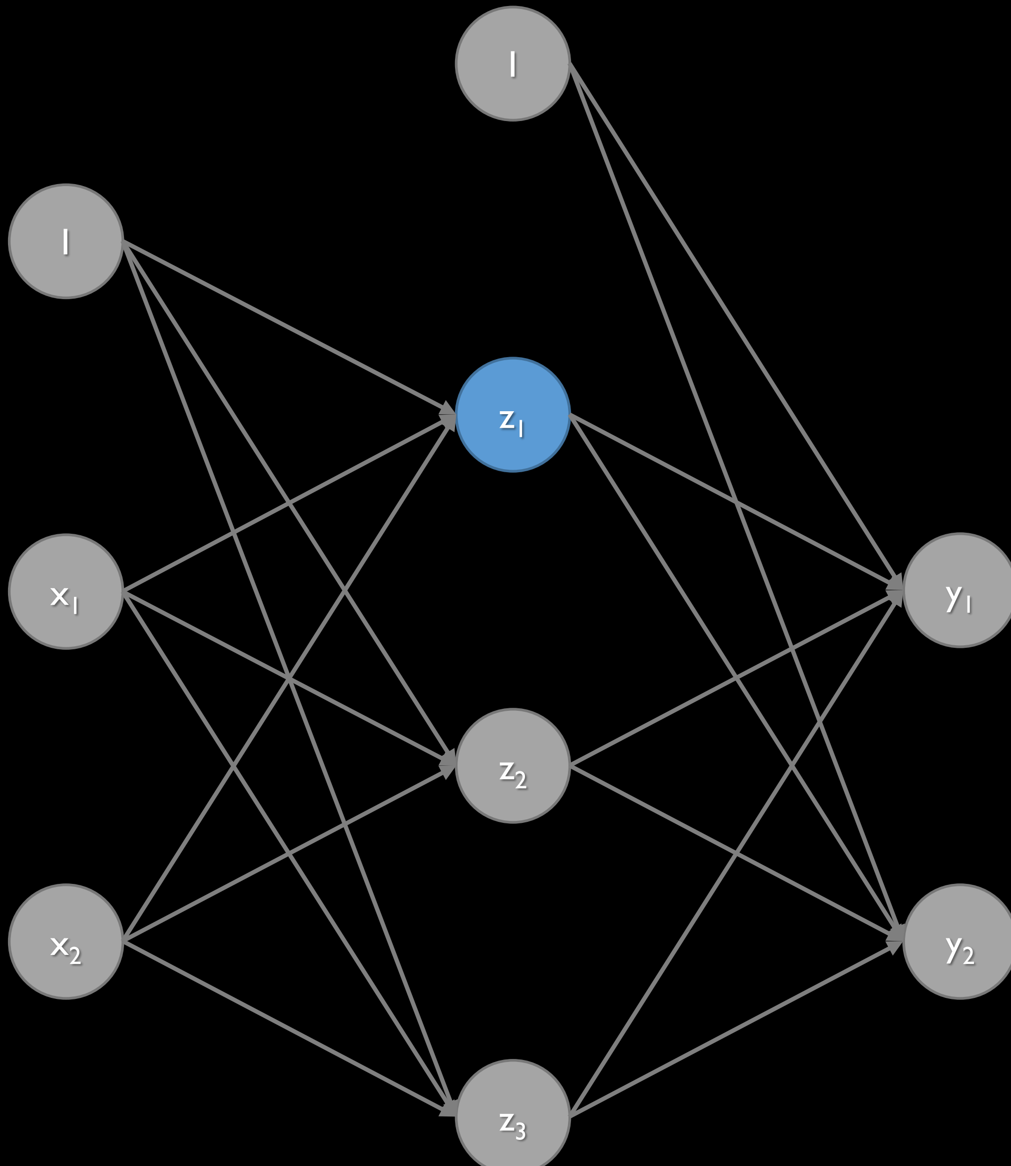
Fase 1:
Feedforward



Data latih 1 masuk
(tidak ada aktivasi)

