IF3270 Pembelajaran Mesin

Artificial Neural Network (ANN)

Tim Pengajar IF3270

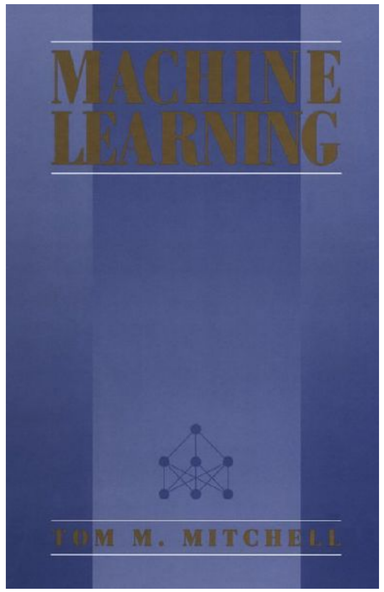
Review

- Machine learning (ML) overview
  - AI vs ML
  - Learning Type:
    - Supervised Learning: Classification, Regression
    - Unsupervised Learning
    - Reinforcement Learning

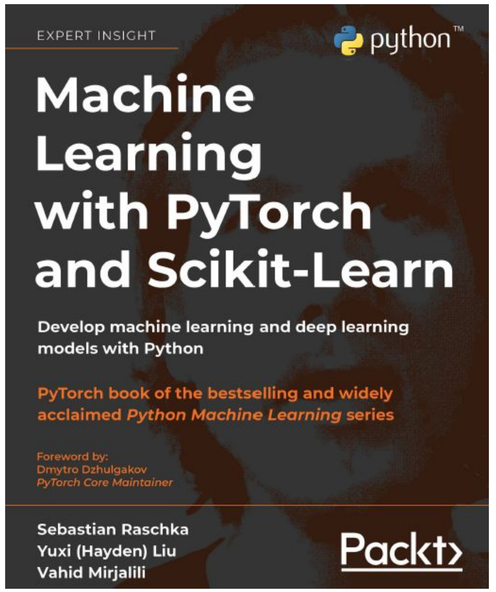
- Ensemble Methods Supervised Learning
  - Homogeneous Parallel Ensemble: Bagging, RF
  - Heterogeneous Parallel Ensemble: Weighting, Meta Learning (Stacking)
  - Sequential Ensemble: Adaboost, Gradient Boosting

- Supervised Learning: Perceptron
- Supervised Learning: ANN: Feed Forward Neural Network Today

Reference

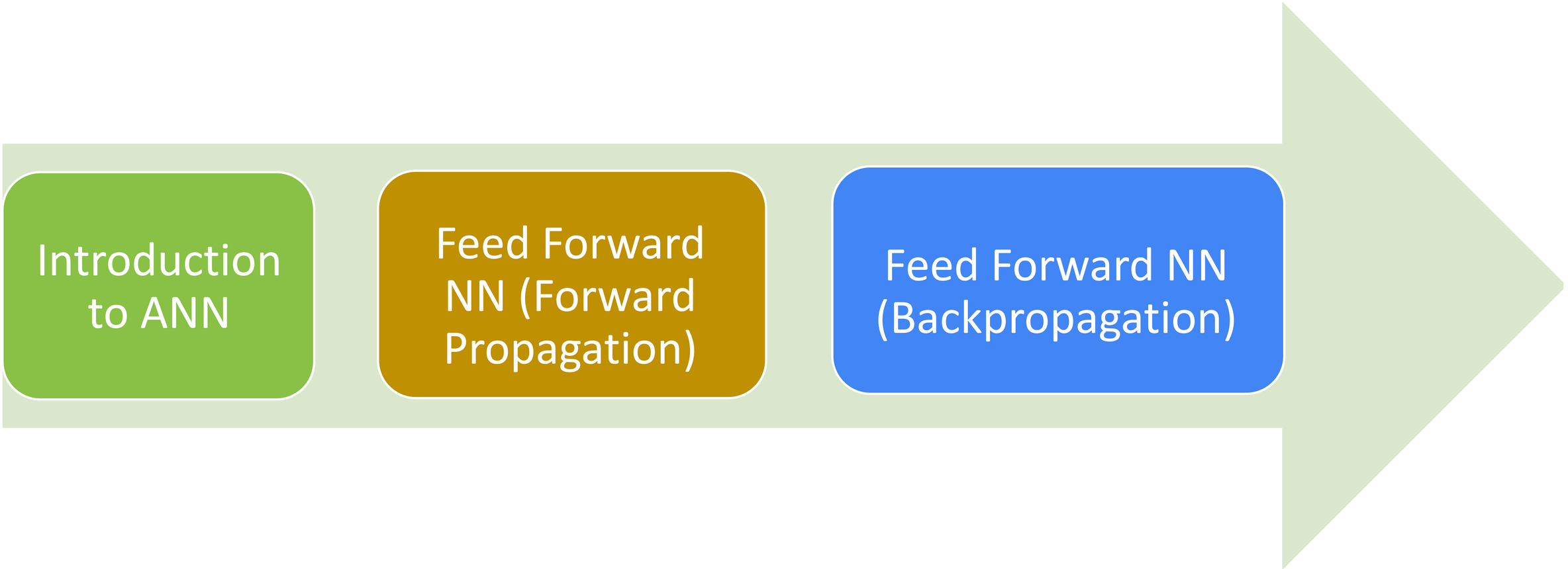


Tom Mitchell,
Machine Learning,
McGraw Hill, 1997



Raschka, et.al., Machine Learning with Pytorch and Scikit-Learn, Packt Publishing Ltd., 2022 (Chapter 11)

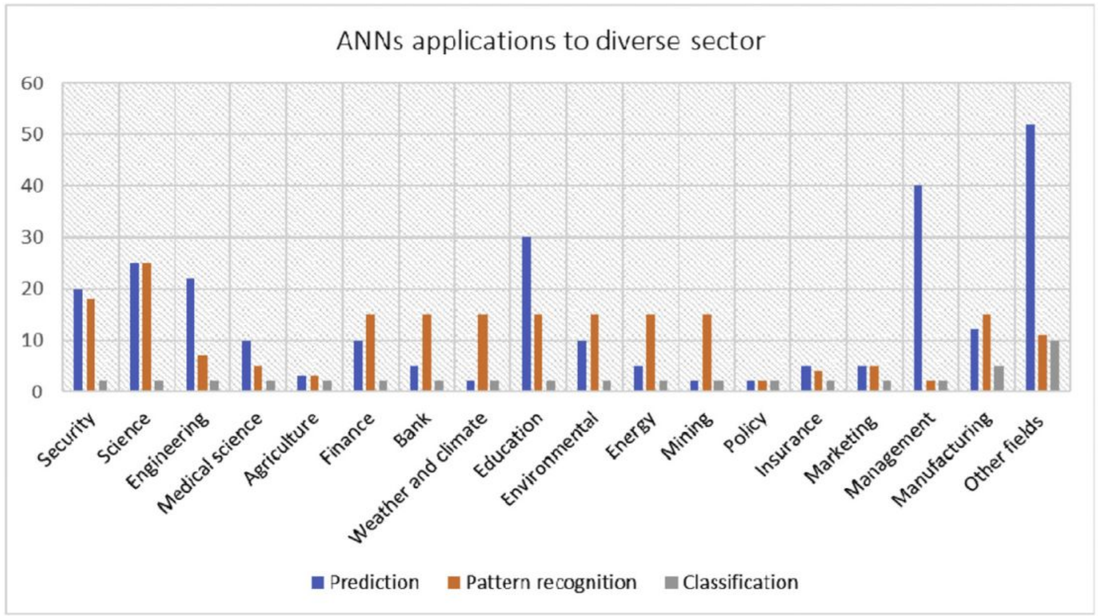
Outline


Introduction to ANN
Feed Forward NN (Forward Propagation)
Feed Forward NN (Backpropagation)