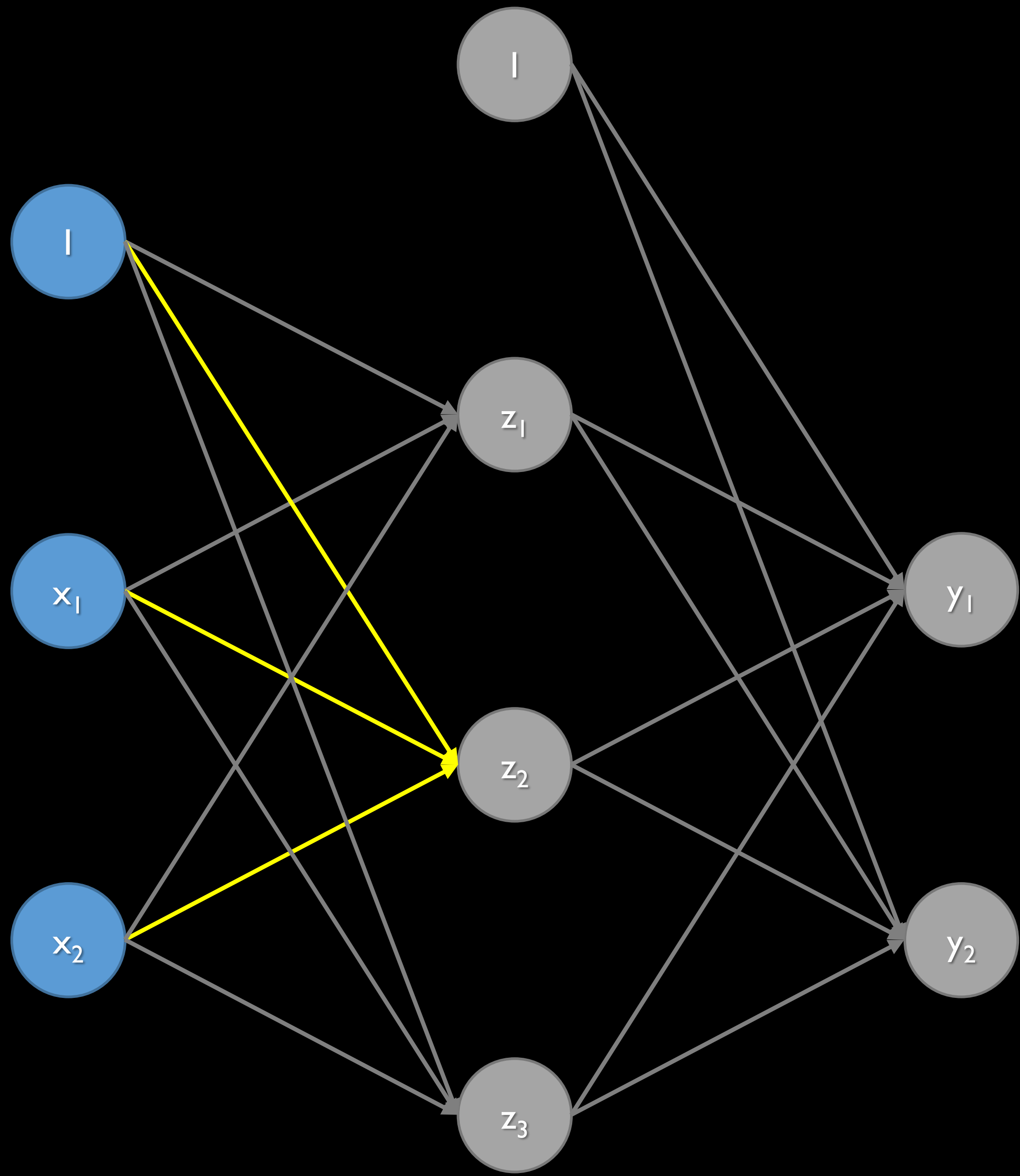


Hitung z_{in_1}

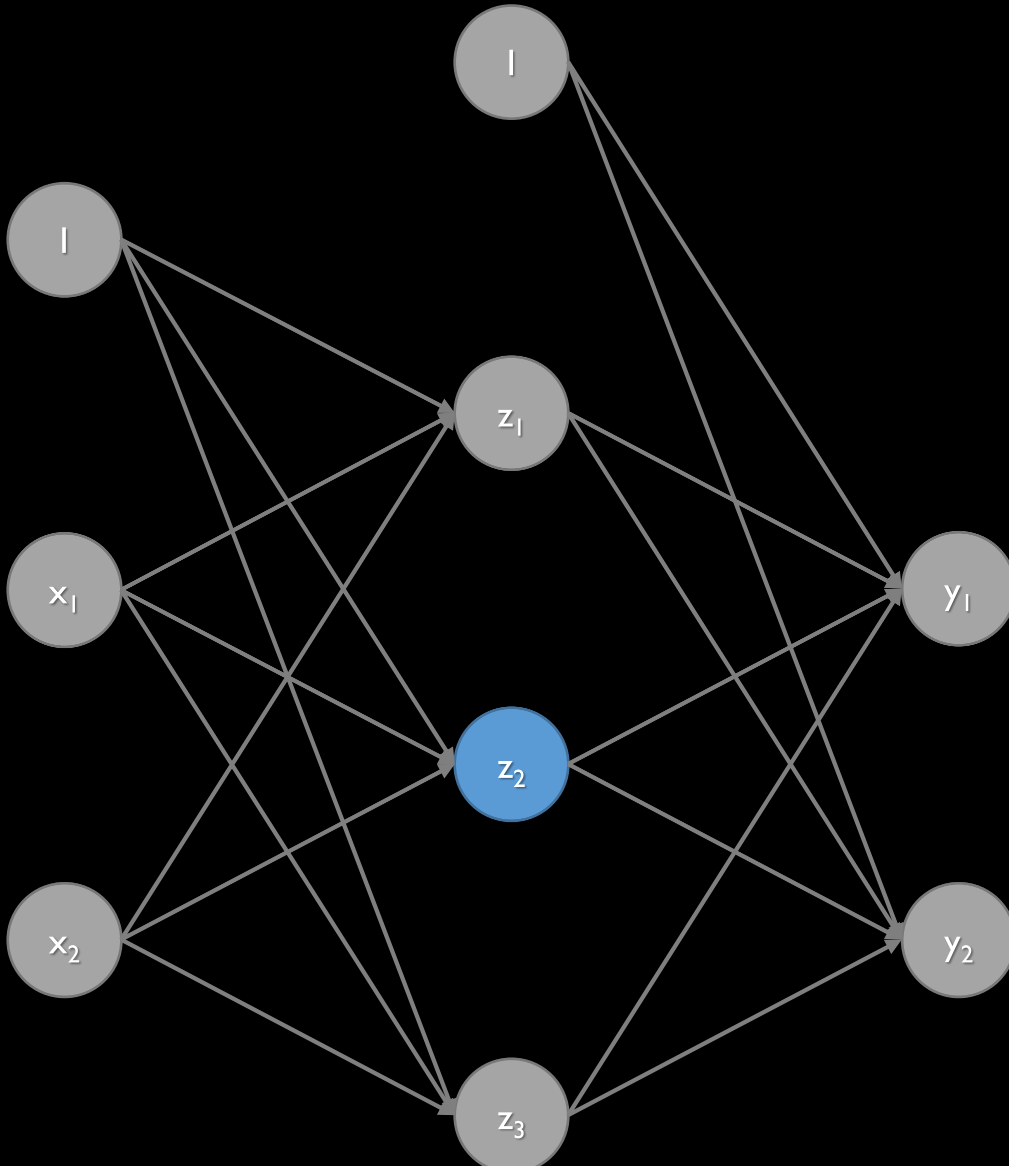


Hitung nilai aktivasi

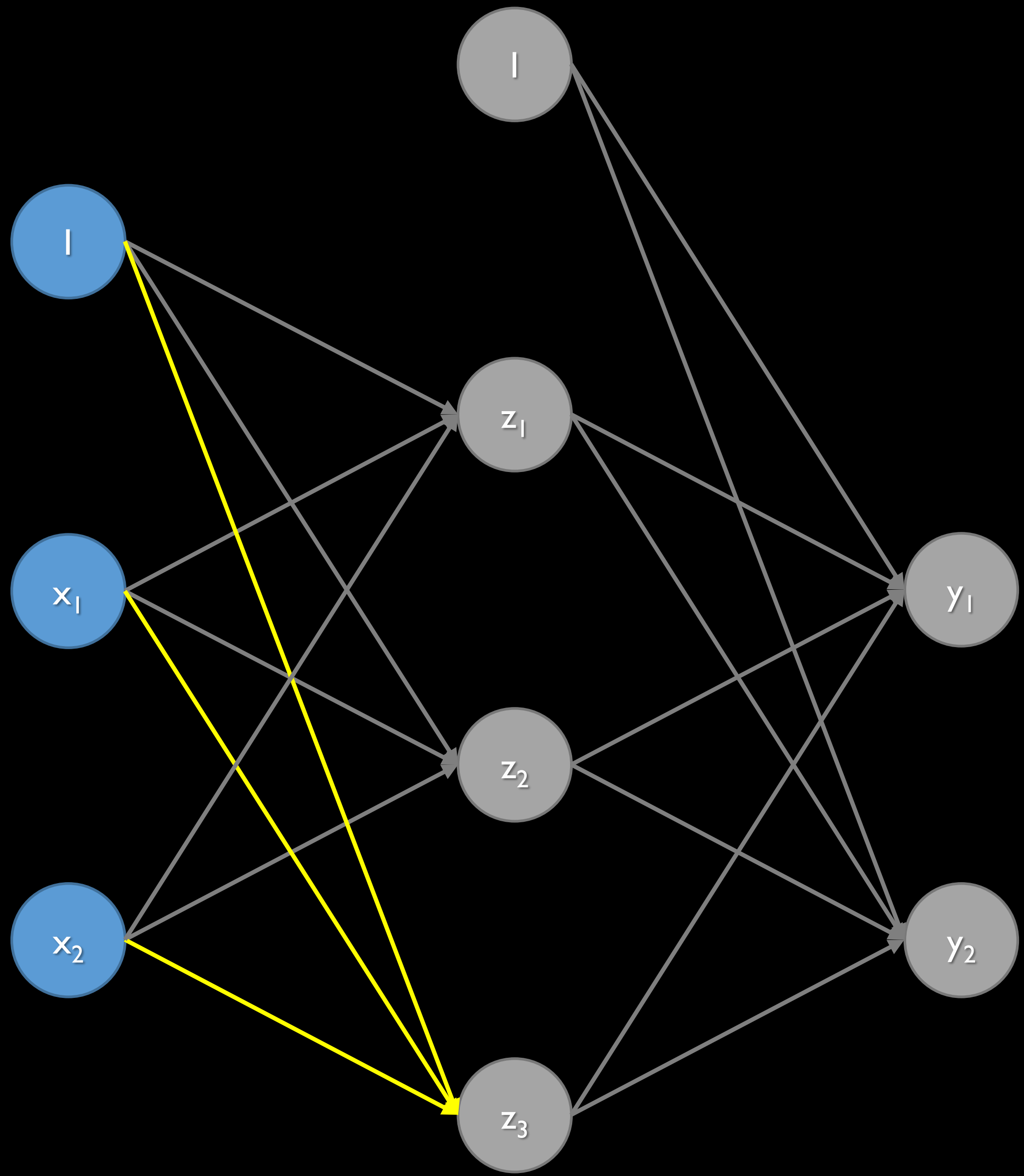
$$z_1 = f(z_{in_1})$$



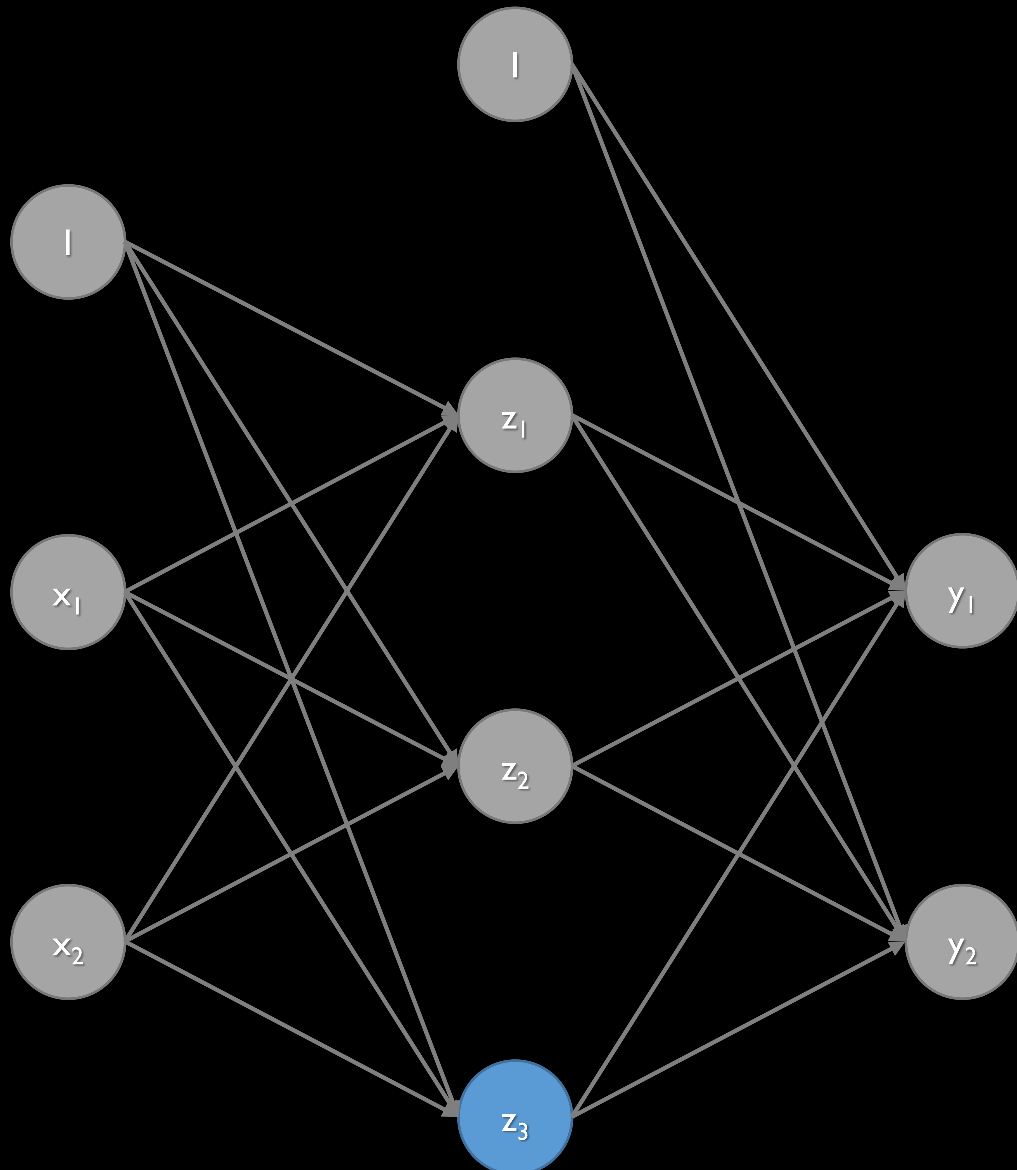
Hitung z_{in_2}



Hitung nilai aktivasi
$z_2 = f(z_{in_2})$

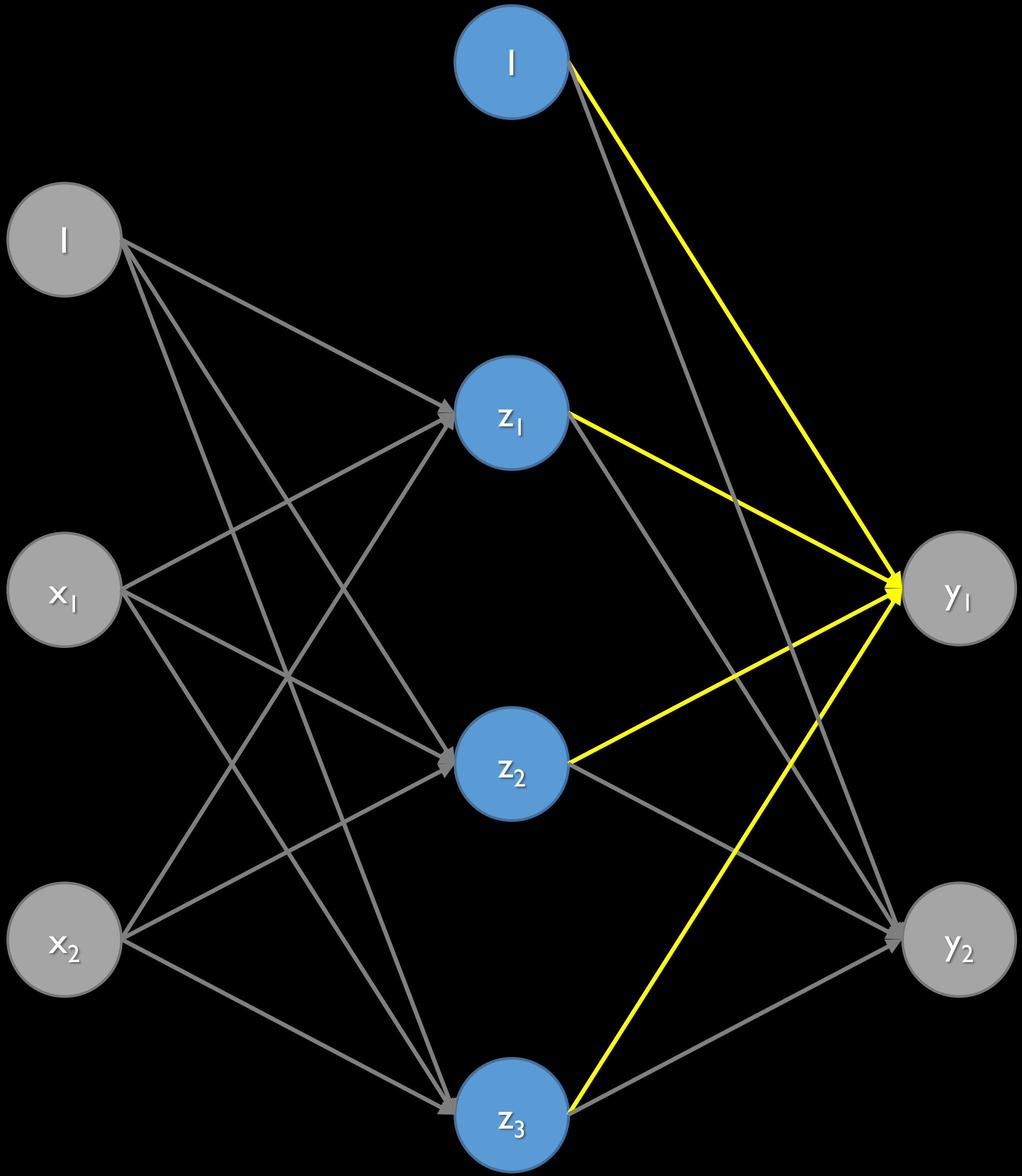


Hitung $z_{in\,3}$

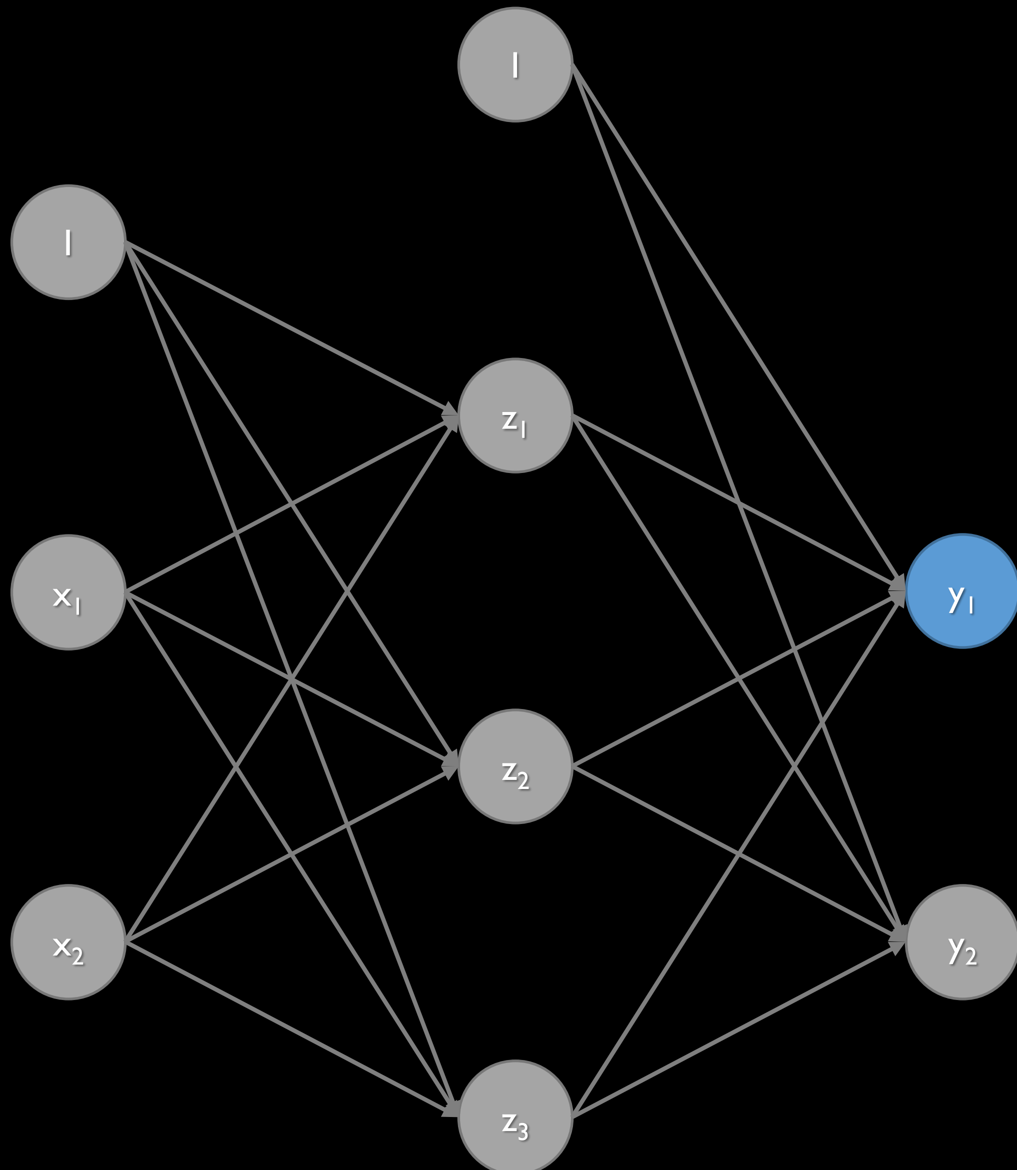


Hitung nilai aktivasi

$$z_3 = f\left(z_{in_3}\right)$$

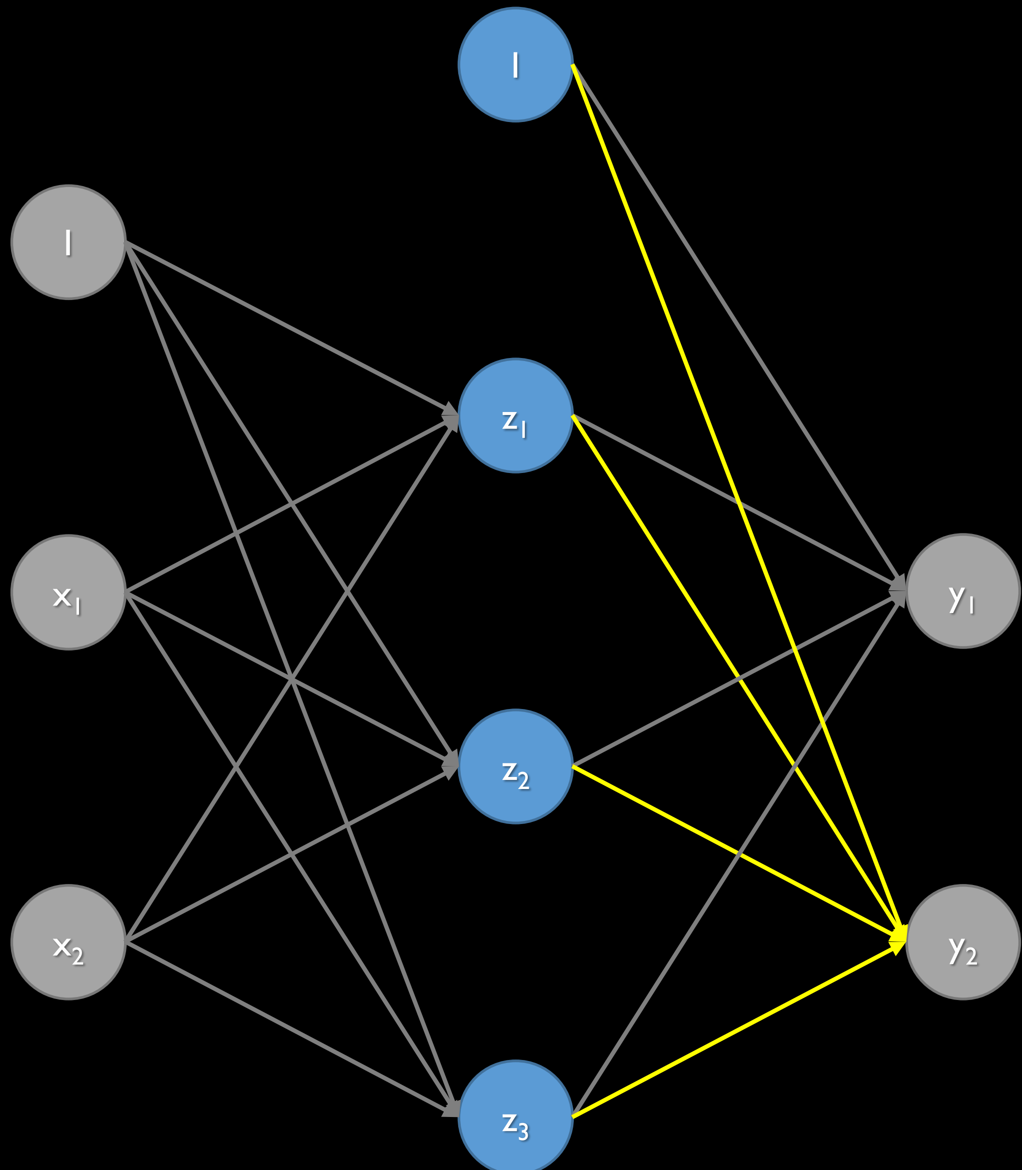


Hitung y_{in_1}

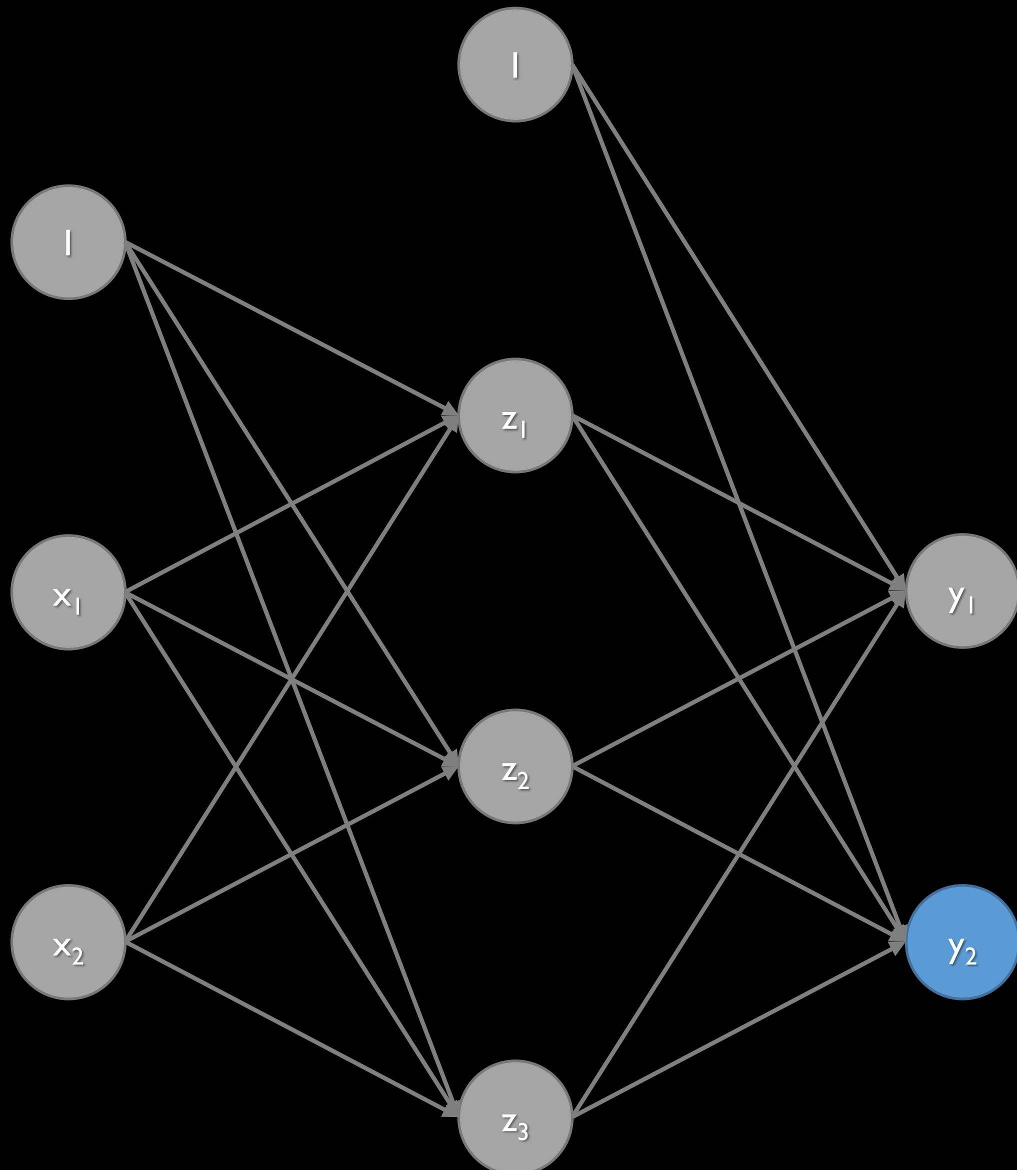


Hitung nilai aktivasi

$$y_1 = f(y_{in_1})$$

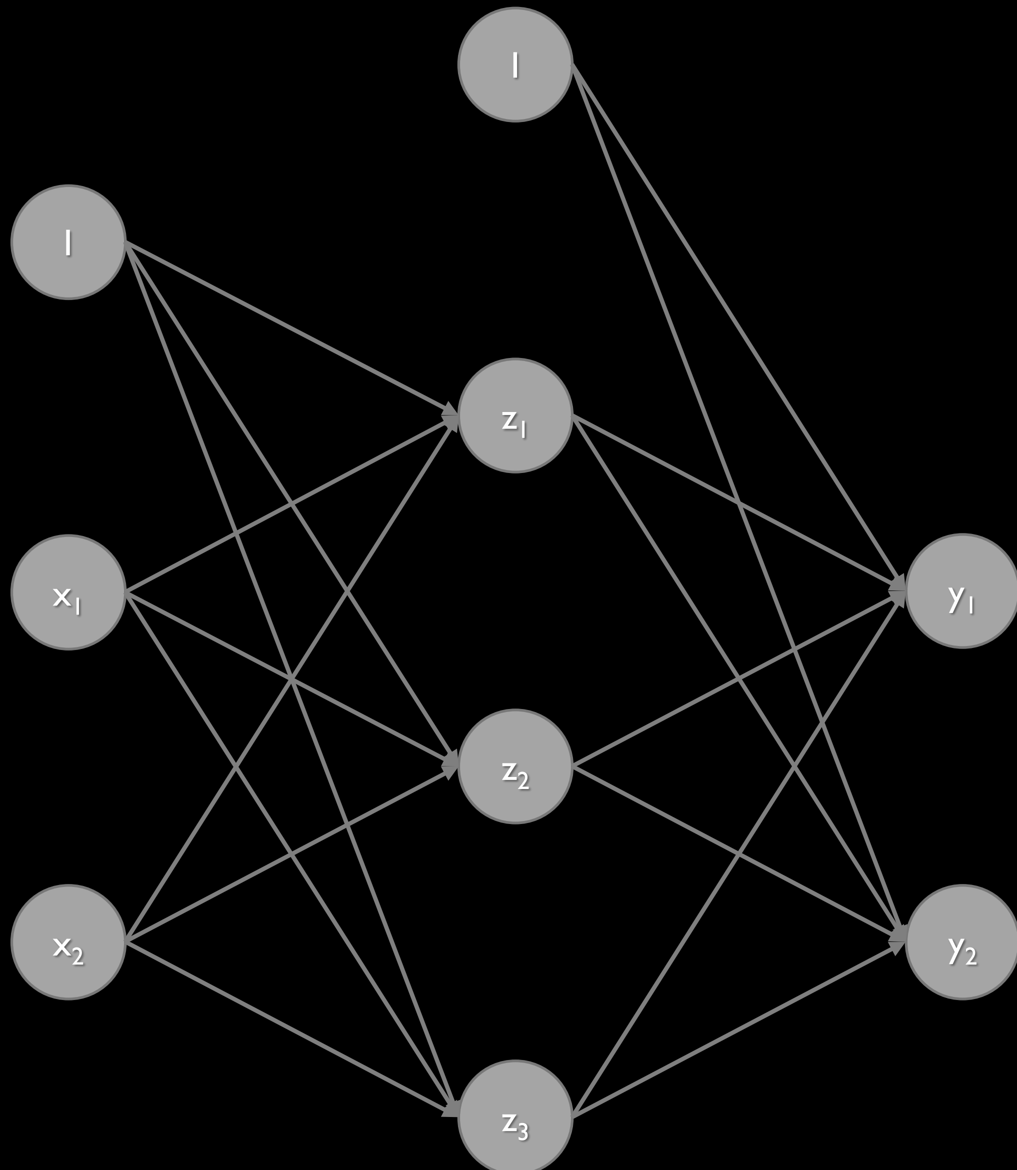


Hitung y_{in_2}