Introduction to ANN

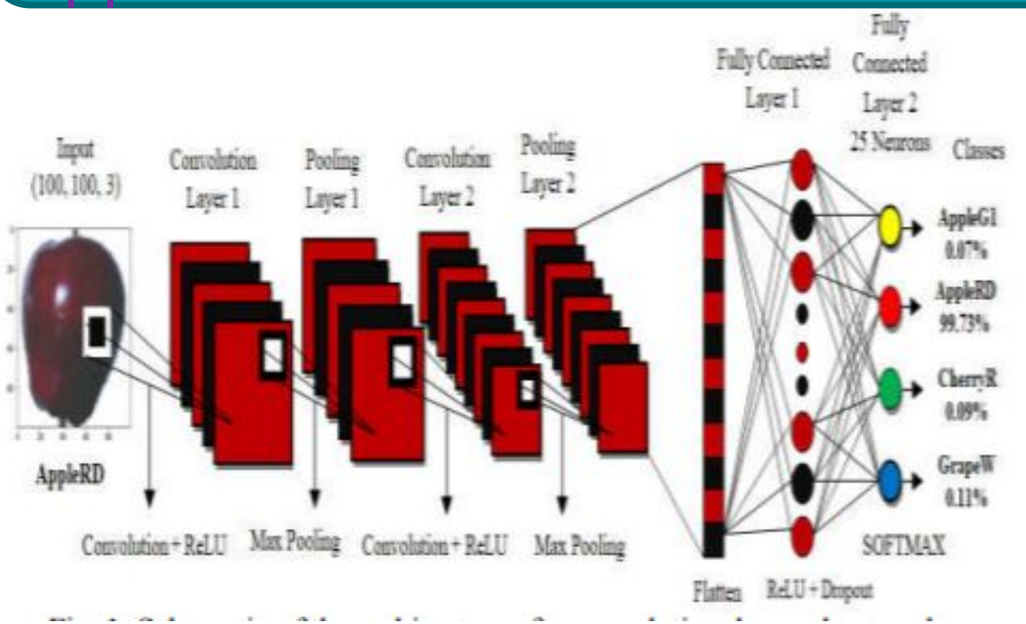
Artificial neural network: Why

ANN apps to diverse sectors, and perform well



Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Mohamed, N. A., & Arshad, H. (2018). State-of-the-art in artificial neural network applications: A survey. *Heliyon*, *4*(11), e00938.

Rapid increase in ANN architectural innovations and application



Sakib, S., Ashrafi, Z., Siddique, M., & Bakr, A. (2019). Implementation of Fruits Recognition Classifier using Convolutional Neural Network Algorithm for Observation of Accuracies for Various Hidden Layers. https://arxiv.org/ftp/arxiv/papers/1904/1904.00783.pdf

Appropriate Problems for ANN Learning

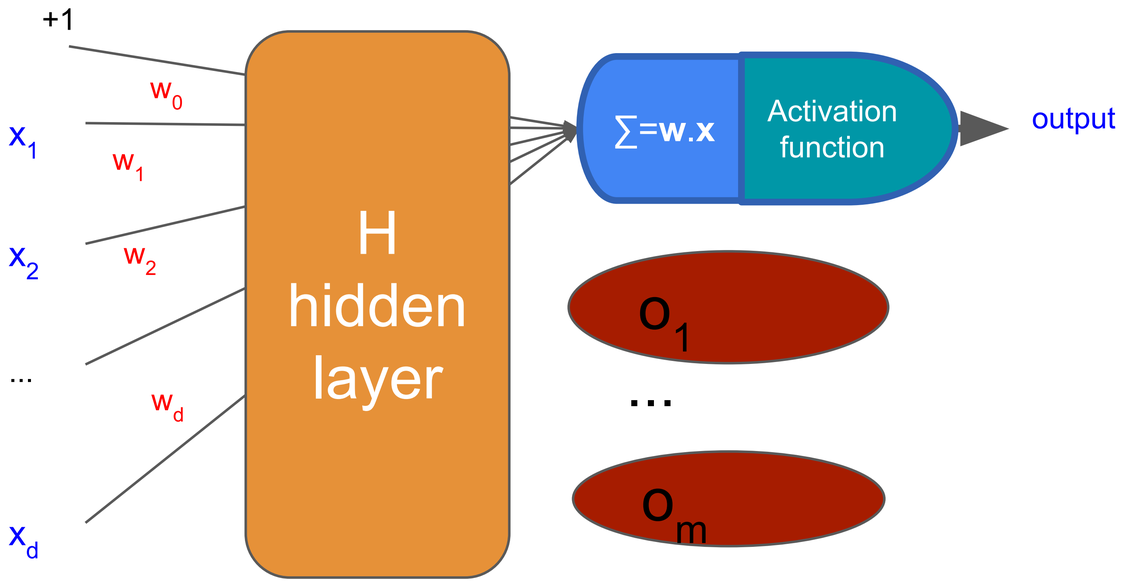
- Instances are represented by many attribute-value pairs.
- Target function output may be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes
- Training examples may contain errors
- Long training times are acceptable
- Fast evaluation of the learned target function may be required
- Ability of humans to understand the learned target function is not important

Artificial neural network: What

Perceptron (incl. Adaline)
single unit/neuron, estimation function (hypothesis)

Artificial Neural
many computational units become intelligent only via their interactions with each other is inspired by the brain (neuroscience)