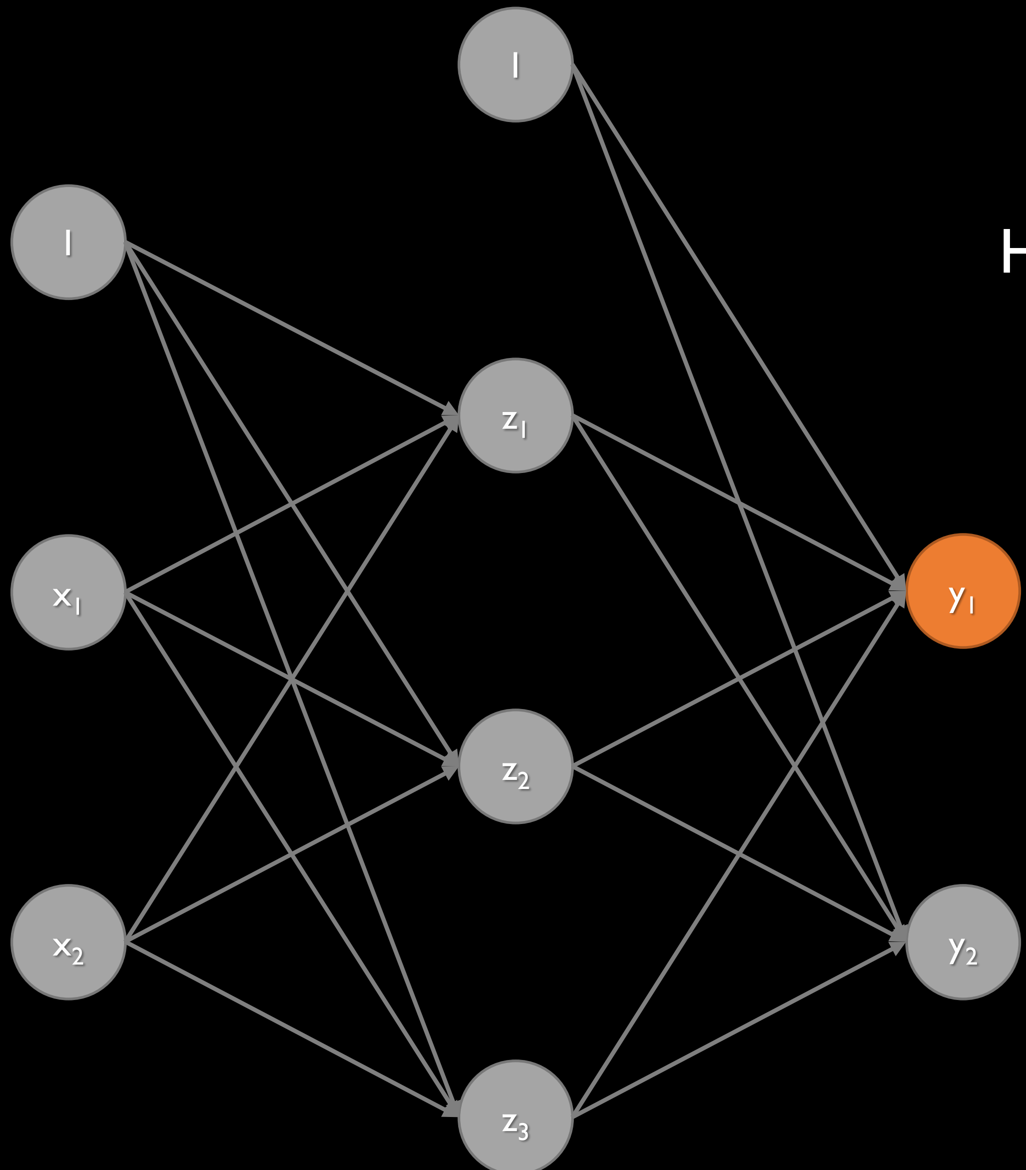


Hitung nilai aktivasi

$$y_2 = f(y_{in_2})$$

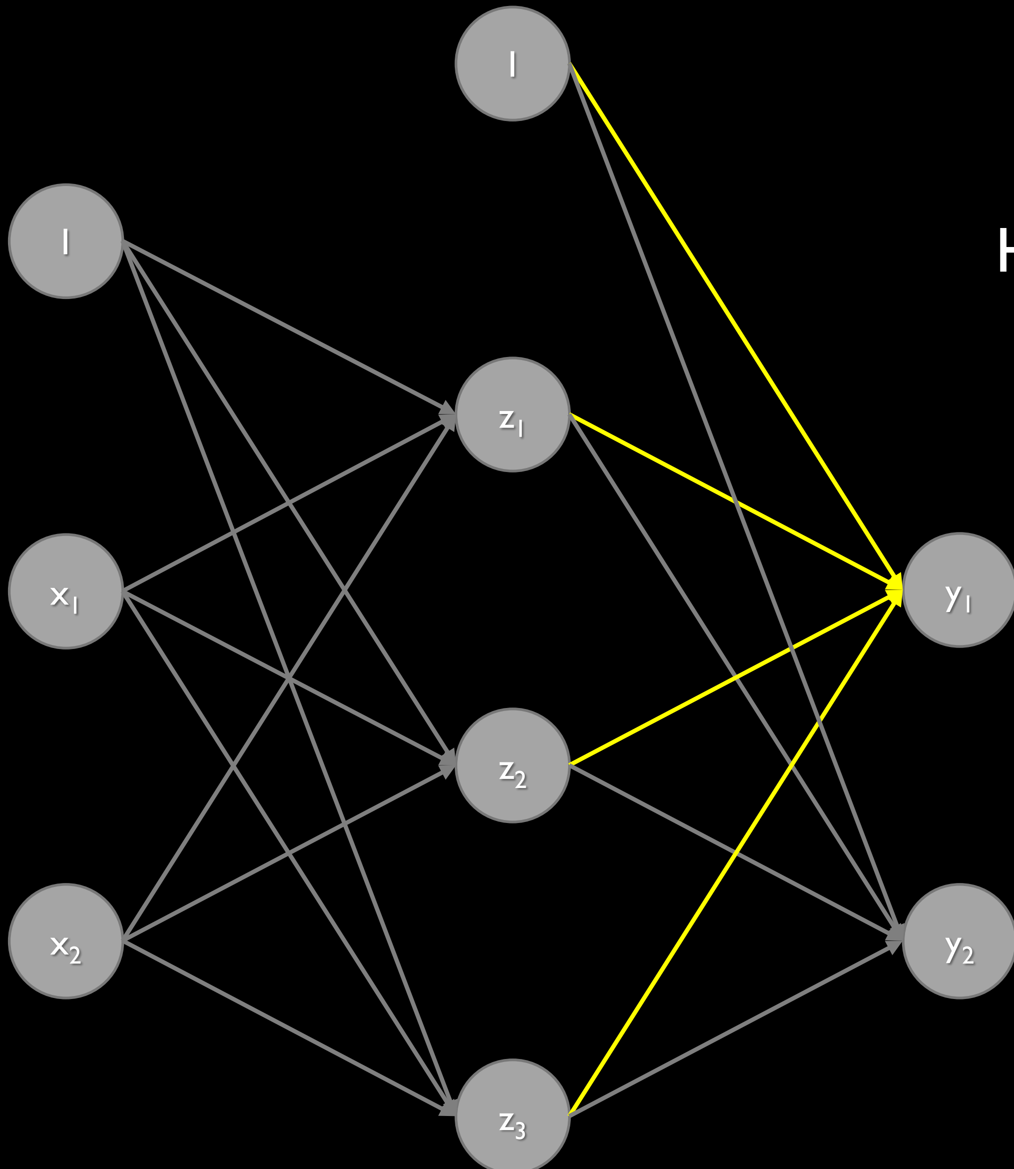


Fase 2:
Backpropagation



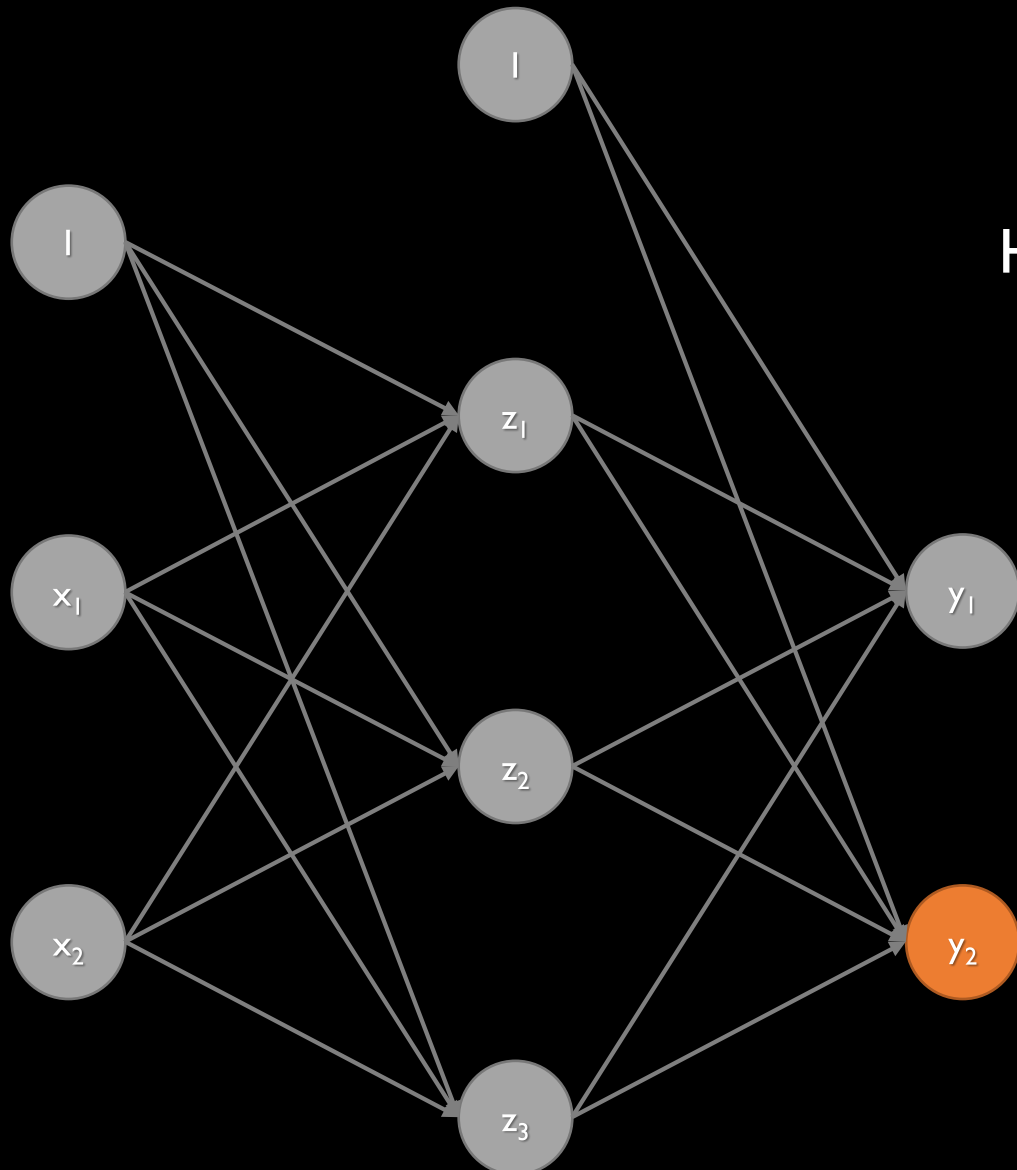
Hitung

$$\delta_1 = (t_1 - y_1) f'(y_{in_1})$$



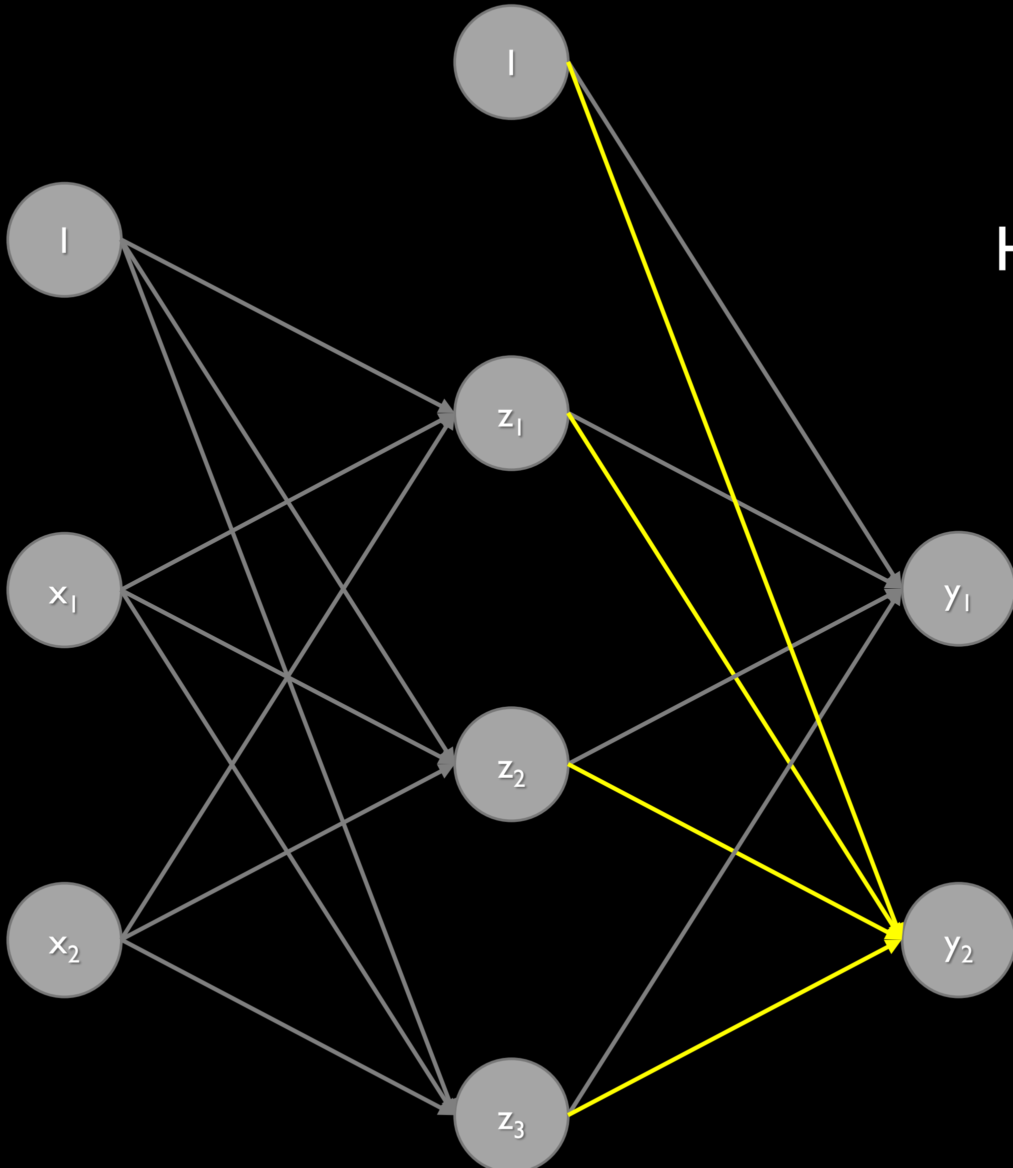
Hitung

$$\Delta w_i = \alpha \delta_1 z_i$$



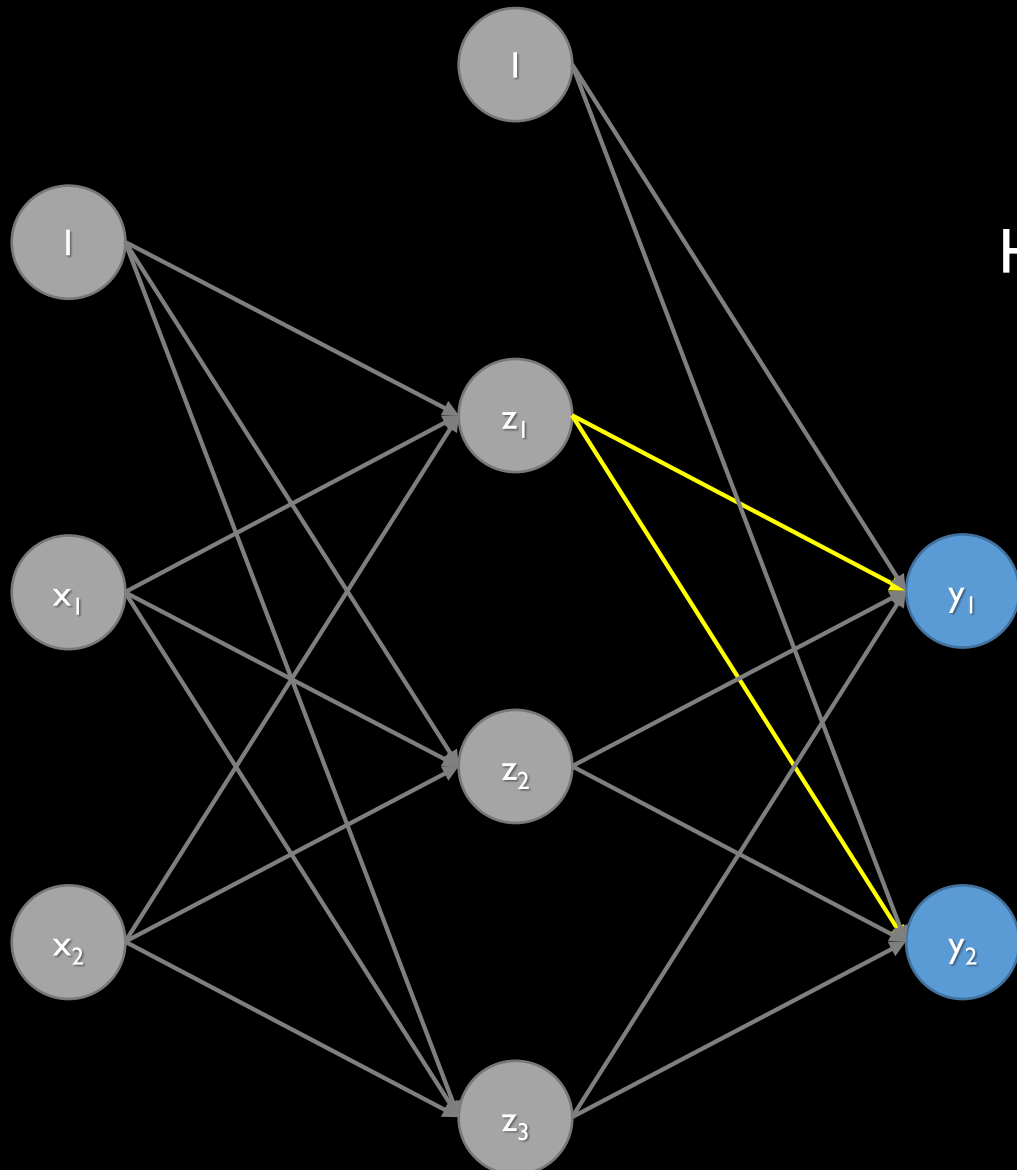
Hitung

$$\delta_2 = (t_2 - y_2) f'(y_{in_2})$$



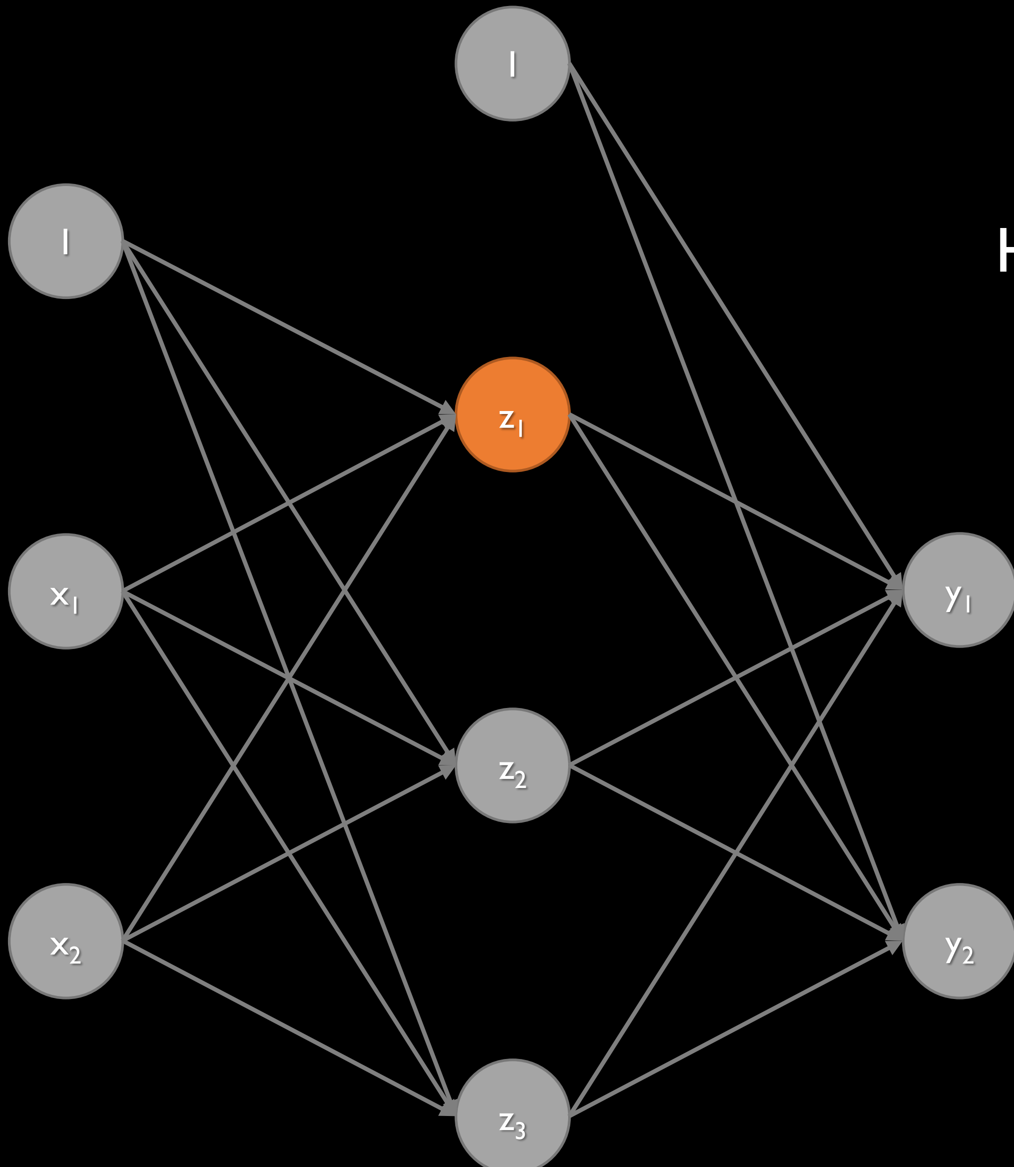
Hitung

$$\Delta w_i = \alpha \delta_2 z_i$$



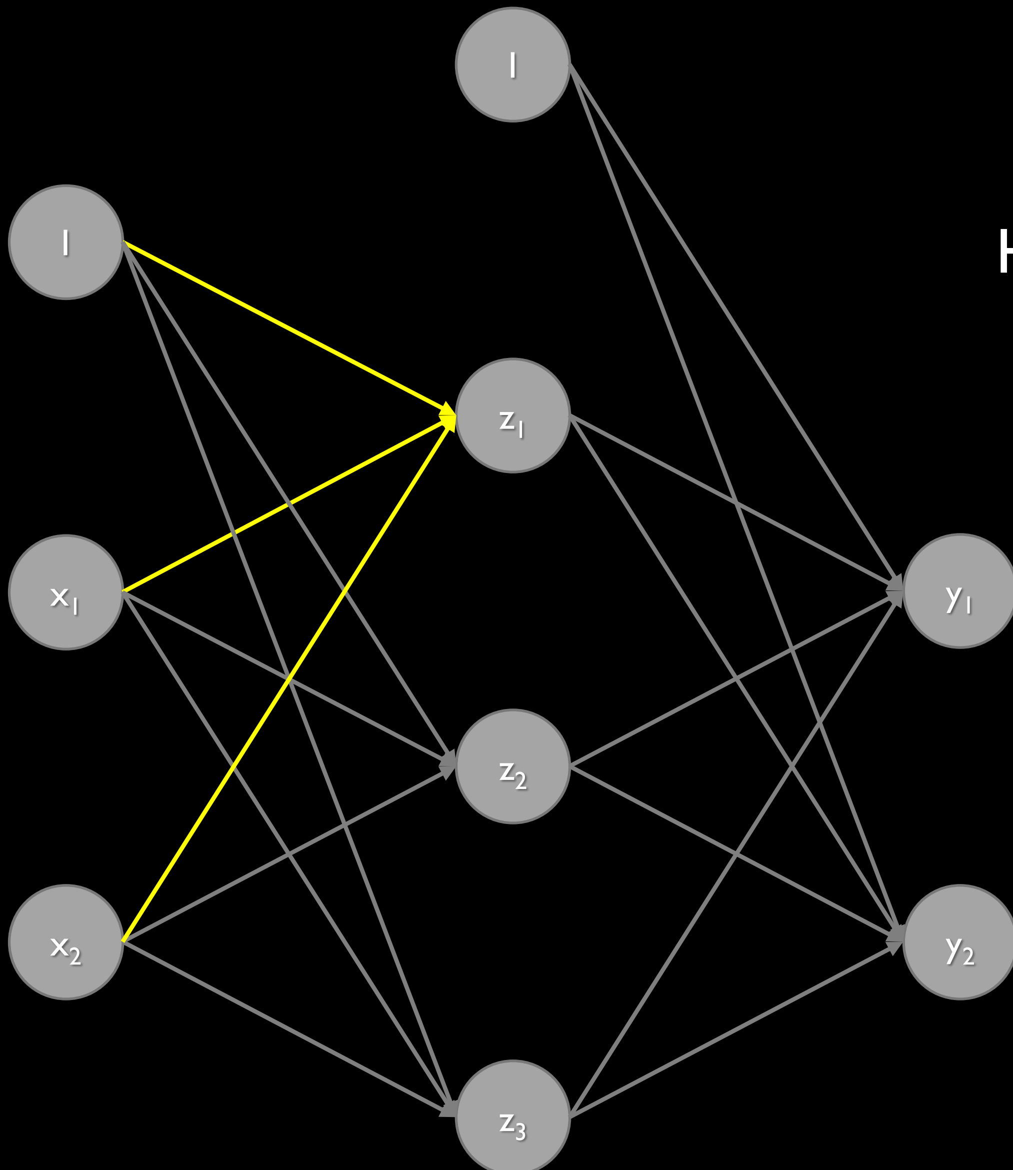
Hitung

$$\delta_{in_1} = \sum_{i=1}^{n} \delta_i w_i$$



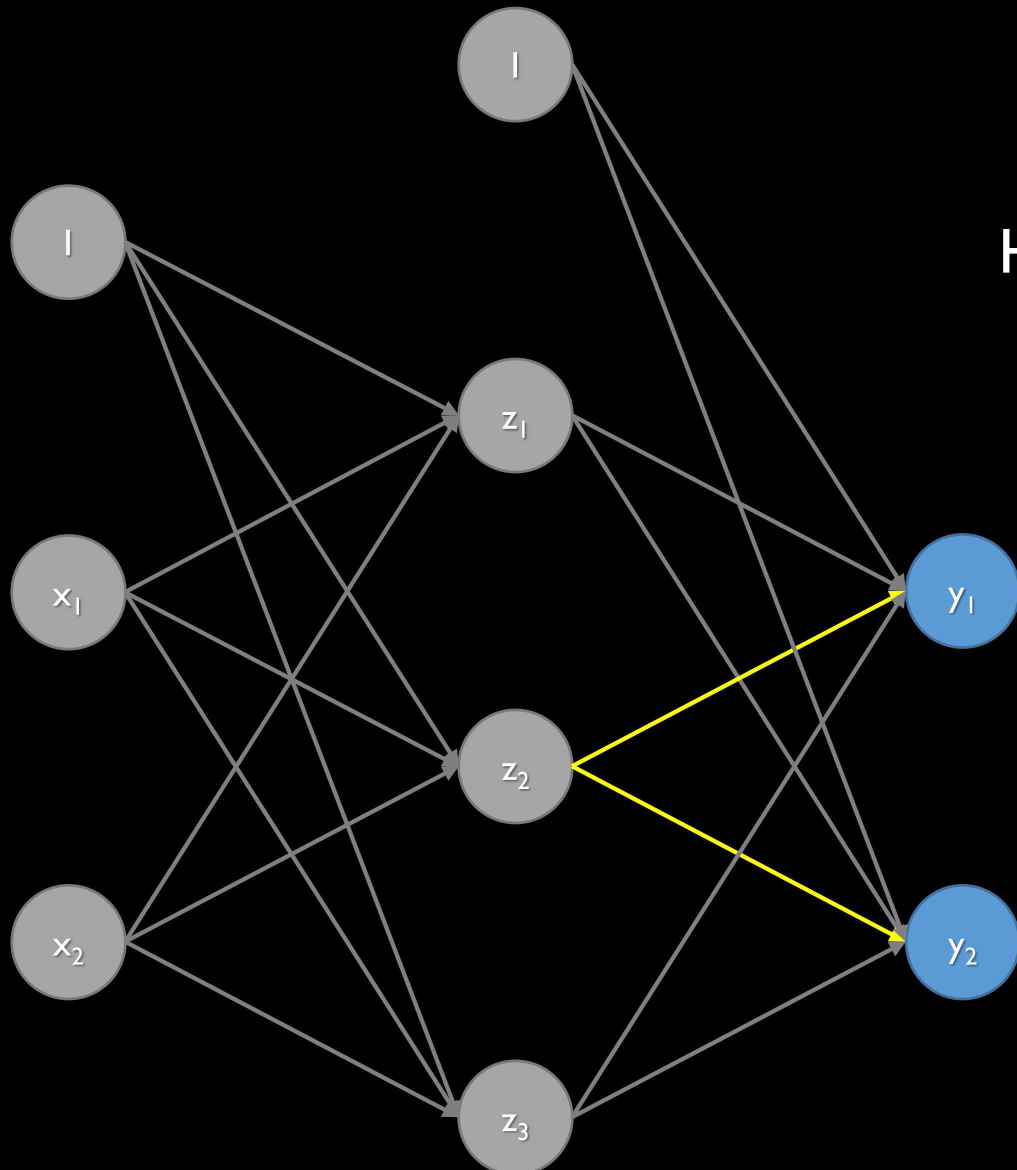
Hitung

$$\delta_1 = \delta_{in_1} f'(z_{in_1})$$



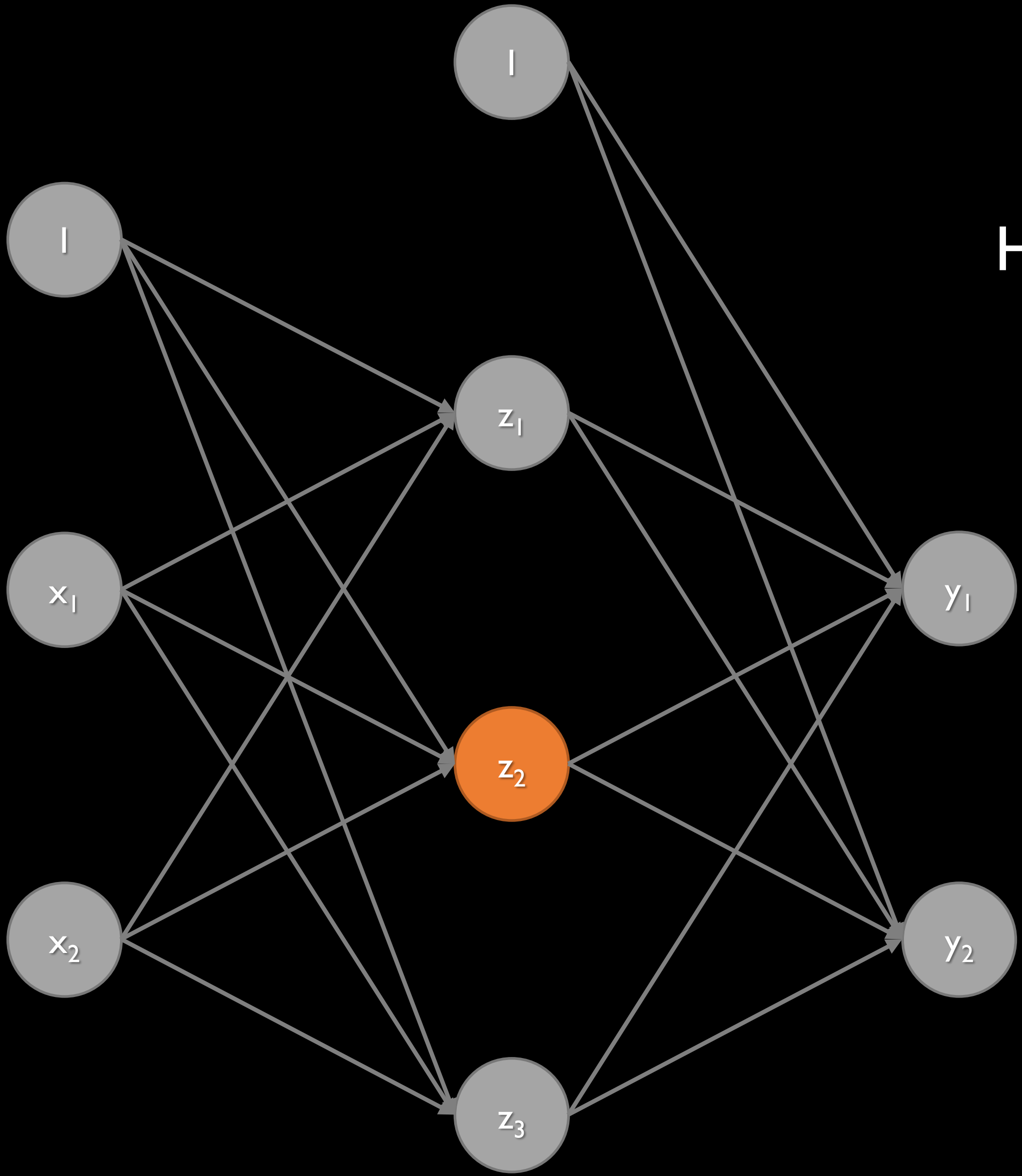
Hitung

$$\Delta w_i = \alpha \delta_1 x_i$$



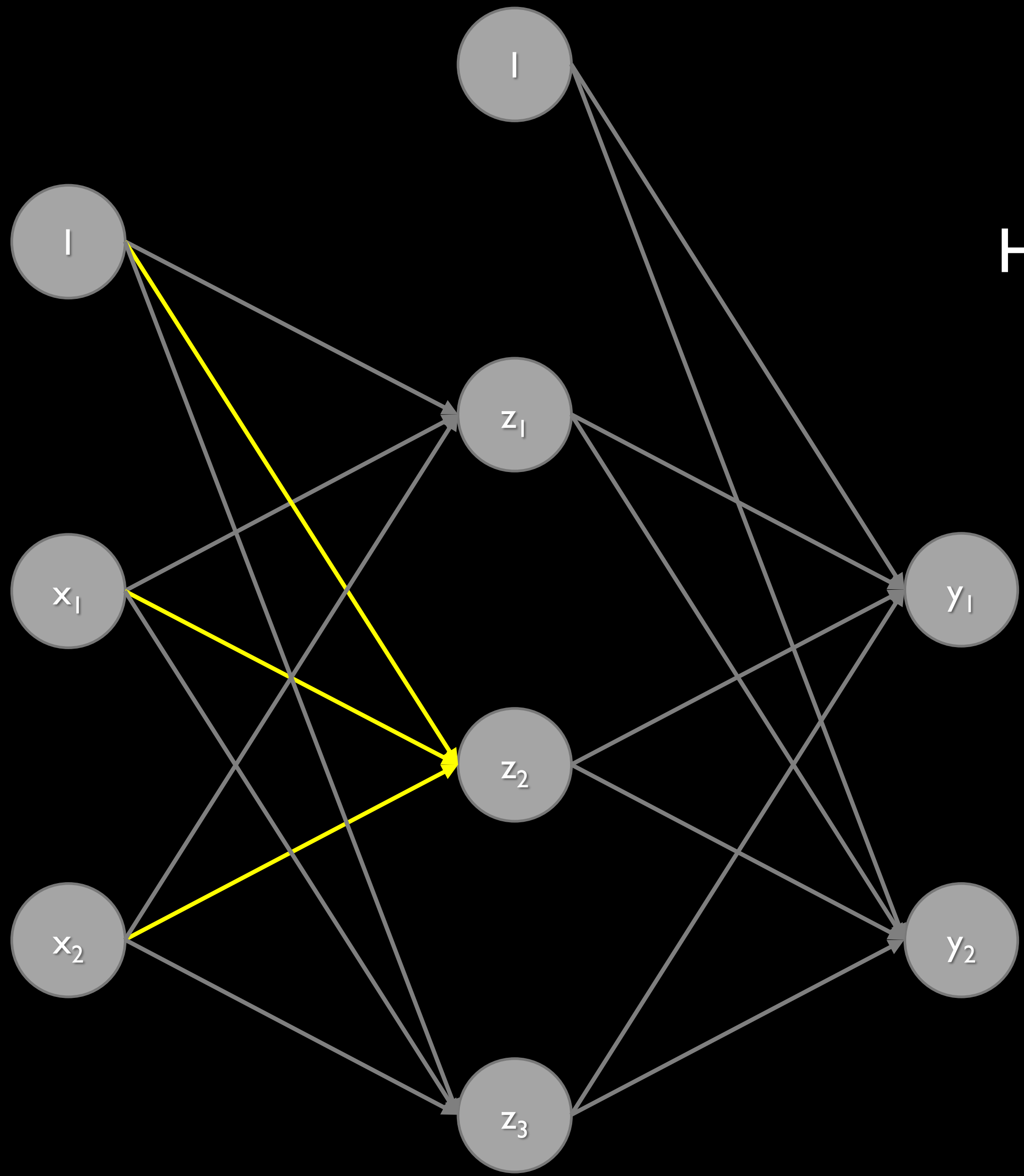
Hitung

$$\delta_{in_2} = \sum_{i=1}^{n} \delta_i w_i$$



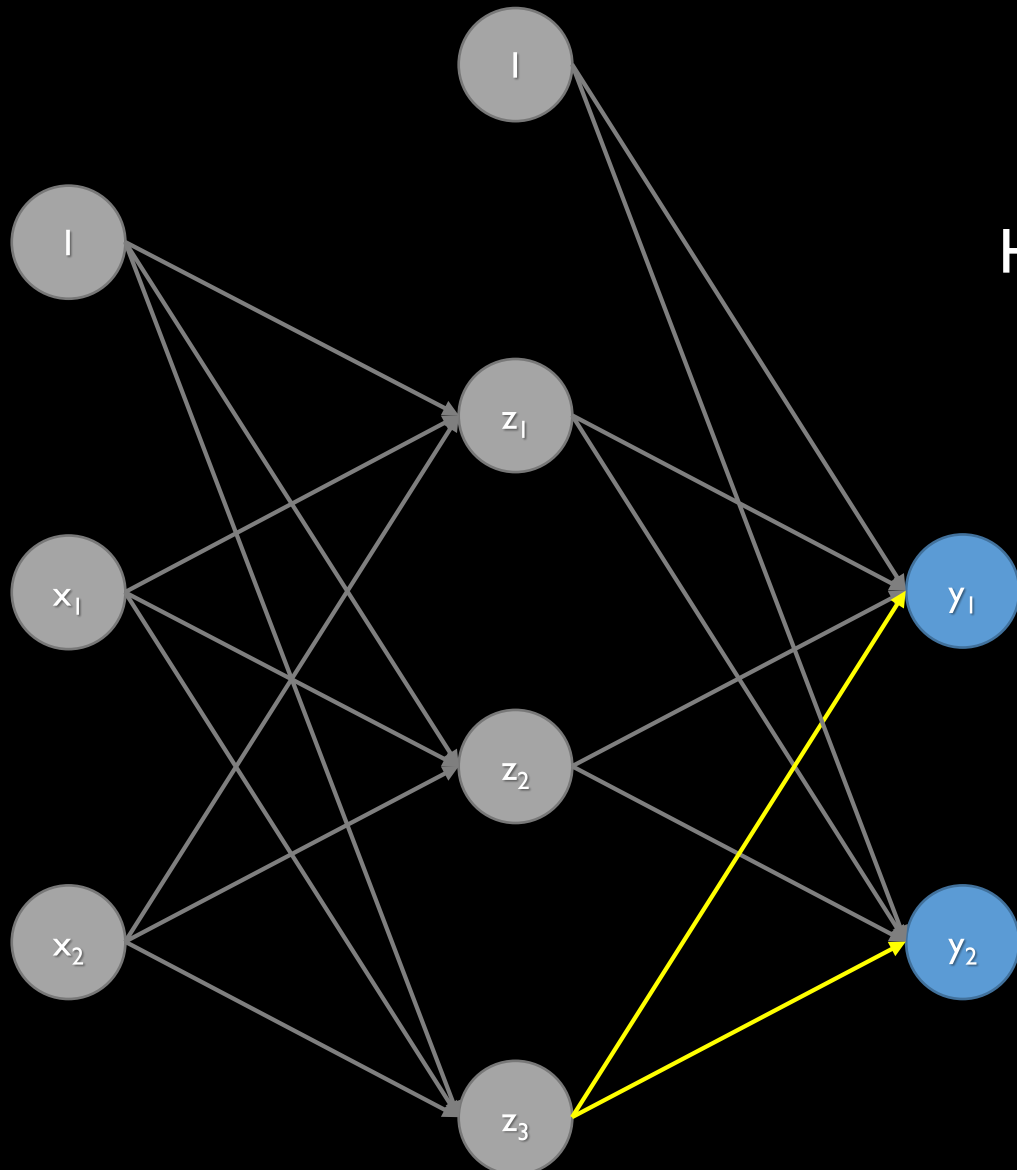
Hitung

$$\delta_2 = \delta_{in_2} f'(z_{in_2})$$



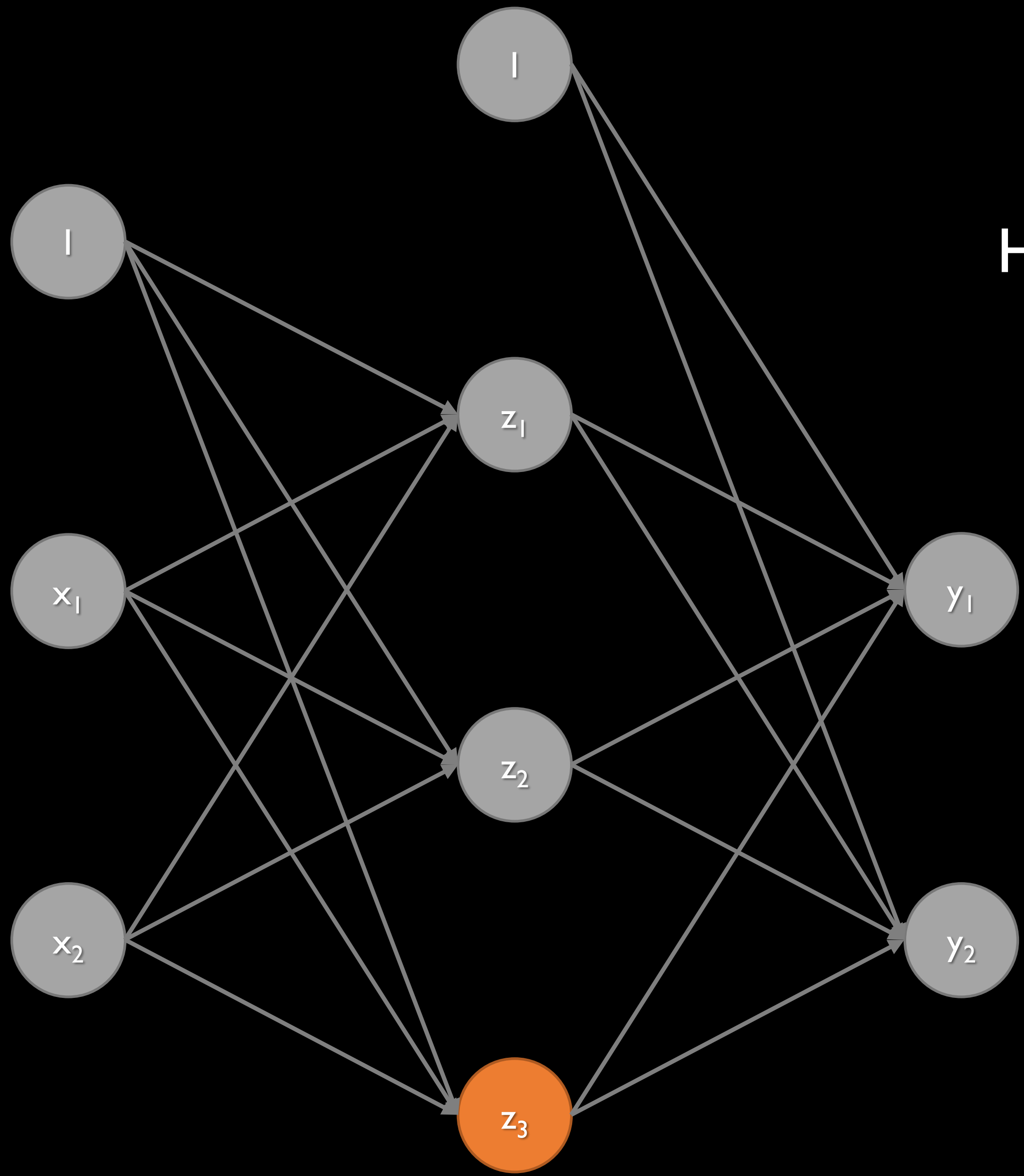
Hitung

$$\Delta w_i = \alpha \delta_2 x_i$$



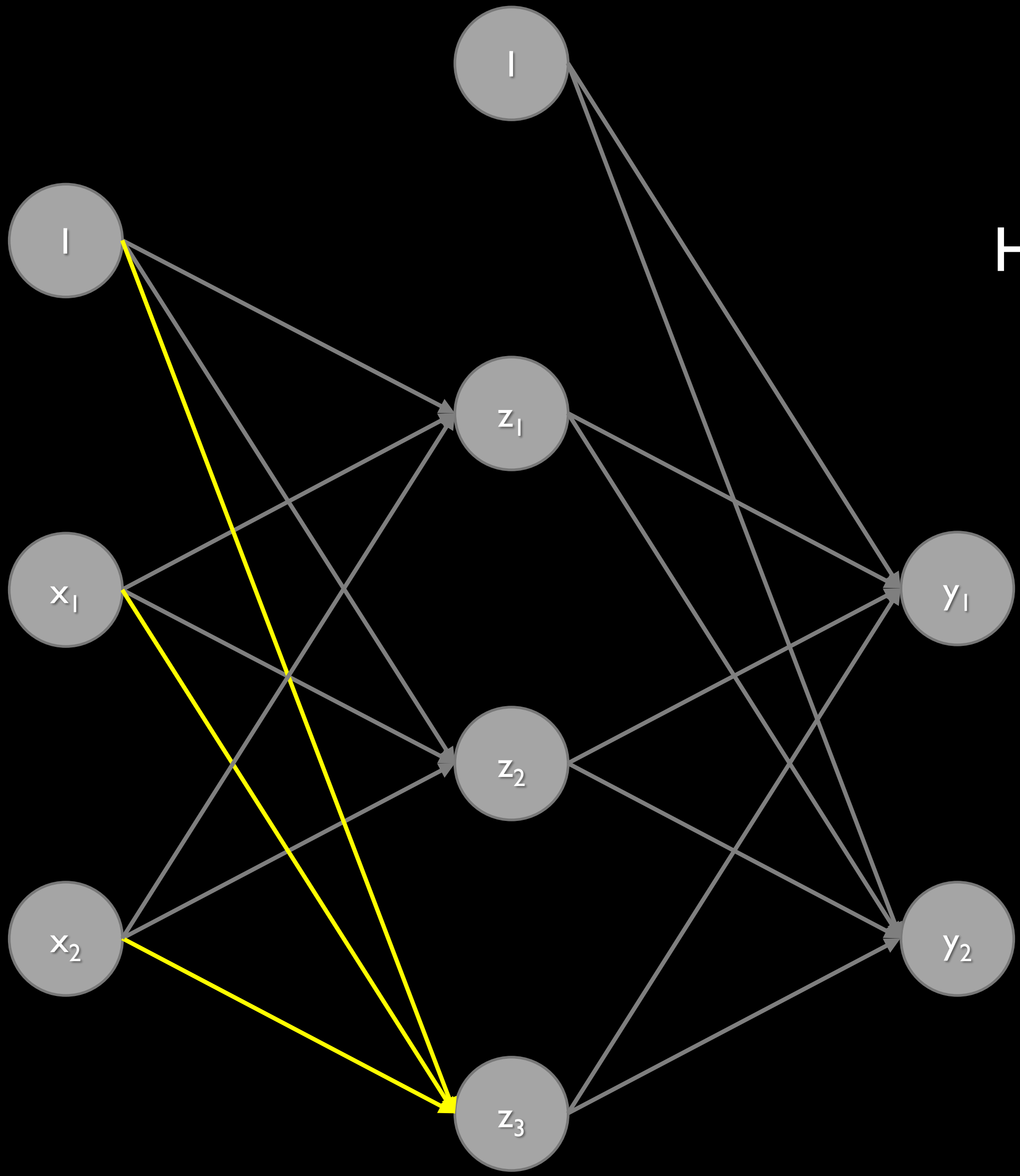
Hitung

$$\delta_{in_3} = \sum_{i=1}^{n} \delta_i w_i$$



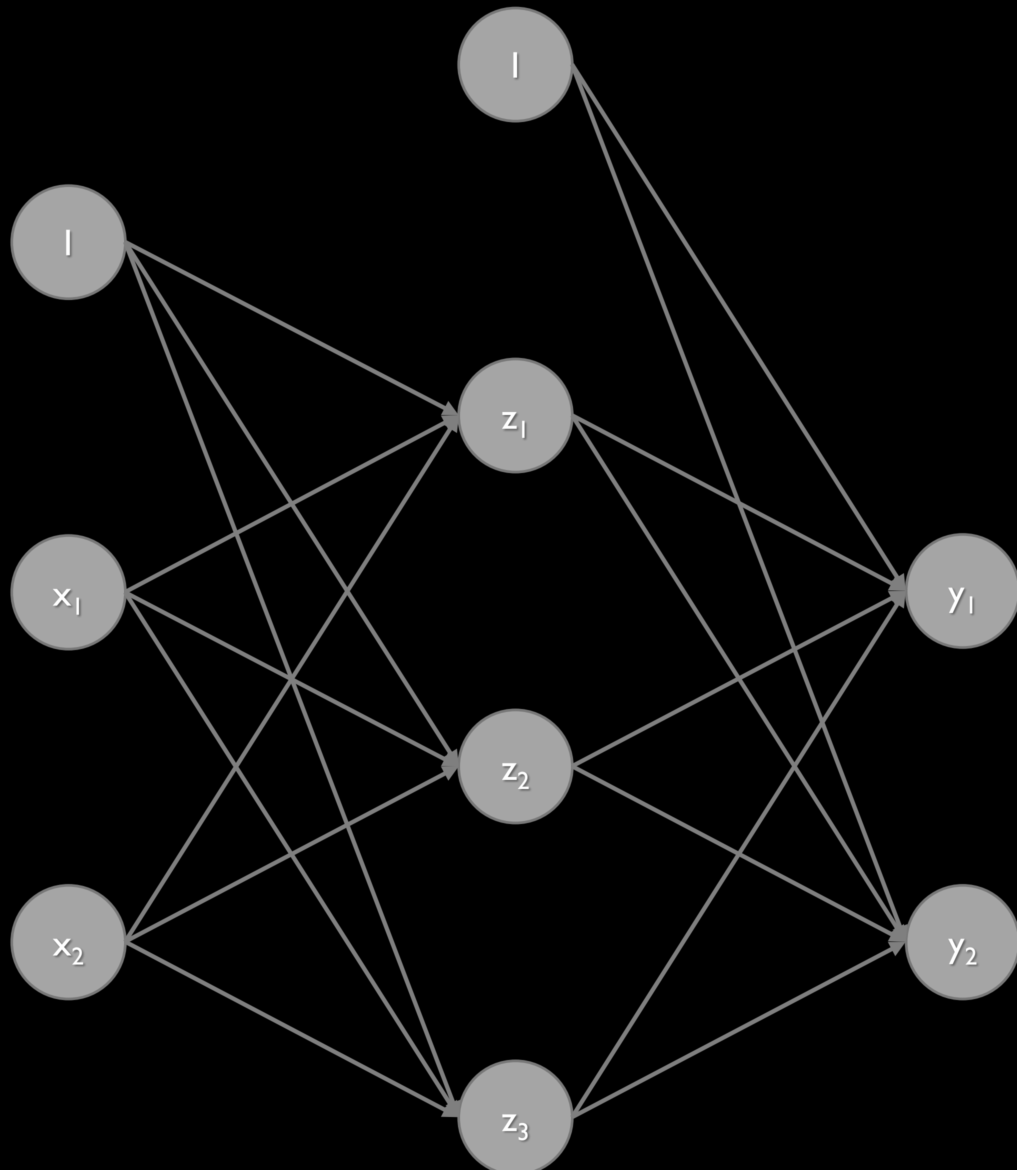
Hitung

$$\delta_3 = \delta_{in_3} f'(z_{in_3})$$

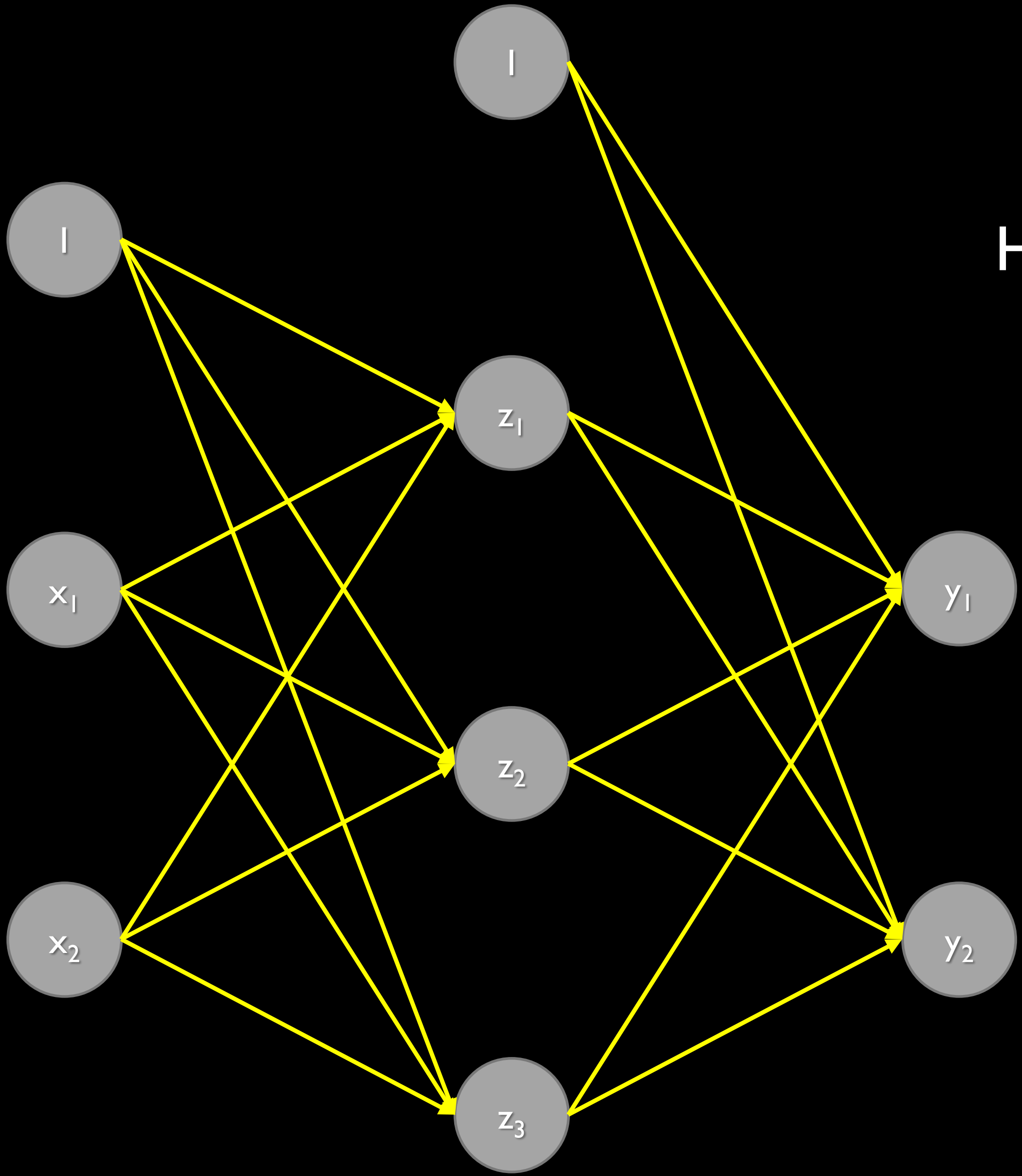


Hitung