Network
composing together many different functions

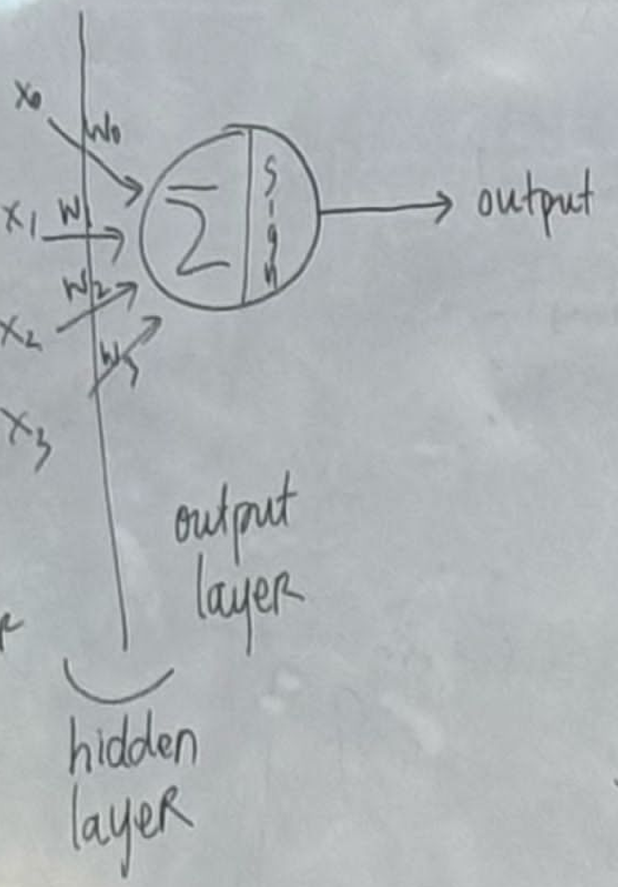


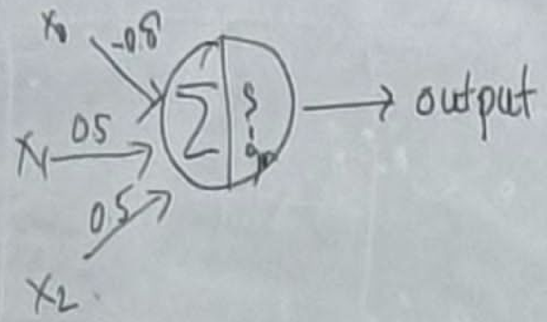
Input layer has d input neurons

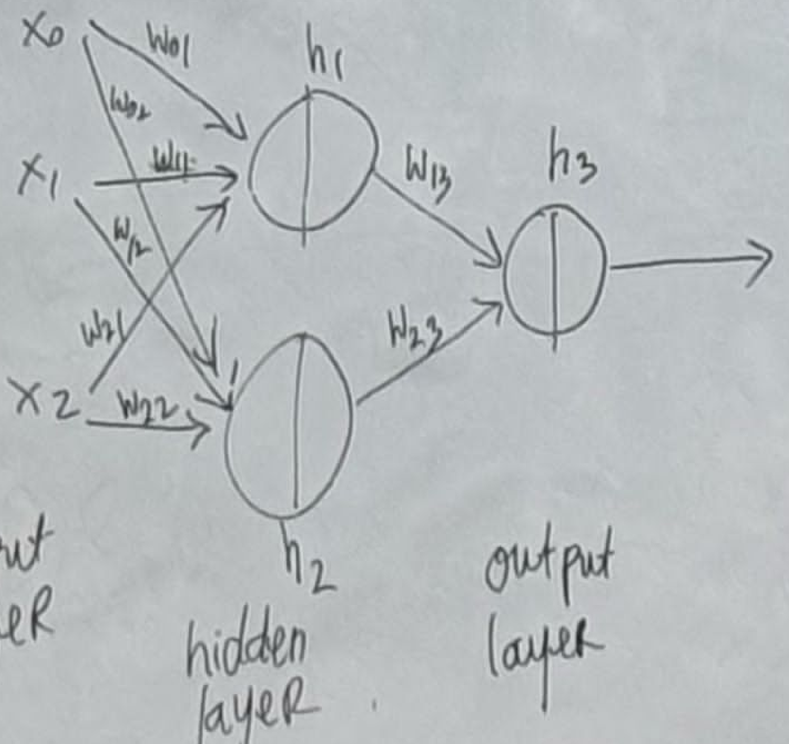
Hidden layer

Output layer has m neurons

XOR (A, B) = $A \wedge \bar{B} \vee \bar{A} \wedge B$





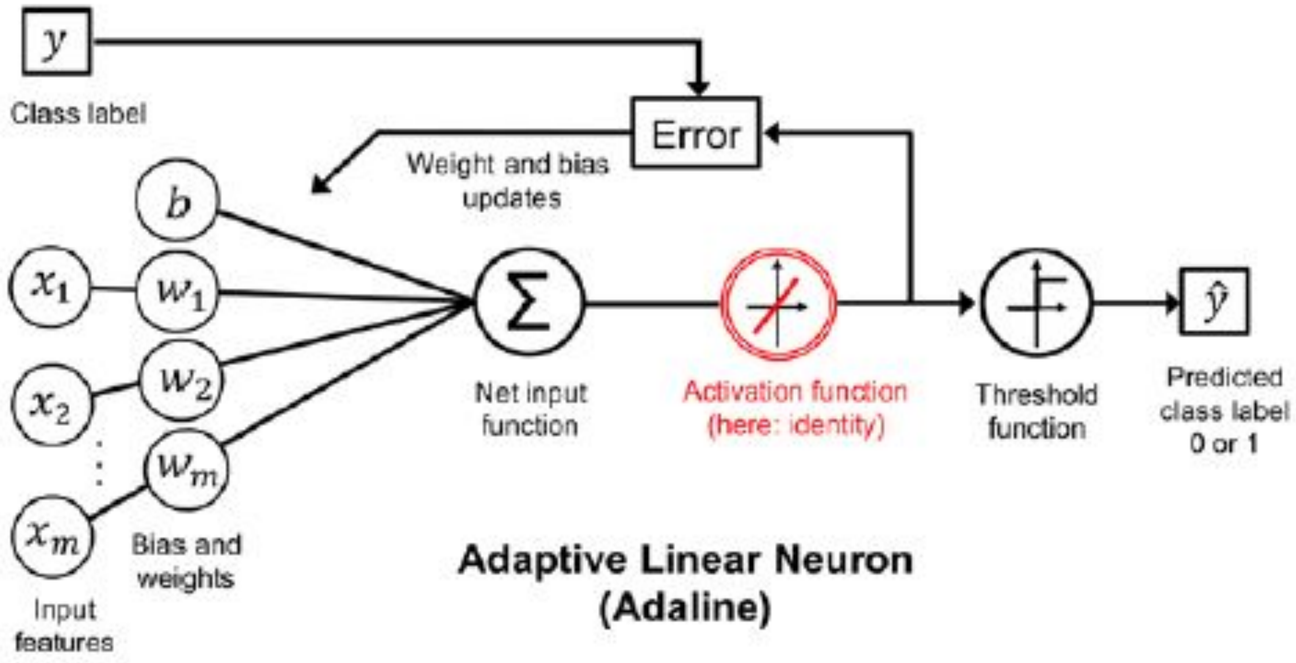


Recap “Artificial Neuron”

Single Layer Neural Network → ADAptive LInear NEuron (Adaline) Algorithm

Update Rule: $w = w + \Delta w; b = b + \Delta b; where\ \Delta$

$$\Delta w_j = -\eta \frac{\partial L}{\partial w_i} \qquad \Delta b = -\eta \frac{\partial L}{\partial b}$$



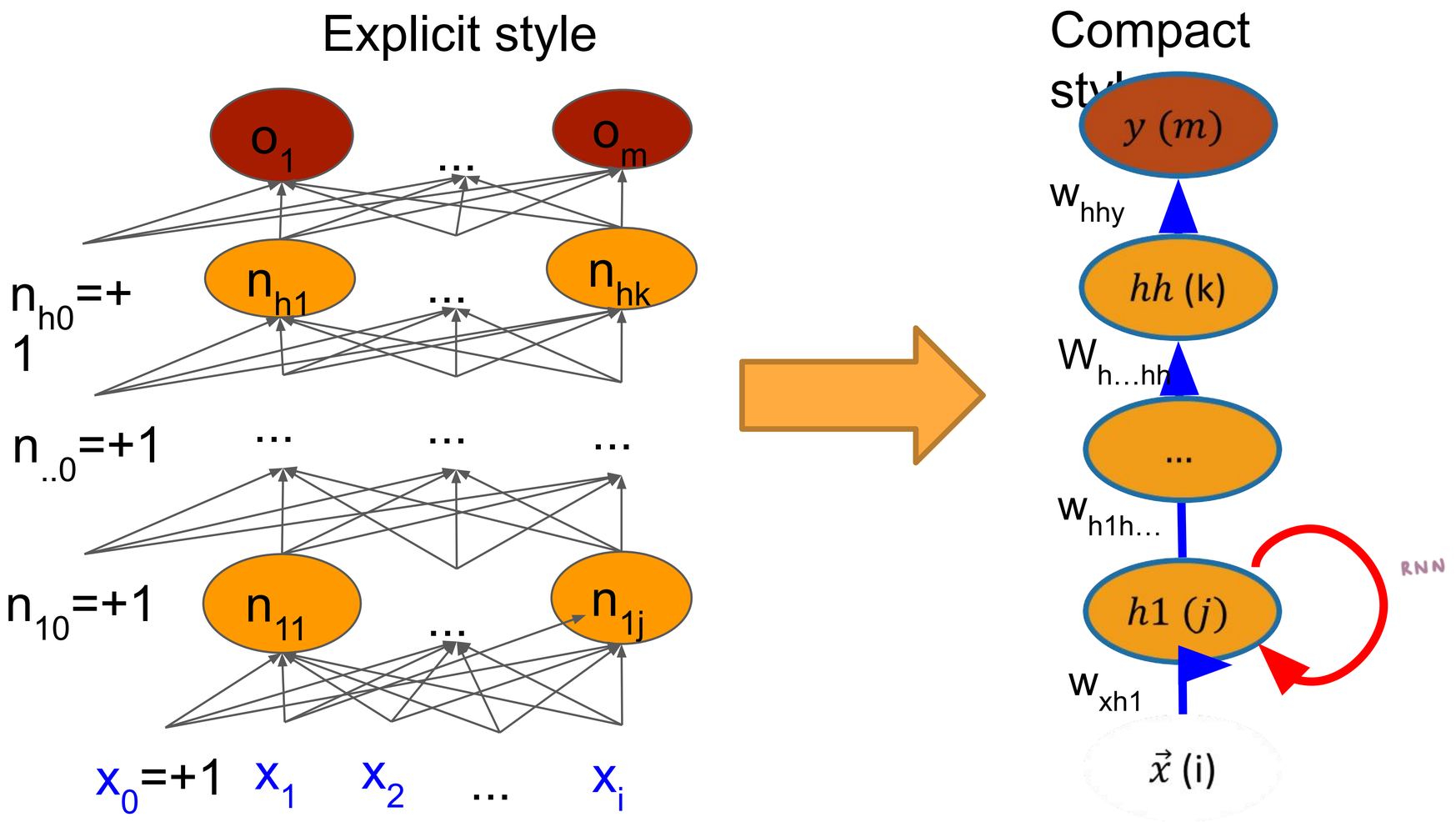
$$L(w) : Squared\ Error\ lost\ function = \frac{1}{2}\,(y^{(i)} - a^{(i)})^2$$