$\Delta w_i = \alpha \delta_3 x_i$

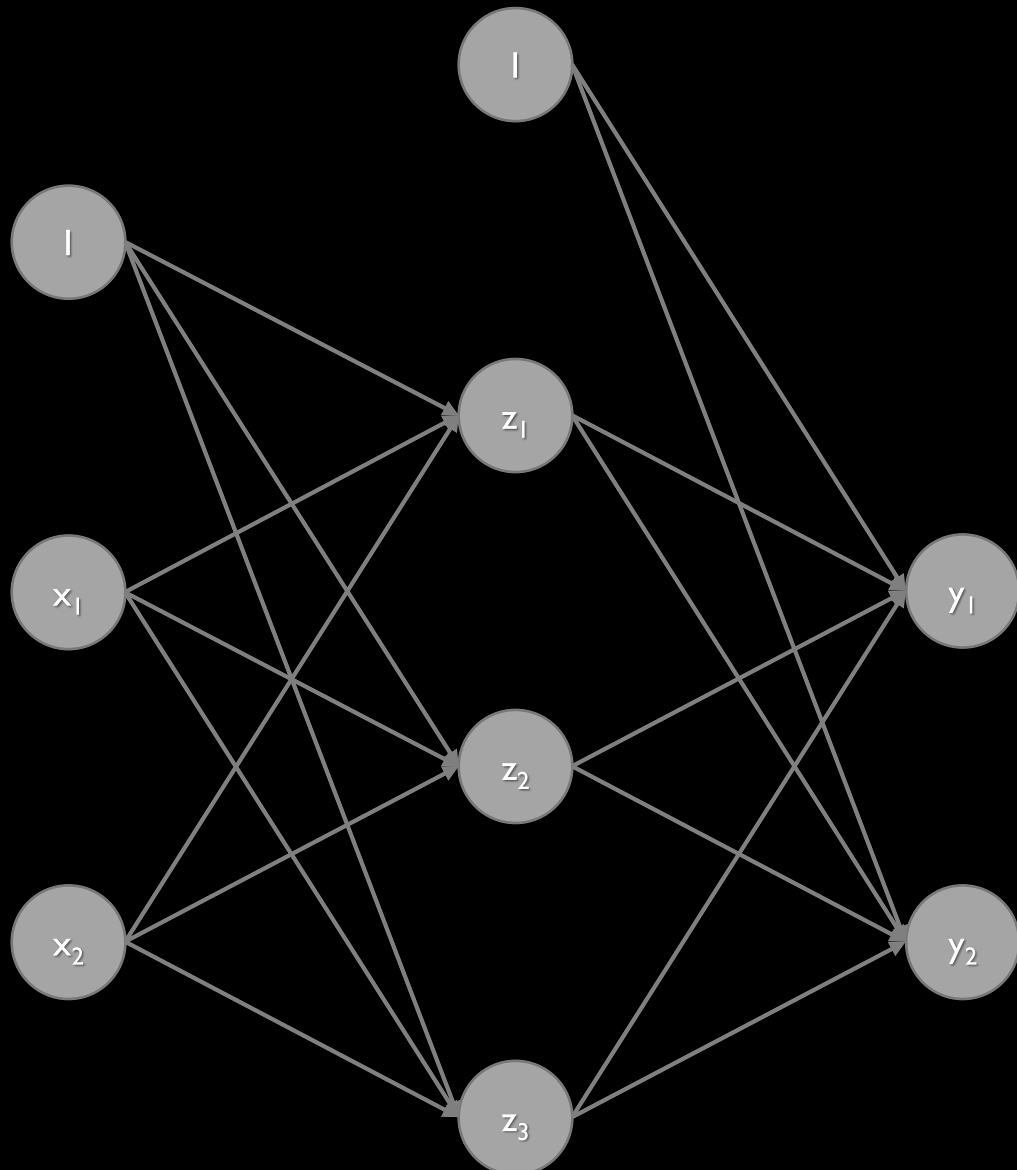


Fase 3:
Update nilai bobot

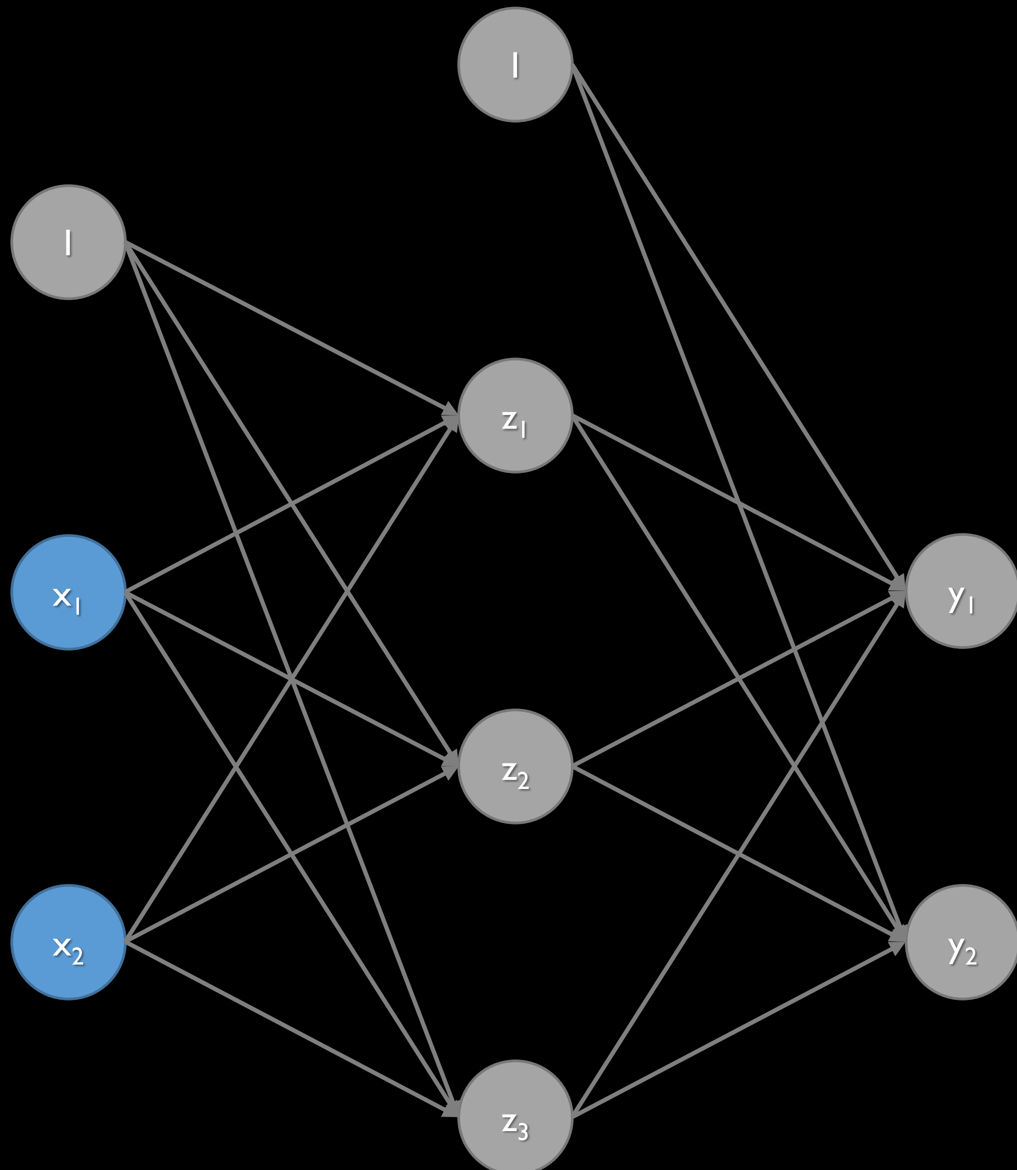


Hitung

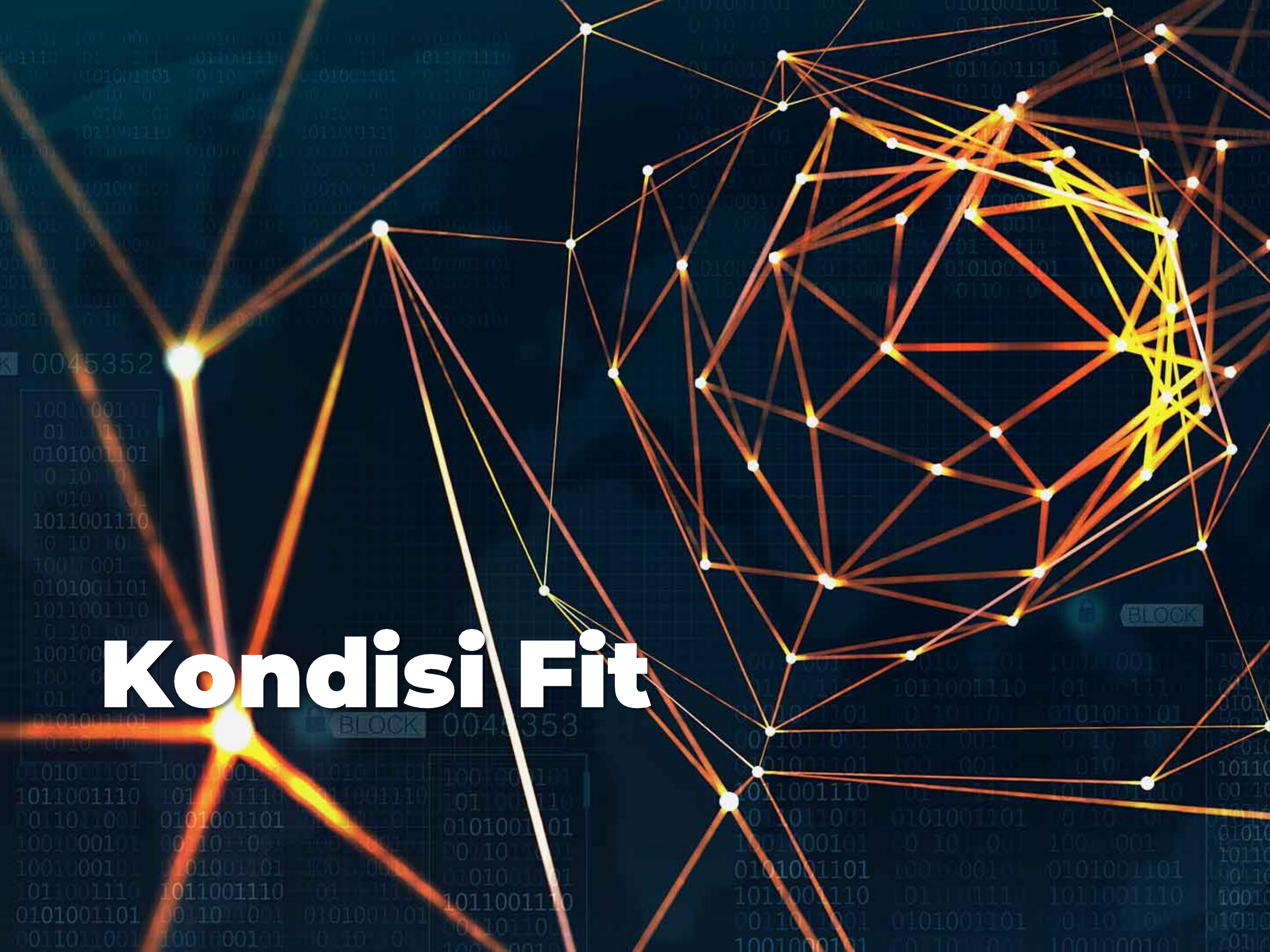
$$w'_{ij} = w_{ij} + \Delta w_{ij}$$



Fase 1:
Feedforward



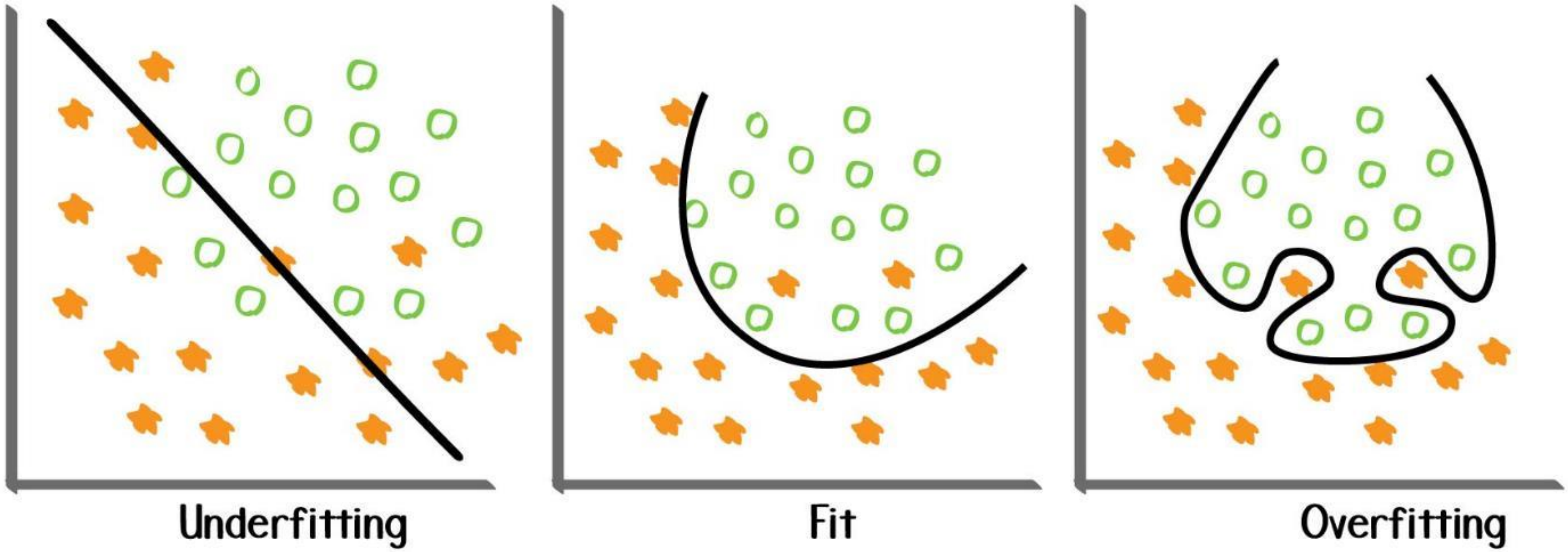
Data latih 2 masuk



Kondisi Fit

Kondisi Fit

- Proses pelatihan/*training*/*fitting* menghasilkan nilai-nilai bobot sesuai dengan data latih
- Proses pelatihan yang terlalu lama akan menghasilkan nilai-nilai bobot yang terlalu spesifik terhadap data latih, yang disebut dengan *overfitting*
- Jika digunakan untuk mengklasifikasi data uji, akurasi akan menjadi kurang baik
- Sebuah metode klasifikasi seharusnya bersifat *general*, tidak spesifik terhadap data latih saja


Underfitting
Fit
Overfitting

Overfitting

- Contoh: klasifikasi jenis kelamin berdasarkan wajah
- *Rule* terbaik/optimal (misal):
  - Pria: rahang lebar, mata besar, hidung lebar
  - Wanita: rahang kecil, mata kecil, hidung kecil
- *Rule* yang dihasilkan oleh JST *overfitting*:
  - Pria: rule terbaik + warna kulit gelap
  - Wanita: rule terbaik + warna kulit cerah

Kondisi Fit

- Maka, proses pelatihan sebaiknya tidak terlalu lama sehingga tidak terlalu spesifik terhadap data latih
- Sehingga klasifikasi data uji akan menghasilkan akurasi yang baik

Kondisi Fit

Beberapa cara untuk menghindari *overfitting* (tidak selalu berlaku):

- Tambah data latih
- Gunakan arsitektur yang tidak terlalu kompleks
- Gunakan *error* maks. yang tidak terlalu rendah
- Batasi jumlah *epoch*
- Reduksi dimensi data (seleksi ciri/*feature*)

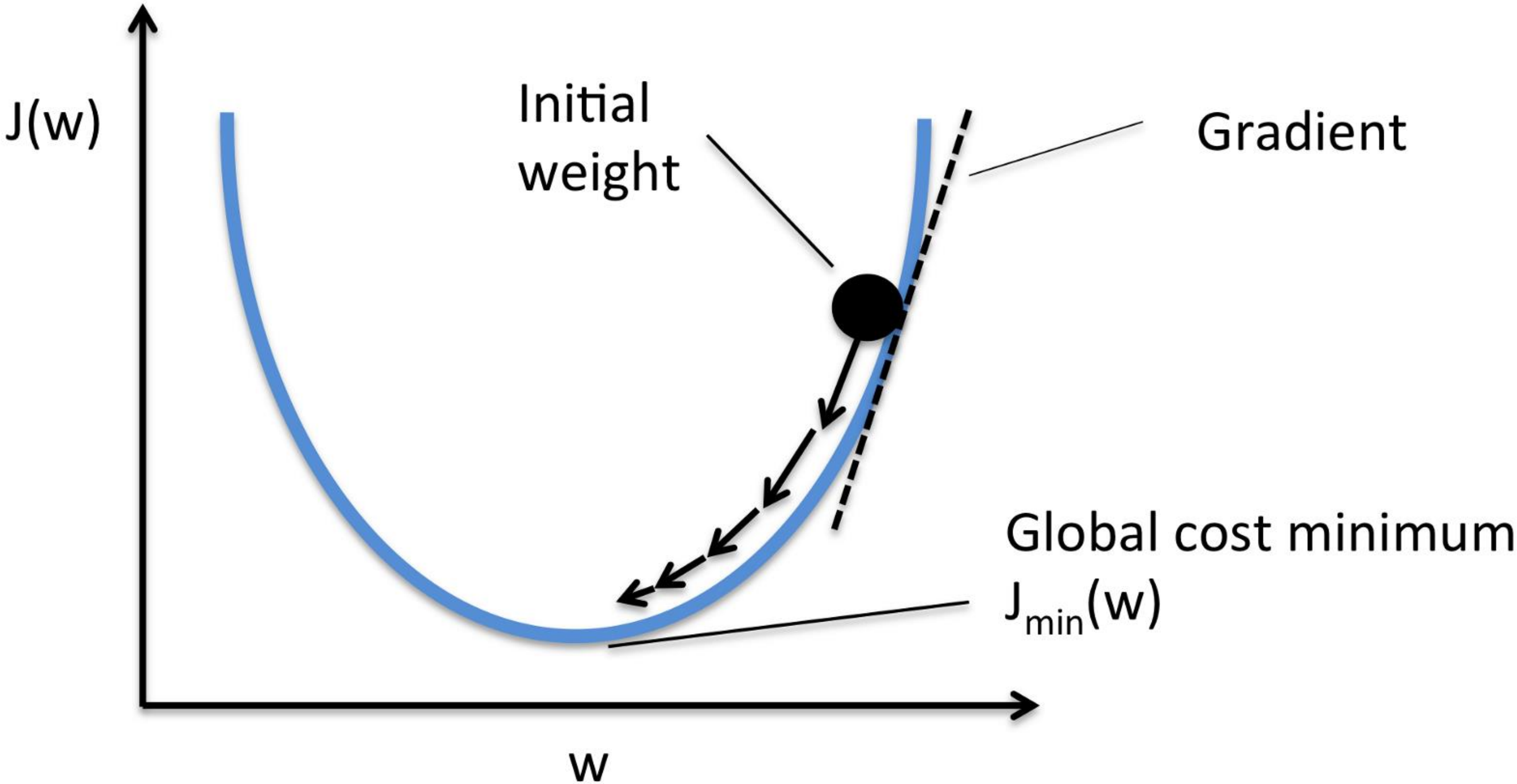

Gradient Descent

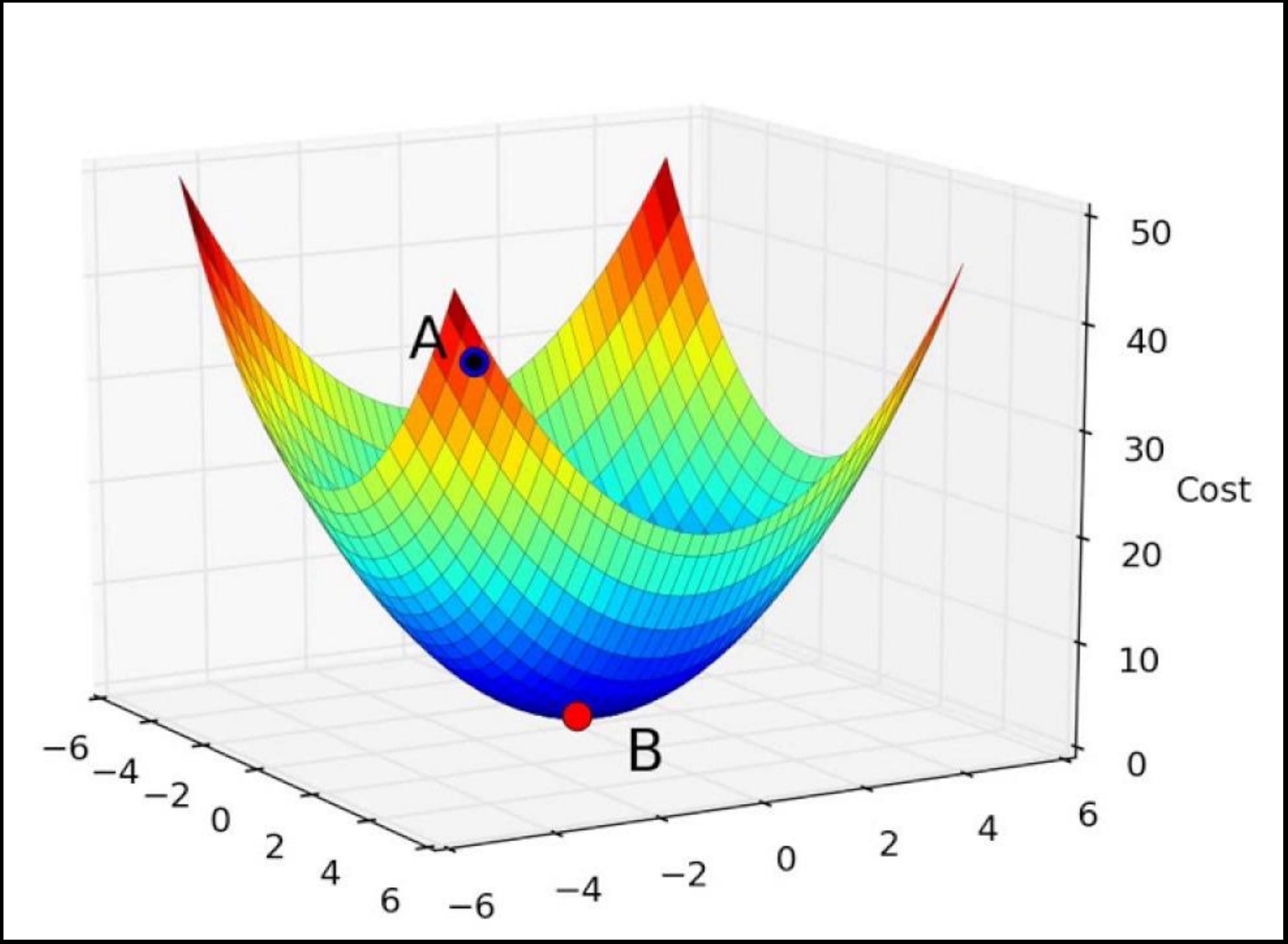
Gradient Descent

- Proses pelatihan/*training* mencari nilai-nilai bobot optimal yang meminimalkan *error/loss/residual/cost*/selisih antara nilai *output* JST dan nilai target
- Pencarian nilai-nilai bobot tersebut dapat saja dilakukan secara *brute force*, namun tentu saja pelatihan akan menjadi tidak terarah