$$a = \sigma\left(\sum_j w_j . x_j + b\right)$$

$$\hat{y} = \begin{cases} 1, if\ a \geq 0 \\ 0, otherwise \end{cases}$$

Raschka, *Figure 11.1: The Adaline algorithm*

Artificial Neural Network Architecture



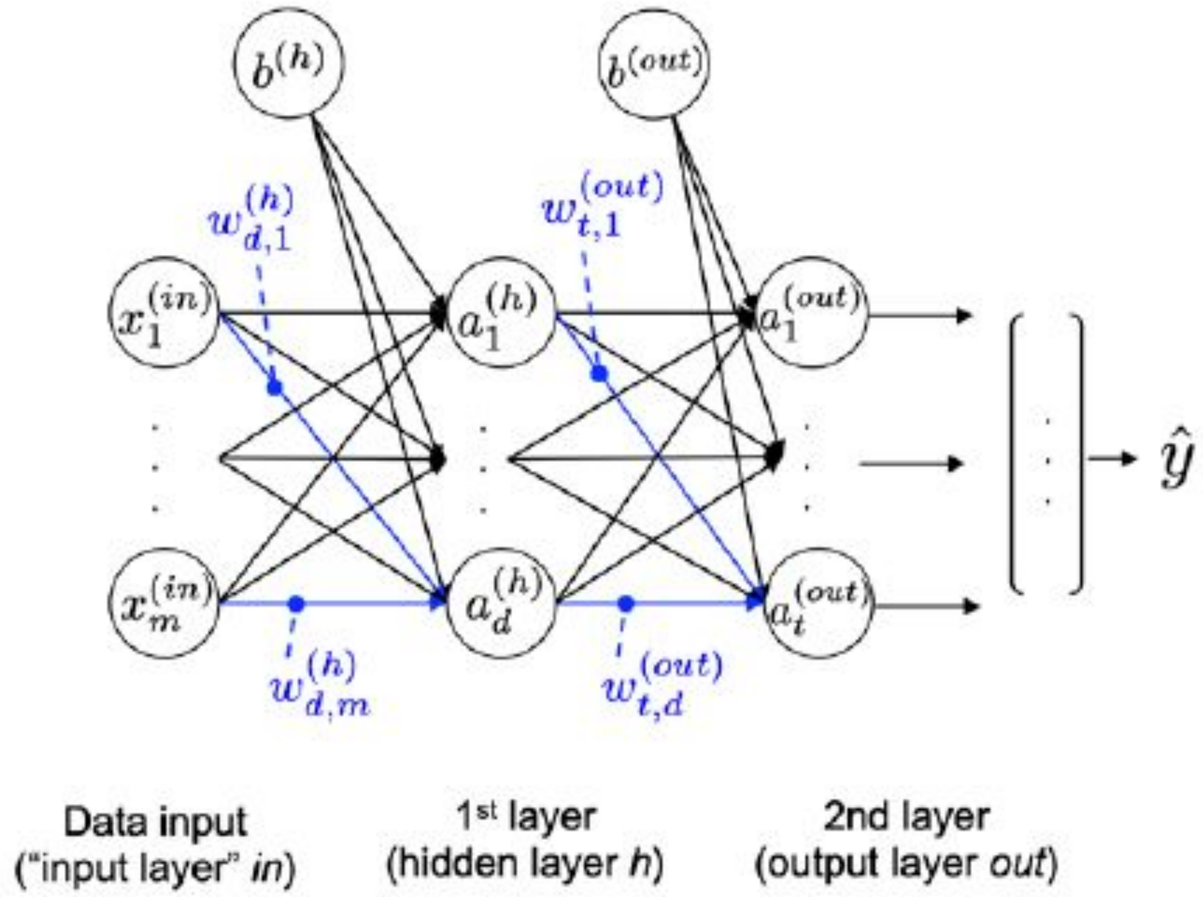
ANN Architecture (Russell & Norvig, AIMA 4th ed, 2022)

- Feed Forward Neural Network (FFNN)
  - connections only in one direction → it forms a directed acyclic graph with designated input and output nodes
    - Multi layer Perceptron: Fully Connected Feed Forward, non linear activation function
  - Information flows through the network from the input nodes to the output nodes, and there are no loops

- Recurrent Neural Network → LATER
  - feeds its intermediate or final outputs back into its own inputs (loops)
  - the signal values within the network form a dynamical system that has internal state or memory

Feed Forward Neural Network (FFNN)

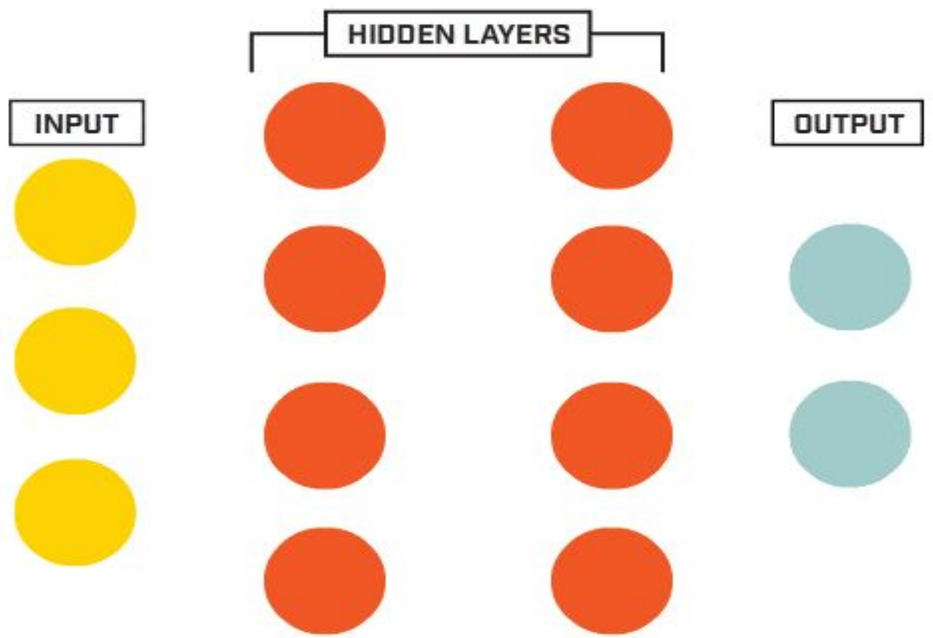
Example of Multi Layer Perceptron with 2 Layer

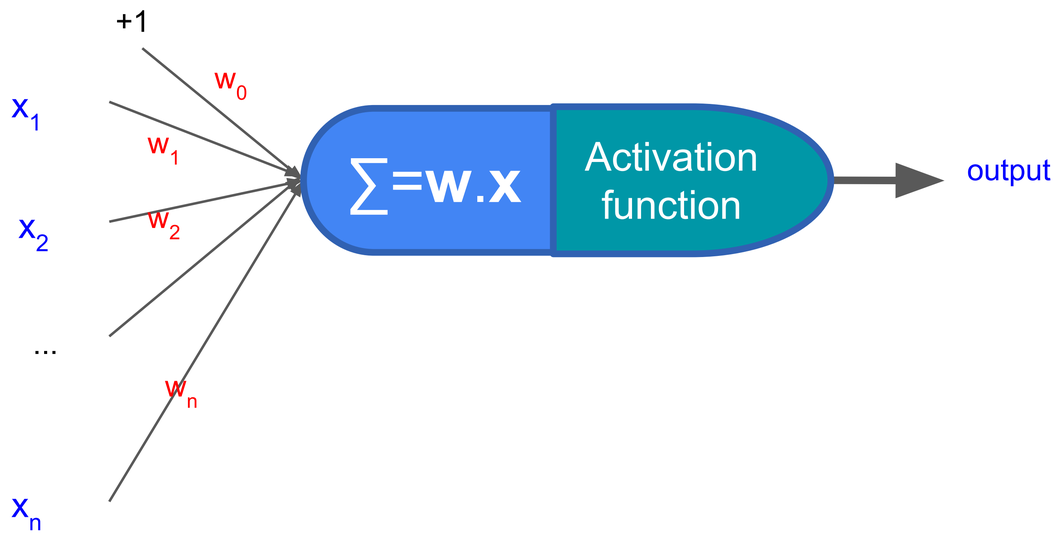


Raschka, *Figure 11.2: A two-layer MLP*

Note that in some contexts, the inputs are also regarded as a layer. However, in this case,
it would make the Adaline model, which is a single-layer neural network, a two-layer neural network, which may be counterintuitive

Artificial Neural Network: Inference (Forward Propagation)





Forward propagation: Information flows through the function being evaluated from vector x, through intermediate computations used to define f, and finally to the output y calculate the output of FFNN

Input $\mathbf{x}=(1, x_1, x_2, \ldots, x_n)$

Model $\mathbf{w}=(w_0, w_1, w_2, \ldots, w_n)$

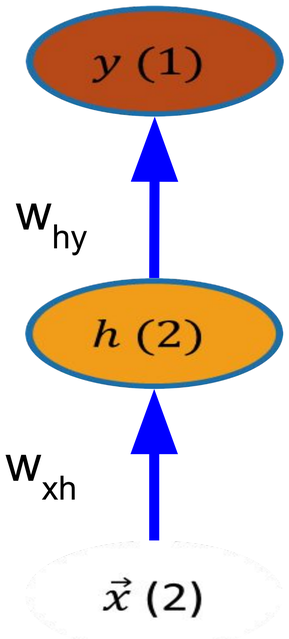
$\sum=\mathbf{w}.\mathbf{x}=w_0+w_1x_1+..+w_nx_n$

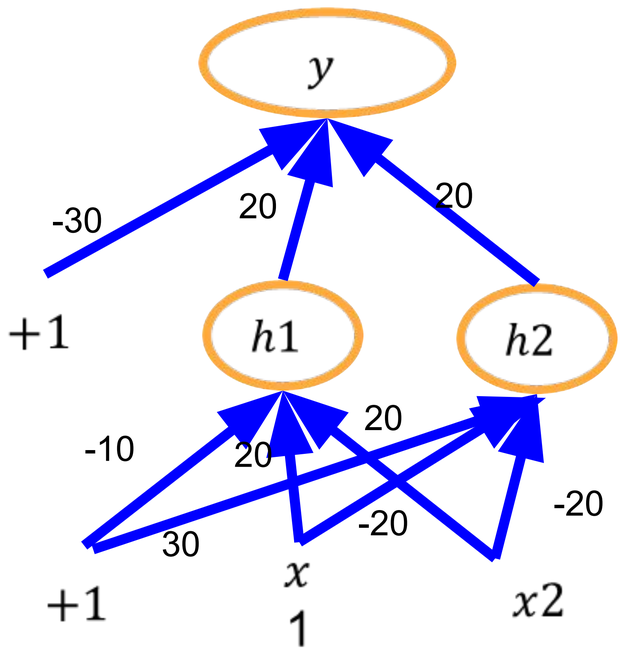
Output = f (∑)

https://kookycoder.com/blog/deep-learning-and-its-functioning/

Example: XOR (Sigmoid Model)

σ (net): $1/(1+e^{-net})$





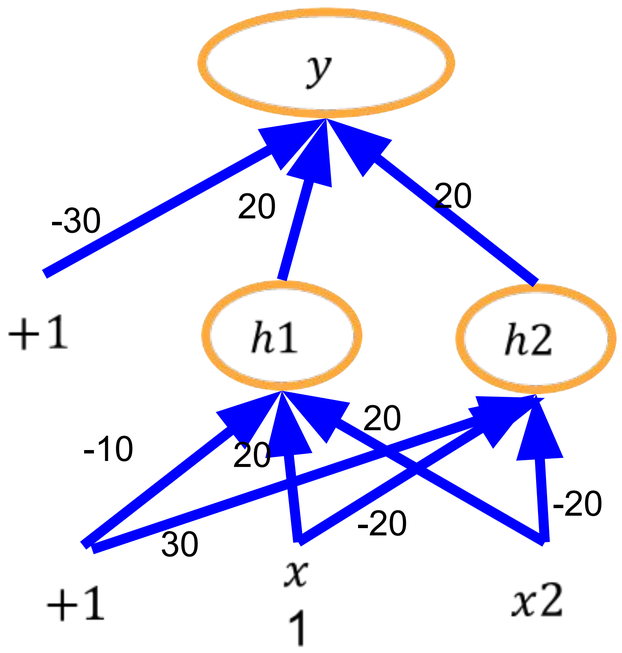
sigmoid

h1= σ(-10+20*x1+20*x2)
h2= σ(30-20*x1-20*x2)
y = σ(-30+20*h1+20*h2)

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

Target function classifies two input into predefined classes {0, 1}.

FFNN: XOR (Sigmoid Model)

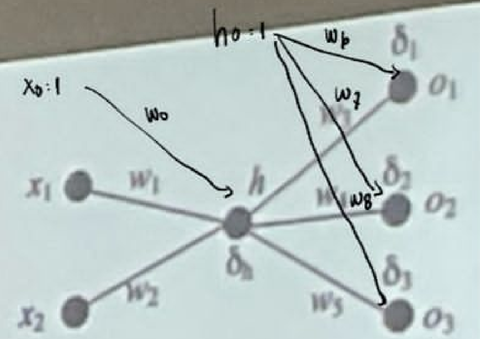


h1= σ(-10+20*x1+20*x2)
h2= σ(30-20*x1-20*x2)
y = σ(-30+20*h1+20*h2)

x0	x1	x2	y	∑h1	h1	∑h2	h2	∑ŷ	ŷ
1	0	0	0	-10	0.00	30	1.00	-10.00	0.00
1	0	1	1	10	1.00	10	1.00	10.00	1.00
1	1	0	1	10	1.00	10	1.00	10.00	1.00
1	1	1	0	30	1.00	-10	0.00	-10.00	0.00

σ (net): $1/(1+e^{-net})$

Latihan 1



Diketahui sebuah arsitektur jaringan Neural Network sebagai berikut yang memiliki dua unit input x, tiga unit output o dan satu unit hidden h, serta menggunakan fungsi aktivasi Sigmoid di setiap unitnya. Proses pembelajaran menerapkan algoritma Backpropagation.

Tuliskan formula untuk menghitung luaran h dan o_1

$h = \sigma(w_0 \cdot 1 + w_1 \cdot x_1 + w_2 \cdot x_2)$

(yg masuk ke o_3 cuman w_8 dan w_5)

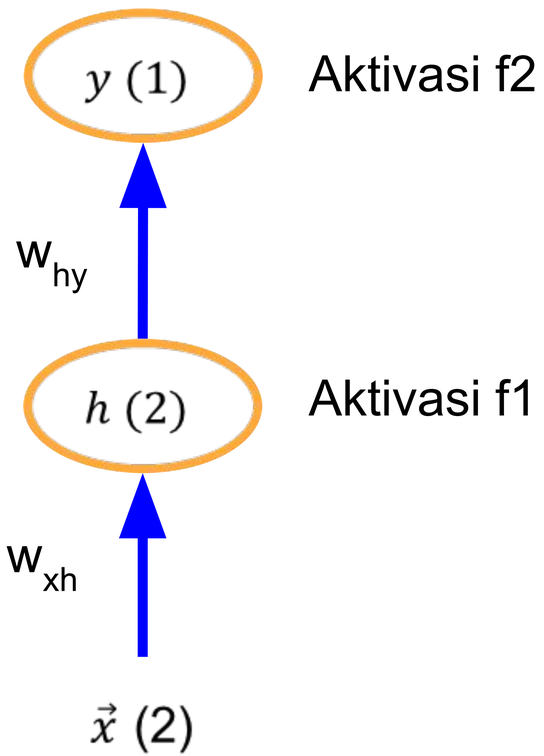
$o_3 = \sigma(w_8 \cdot 1 + w_5 \cdot h)$

$o_1 = \sigma(w_4 + w_3 \cdot h)$

$o_2 = \sigma(w_7 + w_6 \cdot h)$

FFNN: Forward Propagation dengan Input Vector **x**

Compact style



Input vector x:

$$f(x; W_{xh}, c, W_{hy}, b) = f2(W_{hy}^{T}.f1(W_{xh}^{T}.x+c)+b)$$

Jika bobot bias c menjadi bagian Wxh dan bobot bias b menjadi bagian dari Why, maka:

$$f(x; W_{xh}, W_{hy}) = f2(W_{hy}^{T}.f1(W_{xh}^{T}.x))$$

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

tips soal :

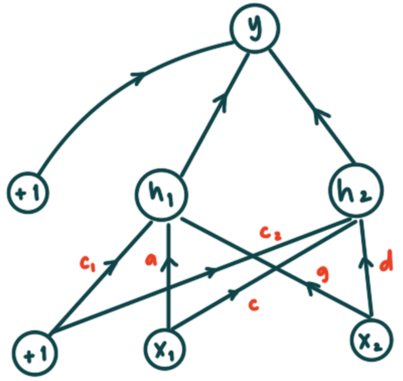
kalo cm dikasi 1 instance → vektor

XOR = ⊽

XOR = RELU & Linear

$f_2 \longrightarrow W_{hy} = \begin{bmatrix} e \\ f \end{bmatrix}$; $b = [h]$

$f_1 \longrightarrow W_{xh} = \begin{bmatrix} a & c \\ g & d \end{bmatrix}$; $c = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$



kalo datanya cuma 1, jgn lupa di transpose

formula 2

$W_{hy} = \begin{bmatrix} b_1 \\ e \\ f \end{bmatrix}$ (W_{hy_0})

$W_{xh} = \begin{bmatrix} c_1 & a & c \\ c_2 & g & d \end{bmatrix}$ (0)

$f(\vec{x}) = f_2(W_{hy} \cdot f_1(W_{xh} \cdot x))$

formula 1

$h_1 = f_1(x_1 \cdot a + x_2 \cdot y + c_1)$

$h_2 = f_1(c_2 + x_1 c + x_2 d)$

$y = f_2([b_1] + h_1 e + h_2 f) \longrightarrow f_2 = \left([b_1] + \begin{bmatrix} e \\ f \end{bmatrix}^T \cdot \begin{bmatrix} h_1 \\ h_2 \end{bmatrix}\right)$

(e & f adlh W_{hy}, dikalikan dgn $h_1 . h_2$, ditambah dgn b)

$f(\vec{x}) = f_2\left(W_{hy}^T \cdot f_1\left(W_{xh}^T \cdot x + c\right) + b\right)$

$\begin{bmatrix} a & g \\ c & d \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$