Gradient Descent

- Gradient descent adalah algoritme yang meminimalkan *error* secara terarah menuju ke titik konvergen
- *Gradient/slope*/kemiringan dari *cost function* (mis. MSE) digunakan untuk menurunkan *error* secara bertahap berdasarkan *learning rate*
- *Gradient* dari sebuah fungsi diketahui dari diferensial/turunan dari fungsi tersebut


J(w)
Initial weight
Gradient
Global cost minimum
$J_{min}(w)$
w

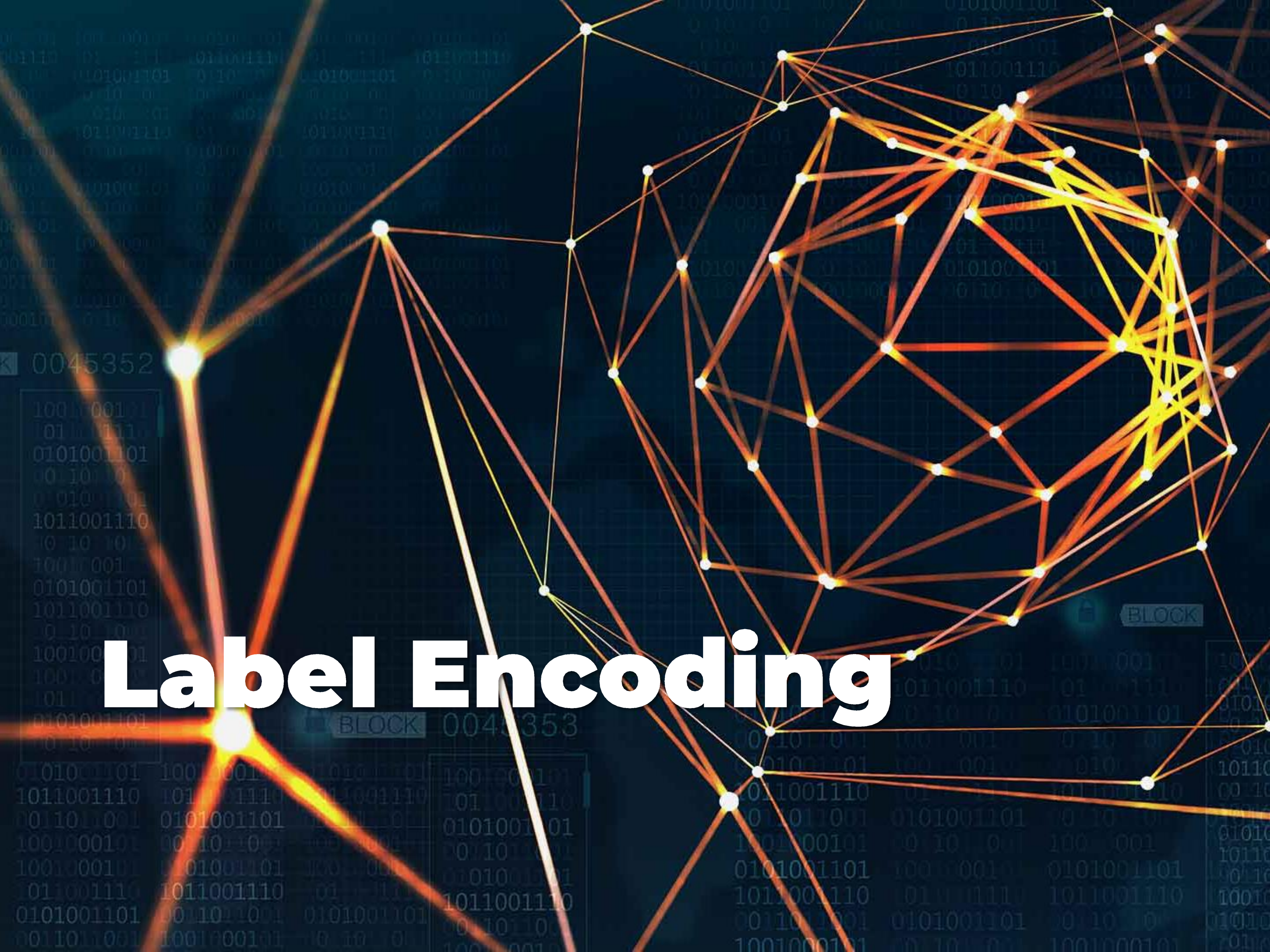


A
B
Cost
50
40
30
20
10
0
−6
−4
−2
0
2
4
6
−6
−4
−2
0
2
4
6

Update Nilai Bobot

Ada beberapa strategi untuk *update* nilai bobot:

- Batch gradient descent: update dilakukan setelah <u>semua</u> data latih diproses
- Mini-batch gradient descent: update dilakukan setelah setiap <u>beberapa</u> data latih diproses
- Stochastic gradient descent: update dilakukan setelah pada <u>setiap</u> data latih



Label Encoding

Label Encoding

- Label/kelas dapat direpresentasikan (*encode*) dalam beberapa pola: biner dan one-hot

Pola Biner

Contoh:

- Kelas: `[1, 2, 3]`

  Pola biner:

```
[[0, 0],
 [0, 1],
 [1, 0]]
```

Pola Biner

Contoh:

- Kelas: `[1, 2, 3, 4]`

  Pola biner:

```
[[0, 0, 0],
 [0, 0, 1],
 [0, 1, 0],
 [0, 1, 1]]
```

Pola Biner

```
def bin_enc(lbl):
    mi = min(lbl)
    length = len(bin(max(lbl) - mi + 1)[2:])
    enc = []

    for i in lbl:
        b = bin(i - mi)[2:].zfill(length)

        enc.append([int(n) for n in b])

    return enc
```

Pola Biner

```
def bin_dec(enc, mi=0):
    lbl = []

    for e in enc:
        rounded = [int(round(x)) for x in e]
        string = ''.join(str(x) for x in rounded)
        num = int(string, 2) + mi

        lbl.append(num)

    return lbl
```

Pola One-Hot

Contoh:

- Kelas: `[1, 2, 3]`
  Pola biner:

```
[[1, 0, 0],
 [0, 1, 0],
 [0, 0, 1]]
```

Pola One-Hot

Contoh:

- Kelas: `[3, 4, 5, 6]`
  Pola biner:

```
[[1, 0, 0, 0]
 [0, 1, 0, 0]
 [0, 0, 1, 0]
 [0, 0, 0, 1]]
```

```
def onehot_enc(lbl, min_val=0):
    mi = min(lbl)
    enc = np.full((len(lbl), max(lbl) - mi + 1), min_val,
np.int8)

    for i, x in enumerate(lbl):
        enc[i, x - mi] = 1

    return enc

def onehot_dec(enc, mi=0):
    return [np.argmax(e) + mi for e in enc]
```

Implementasi

```python
import numpy as np

# Fungsi sigmoid
def sig(X):
    return [1 / (1 + np.exp(-x)) for x in X]


# Turunan dari fungsi sigmoid
def sigd(X):
    for i, x in enumerate(X):
        s = sig([x])[0]

        yield s * (1 - s)
```

```python
def bp_fit(C, X, t, a, mep, mer):
    # nin: neuron input
    nin = [np.empty(i) for i in C]

    # n: neuron
    n = [np.empty(j + 1) if i < len(C) - 1 else np.empty(j) for i,
j in enumerate(C)]

    # w: weight
    w = np.array([np.random.rand(C[i] + 1, C[i + 1]) for i in
range(len(C) - 1)])

    # dw: delta weight
    dw = [np.empty((C[i] + 1, C[i + 1])) for i in range(len(C) -
1)]

    # d: delta
    d = [np.empty(s) for s in C[1:]]

    # din: delta input
    din = [np.empty(s) for s in C[1:-1]]
```

```
# din: delta input
din = [np.empty(s) for s in C[1:-1]]

# ep: epoch
ep = 0

# mse: mean square error
mse = 1

# Inisialisasi bias dengan nilai 1
for i in range(0, len(n) - 1):
   n[i][-1] = 1
```

```python
# Inisialisasi bias dengan nilai 1
for i in range(0, len(n) - 1):
    n[i][-1] = 1

# Lakukan training selama belum mencapai epoch maks.
# atau mse masih lebih dari error maks.
while (mep == -1 or ep < mep) and mse > mer:
    ep += 1
    mse = 0

    # Loop setiap layer
    for r in range(len(X)):
        n[0][:-1] = X[r]

        # Fase 1: feedforward
        for L in range(1, len(C)):

            # Hitung neuron input
            nin[L] = np.dot(n[L - 1], w[L - 1])

            # Hitung nilai aktivasi
            n[L][:len(nin[L])] = sig(nin[L])
```

```
    # Hitung nilai aktivasi
    n[L][:len(nin[L])] = sig(nin[L])

# Selisih antara nilai neuron output dengan target
e = t[r] - n[-1]

# Hitung MSE
mse += sum(e ** 2)

# Fase 2: backpropagation
# Hitung delta di output layer
d[-1] = e * list(sigd(nin[-1]))

# Hitung delta w di output layer
dw[-1] = a * d[-1] * n[-2].reshape((-1, 1))
```

```
        # Hitung delta w di output layer
        dw[-1] = a * d[-1] * n[-2].reshape((-1, 1))

        for L in range(len(C) - 1, 1, -1):
            # Hitung delta input
            din[L - 2] = np.dot(d[L - 1], np.transpose(w[L -
1][:-1]))
            # Hitung delta
            d[L - 2] = din[L - 2] * np.array(list(sigd(nin[L -
1])))
            # Hitung delta w
            dw[L - 2] = (a * d[L - 2]) * n[L - 2].reshape((-1,
1))
        # Update nilai bobot
        w += dw

    # Bagi MSE dengan banyaknya data
    mse /= len(X)

# Return bobot hasil training, jumlah epoch, dan MSE
return w, ep, mse
```

```python
def bp_predict(X, w):

    # Inisialisasi neuron dan neuron input
    n = [np.empty(len(i)) for i in w]
    nin = [np.empty(len(i[0])) for i in w]

    # Hasil
    predict = []

    # Tambahkan bias
    n.append(np.empty(len(w[-1][0])))

    # Loop data input
    for x in X:

        # Masukkan data ke neuron input
        n[0][:-1] = x

        # Untuk setiap neuron output,
        # hitung nilai input dan nilai aktivasi
        for L in range(0, len(w)):
            nin[L] = np.dot(n[L], w[L])
            n[L + 1][:len(nin[L])] = sig(nin[L])

        predict.append(n[-1].copy())

    return predict
```


http://bit.ly/jst-bp

Case Study:
Iris Dataset

```python
import numpy as np

# Instalasi: pip install scikit-learn
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import minmax_scale
from sklearn.metrics import accuracy_score
```

```
# Arsitektur JST
c = 4, 3, 2

# Load dataset
iris = load_iris()

# Normalisasi data
X = minmax_scale(iris.data)

# Konversi label (Y) dengan pola one hot
Y = onehot_enc(iris.target)

# Bagi menjadi data training dan testing
X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size=.3)

# Lakukan training
w, ep, mse = bp_fit(c, X_train, [p[i] for i in y_train],
.1, 1000, .1)
```

```python
print(f'Epoch: {ep}')
print(f'MSE: {mse}')

# Lakukan testing
predict = bp_predict(X_test, w)

# Konversi dari pola one hot ke label
predict = onehot_dec(predict)
y_test = onehot_dec(y_test)

# Hitung akurasi klasifikasi
acc = accuracy_score(out, y_test)

print(f'Output: {out}')
print(f'True: {y_test}')
print(f'Accuracy: {acc}')
```

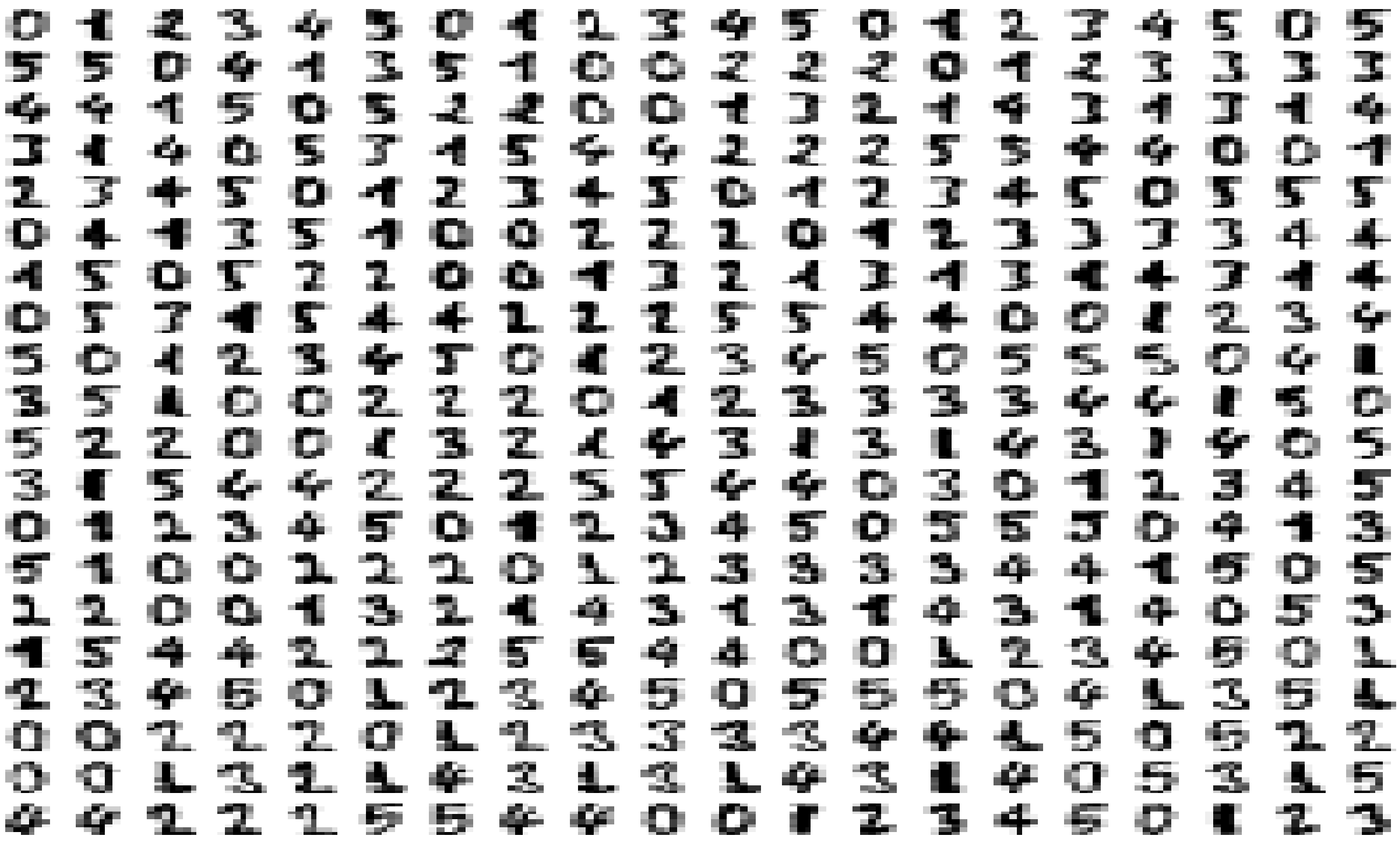
```
# Output:
Epoch: 171
MSE: 0.09914887886321906
Output: [1 2 0 2 0 2 0 0 0 0 1 1 1 2 2 1 2 1 2 0 2 1 0 1 2
 1 1 0 1 0 0 2 2 1 2 1 2 0 1 2 0 1 0 1 1]
True: [1 2 0 2 0 2 0 0 0 0 2 1 1 2 2 1 2 1 2 0 2 1 0 1 2 1
 1 0 1 0 0 2 2 1 2 1 2 0 1 2 0 1 0 1 1]
Accuracy: 0.9777777777777777
```

Case Study:
Digits Dataset

Digits Dataset

- Dataset citra karakter angka 0–9 (10 kelas)
- Jumlah karakter per kelas: ~180
- Jumlah karakter total: 1797
- Ukuran citra: 8x8 piksel (64 dimensi)

A selection from the 64-dimensional digits dataset



```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import minmax_scale
from sklearn.metrics import accuracy_score

digits = datasets.load_digits()
X = minmax_scale(digits.data)
Y = onehot_enc(digits.target)
c = 64, 30, 30, 10

X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size=.3)
w, ep, mse = bp_fit(c, X_train, y_train, .1, -1, .1)

print(f'Epoch: {ep}')
print(f'MSE: {mse}')

predict = bp_predict(X_test, w)
predict = onehot_dec(predict)
y_test = onehot_dec(y_test)
acc = accuracy_score(predict, y_test)

print(f'Output: {predict}')
print(f'True   : {y_test}')
print(f'Accuracy: {acc}')
```


0045352
BLOCK 0045353
BLOCK
Alhamdulillah.