$\begin{bmatrix} h_1 \\ h_2 \end{bmatrix}$

$[e\ f]$

$= [e h_1 + f h_2] + [b_1]$

$= [e h_1 + e h_2 + b_1]$

FFNN: XOR (RELU & LINEAR) □ input vector x

rectified linear unit

Input vector **x**:

$$f(x; W_{xh}, c, W_{hy}, b) = W_{hy}^{T} . (\max\{0, W_{xh}^{T} . x + c\}) + b$$

Compact style

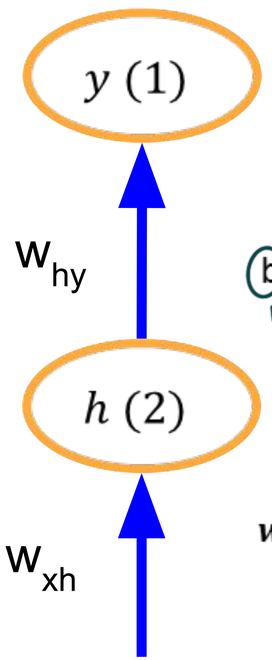
input 1 vektor | 1 instance

$$f(\vec{x}) = \sigma(W_{hy}^{T} . \sigma(W_{xh}^{T} . \vec{x} + c) + b)$$

$$f(\vec{x}) = \text{linear}(W_{hy} . \text{ReLU}(W_{xh} . \vec{x} + c) . b)$$

input 1 batch

$$f(X) = \sigma(\sigma(X W_{xh} + C) . W_{hy} + b)$$



$$W_{hy} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

b=0 (bias)

$$y = W_{hy}^{T} . h + b = [1]$$

Transpose matriks bobot

Bobot ke h1

Bobot ke h2

$$W_{xh} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

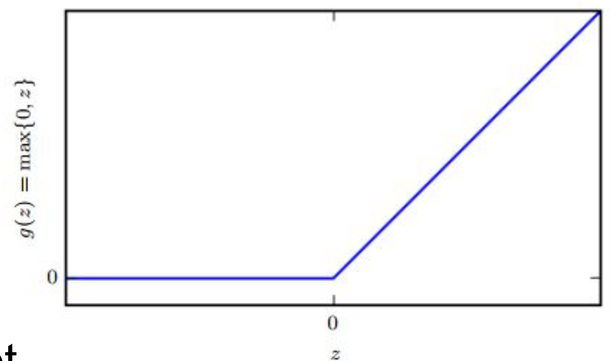
$$c = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$ (bias)

$$h = \text{ReLU}(W_{xh}^{T} . x + c) = \max\{0, W_{xh}^{T} . x + c\} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$X = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Contoh: instance ke-2

ReLU(z)=max{0,z}

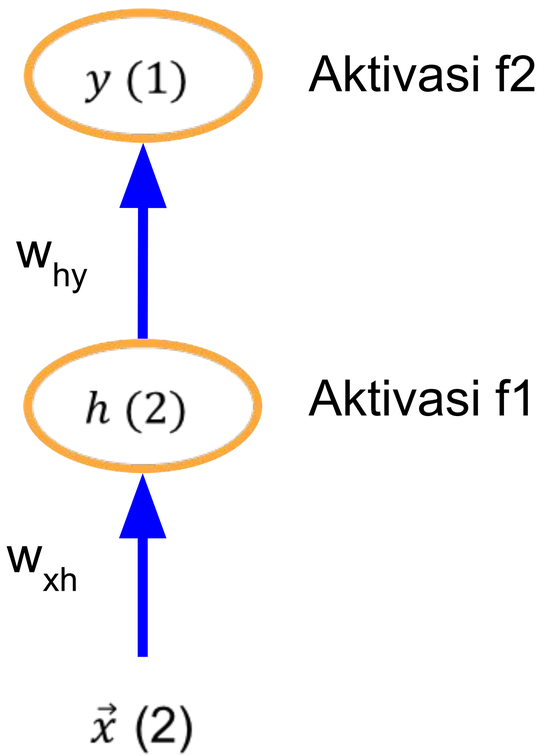


h1= ReLU(0+x1+x2)
h2= ReLU(-1+x1+x2)
y = (0+h1-2*h2)

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

FFNN: Forward Propagation dengan Input Mini-batch **X**

Compact style



Input vector x:

$$f(x; W_{xh}, c, W_{hy}, b) = f2(W_{hy}^{T}.f1(W_{xh}^{T}.x+c)+b)$$

Jika bobot bias c menjadi bagian Wxh dan bobot bias b menjadi bagian dari Why, maka:

$$f(x; W_{xh}, W_{hy}) = f2(W_{hy}^{T}.f1(W_{xh}^{T}.x))$$

Input mini-batch **X**:

$$f(\mathbf{X}; W_{xh}, c, W_{hy}, b) = f2(f1(\mathbf{X}W_{xh}+c).W_{hy}+b)$$

Jika bobot bias c menjadi bagian Wxh dan bobot bias b menjadi bagian dari Why, maka:

$$f(\mathbf{X}; W_{xh}, W_{hy}) = f2(f1(\mathbf{X}W_{xh}).W_{hy})$$

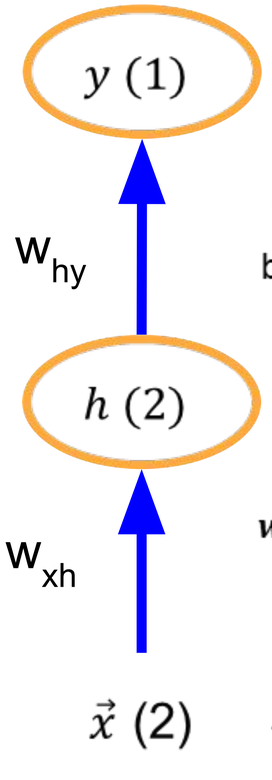
Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

FFNN: XOR (RELU & LINEAR) input mini-batch X

Input mini-batch **X**:

$$f(\mathbf{X}; w_{xh}, c, w_{hy}, b) = (\max\{0, \mathbf{X}w_{xh}+c\})w_{hy}+b$$

Compact style



$w_{hy} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$ b=0 ⇨ $\mathbf{y}=\mathbf{h}w_{hy}+\mathbf{b}=\begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$

Bobot ke h1

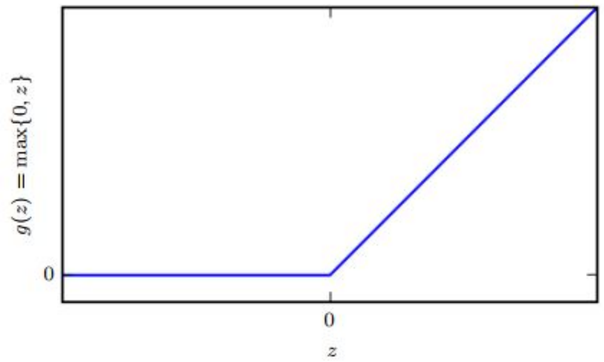
$w_{xh} = \begin{bmatrix} \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} \end{bmatrix}$ $c = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$ ⇨ $\mathbf{h}=\mathrm{ReLU}(\mathbf{X}w_{xh}+c)=\max\{0,XW+c\}=\begin{bmatrix} \mathbf{0} & \mathbf{0} \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$

Bobot ke h2

$X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$

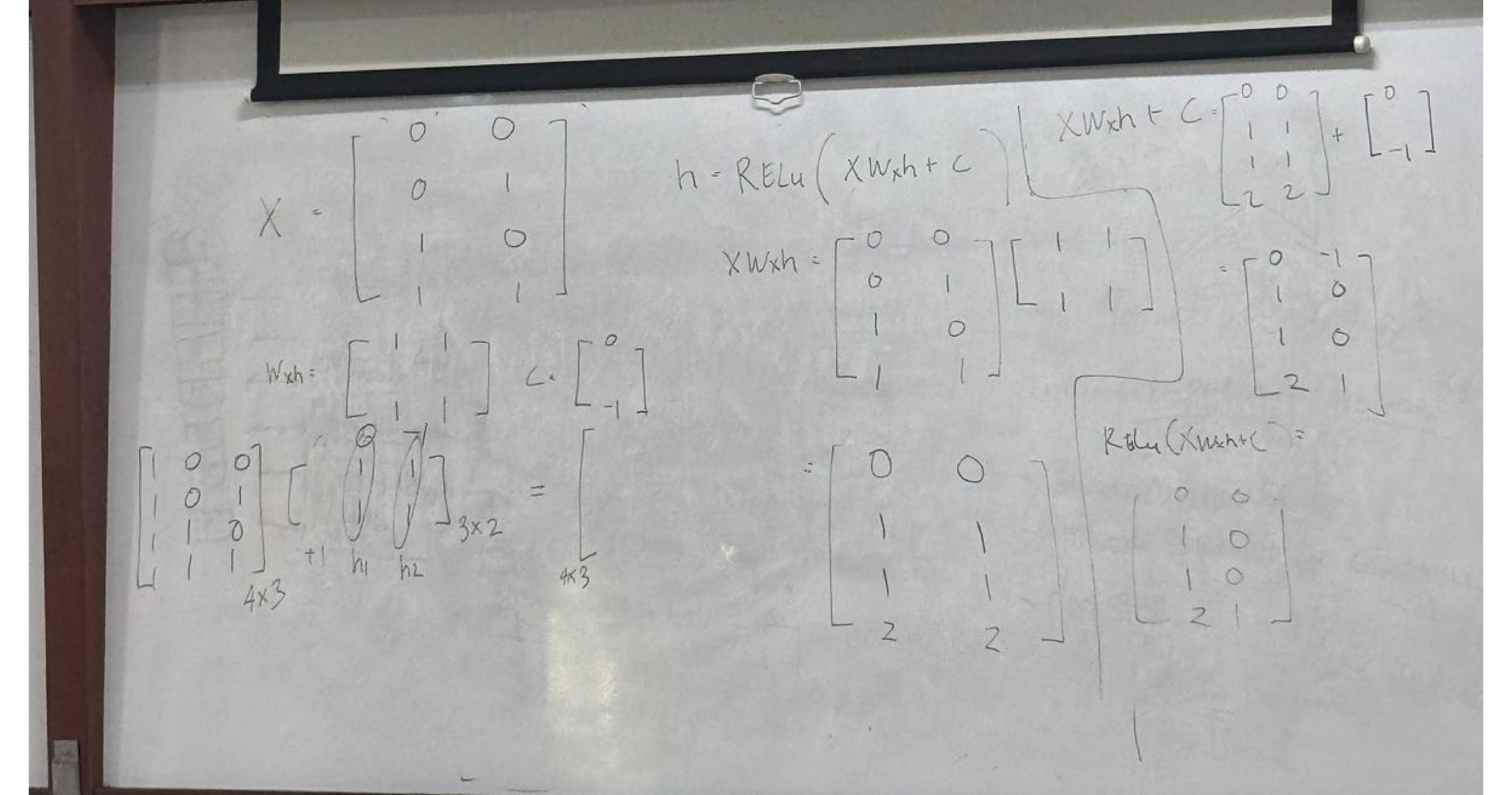
Contoh: mini-batch **X**

ReLU(z)=max{0,z}



h1= ReLU(0+x1+x2)
h2= ReLU(-1+x1+x2)
y = (0+h1-2*h2)

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

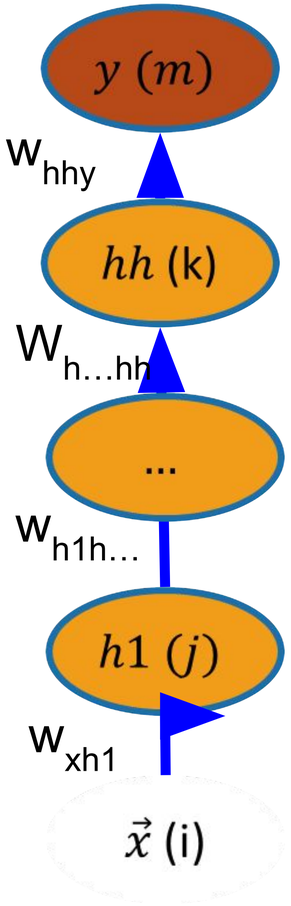

X = [0 0; 0 1; 1 0; 1 1]
W_xh = [1 1; 1 1] c = [0; -1]
[0 0; 0 1; 1 0; 1 1] [1 1; 1 1] 3x2
+1 h1 h2
4x3
=
4x3
h = ReLu(XW_xh + c)
XW_xh = [0 0; 0 1; 1 0; 1 1] [1 1; 1 1]
= [0 0; 1 1; 1 1; 2 2]
XW_xh + c = [0 0; 1 1; 1 1; 2 2] + [0; -1]
= [0 -1; 1 0; 1 0; 2 1]
ReLu(XW_xh+c) =
[0 0; 1 0; 1 0; 2 1]

Forward Propagation

Require: Network depth, l
Require: $\boldsymbol{W}^{(i)}, i \in \{1, \ldots, l\}$, the weight matrices of the model
Require: $\boldsymbol{b}^{(i)}, i \in \{1, \ldots, l\}$, the bias parameters of the model
Require: $\boldsymbol{x}$, the input to process
Require: $\boldsymbol{y}$, the target output

$\boldsymbol{h}^{(0)} = \boldsymbol{x}$
for $k = 1, \ldots, l$ **do**
  $\boldsymbol{a}^{(k)} = \boldsymbol{b}^{(k)} + \boldsymbol{W}^{(k)} \boldsymbol{h}^{(k-1)}$
  $\boldsymbol{h}^{(k)} = f(\boldsymbol{a}^{(k)})$
end for
$\hat{\boldsymbol{y}} = \boldsymbol{h}^{(l)}$
$J = L(\hat{\boldsymbol{y}}, \boldsymbol{y}) + \lambda \Omega(\theta)$

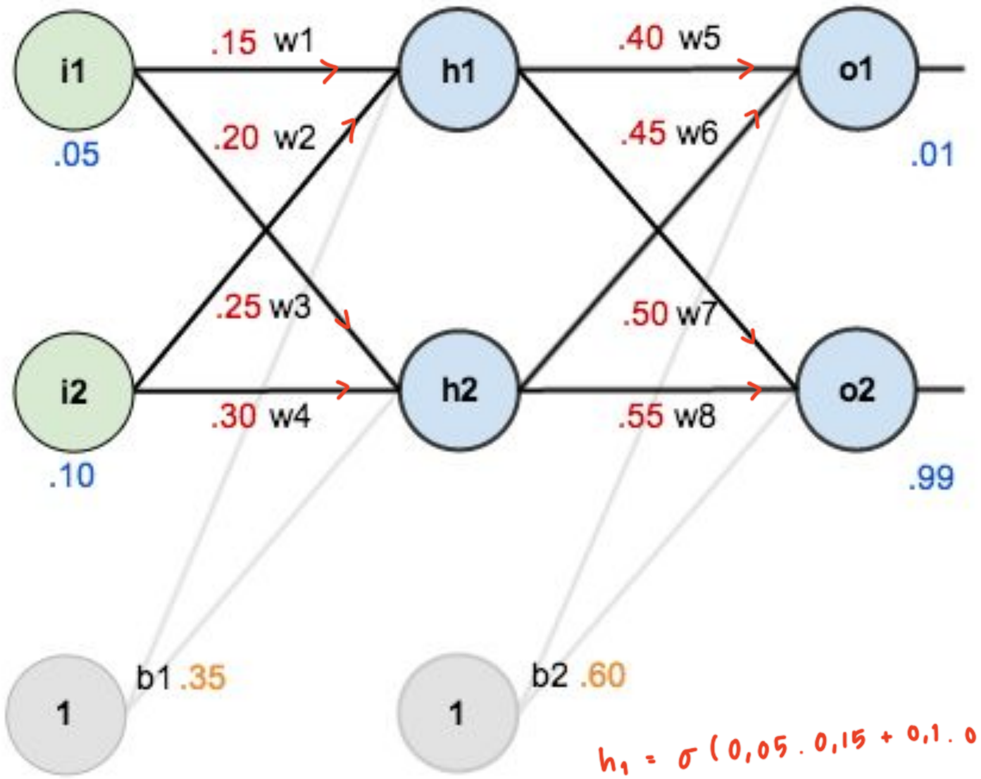
f(a): activation function
$L(\hat{y}, y)$: loss function
θ: all parameters (W,b)
Ω(θ): regularizer



Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

Forward Propagation: Example

(from http://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/)



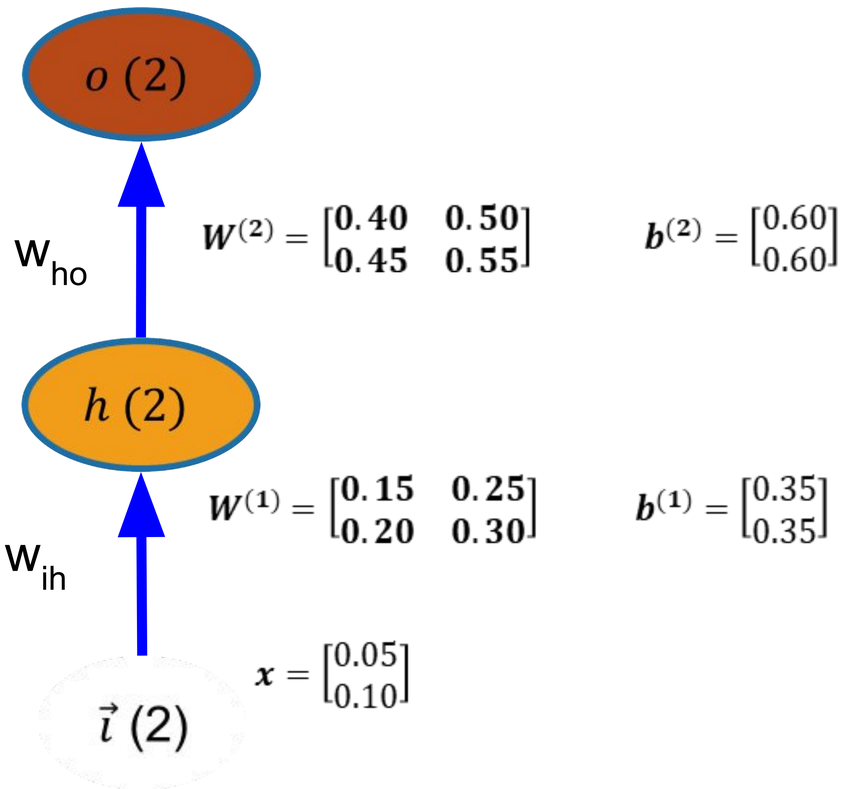
$h_1 = \sigma(0{,}05 \cdot 0{,}15 + 0{,}1 \cdot 0{,}2 + 0{,}35)$

$h_2 = \sigma(0{,}05 \cdot 0{,}25 + 0{,}1 \cdot 0{,}3 + 0{,}35)$

$o_1 = \sigma(h_1 \cdot 0{,}4 + h_2 \cdot 0{,}45 + 0{,}6)$

$o_2 = \sigma(h_1 \cdot 0{,}5 + h_2 \cdot 0{,}55 + 0{,}6)$

Compact style: sigmoid model



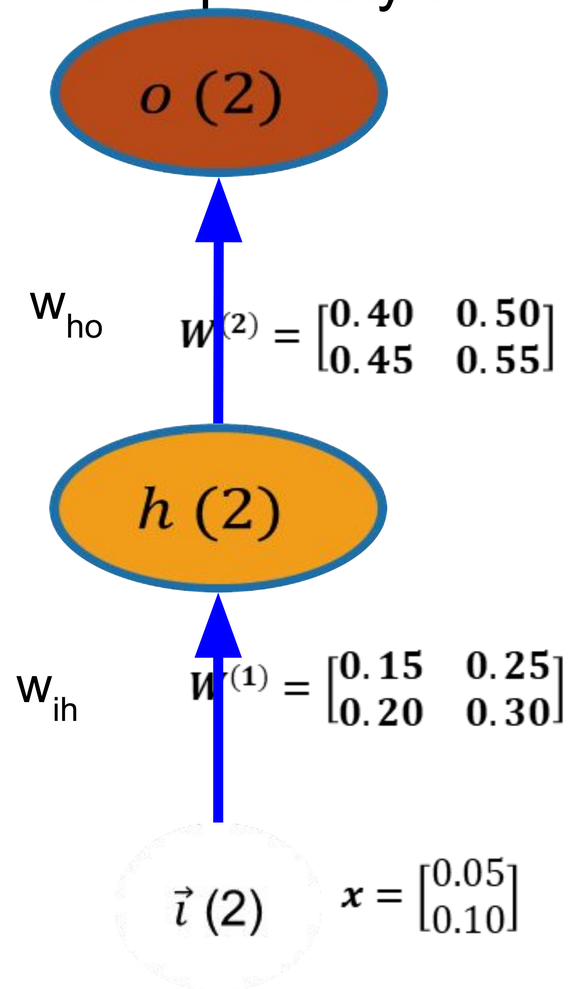
$$W^{(2)} = \begin{bmatrix} 0.40 & 0.50 \\ 0.45 & 0.55 \end{bmatrix} \qquad b^{(2)} = \begin{bmatrix} 0.60 \\ 0.60 \end{bmatrix}$$

$$W^{(1)} = \begin{bmatrix} 0.15 & 0.25 \\ 0.20 & 0.30 \end{bmatrix} \qquad b^{(1)} = \begin{bmatrix} 0.35 \\ 0.35 \end{bmatrix}$$

$$x = \begin{bmatrix} 0.05 \\ 0.10 \end{bmatrix}$$

Forward Propagation

Compact style



W_{ho} $\quad W^{(2)} = \begin{bmatrix} 0.40 & 0.50 \\ 0.45 & 0.55 \end{bmatrix}$ $\quad b^{(2)} = \begin{bmatrix} 0.60 \\ 0.60 \end{bmatrix}$

W_{ih} $\quad W^{(1)} = \begin{bmatrix} 0.15 & 0.25 \\ 0.20 & 0.30 \end{bmatrix}$ $\quad b^{(1)} = \begin{bmatrix} 0.35 \\ 0.35 \end{bmatrix}$

$x = \begin{bmatrix} 0.05 \\ 0.10 \end{bmatrix}$

Target:

$o_1 = 0.10$

$o_2 = 0.99$

k	a	h
0	-	$\begin{bmatrix} 0.05 \\ 0.10 \end{bmatrix}$
1	$\begin{bmatrix} 0.15 * 0.05 + 0.20 * 0.10 + 0.35 \\ 0.25 * 0.05 + 0.30 * 0.10 + 0.35 \end{bmatrix} = \begin{bmatrix} 0.3775 \\ 0.3925 \end{bmatrix}$	$\begin{bmatrix} 0.5933 \\ 0.5969 \end{bmatrix}$
2	$\begin{bmatrix} 0.40 * 0.5933 + 0.45 * 0.5969 + 0.60 \\ 0.50 * 0.5933 + 0.55 * 0.5969 + 0.60 \end{bmatrix} = \begin{bmatrix} 1.1059 \\ 1.2249 \end{bmatrix}$	$\begin{bmatrix} 0.7514 \\ 0.7729 \end{bmatrix}$

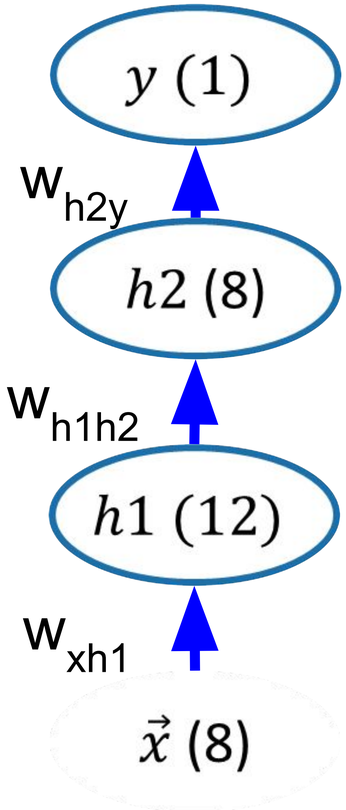
$\hat{y} = \begin{bmatrix} 0.7514 \\ 0.7729 \end{bmatrix}$

$J = \frac{1}{2}\left[(0.01 - 0.7514)^2 + (0.99 - 0.7729)^2\right]$

$\mathbf{J = 0.2984}$

Implementing FFNN on Keras

Compact style



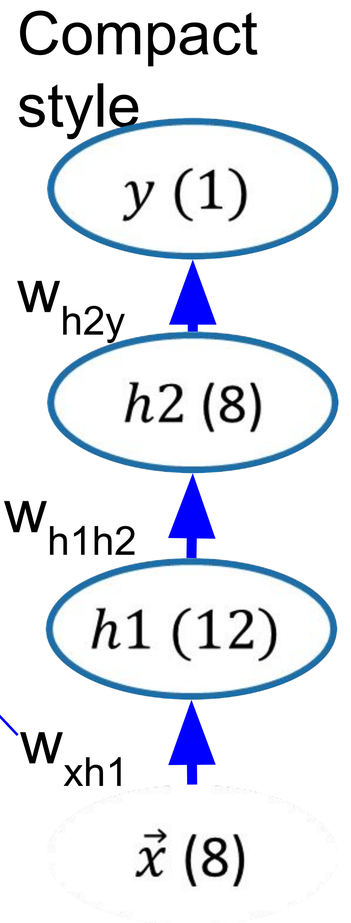

```
from keras.models import Sequential
from keras.layers import Dense


# define the keras model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# compile the keras model
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
# fit the keras model on the dataset
model.fit(X, y, epochs=150, batch_size=10)
```

Number of Parameter

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 12)                108
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 104
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 9
=================================================================
Total params: 221
Trainable params: 221
```



Total parameter = (8+1)*12+(12+1)*8+(8+1)*1=221

Backpropagation (Learning)

Learning Procedure

Goal: set weights to minimize error

Forward propagation: compute output y for input instance **x**

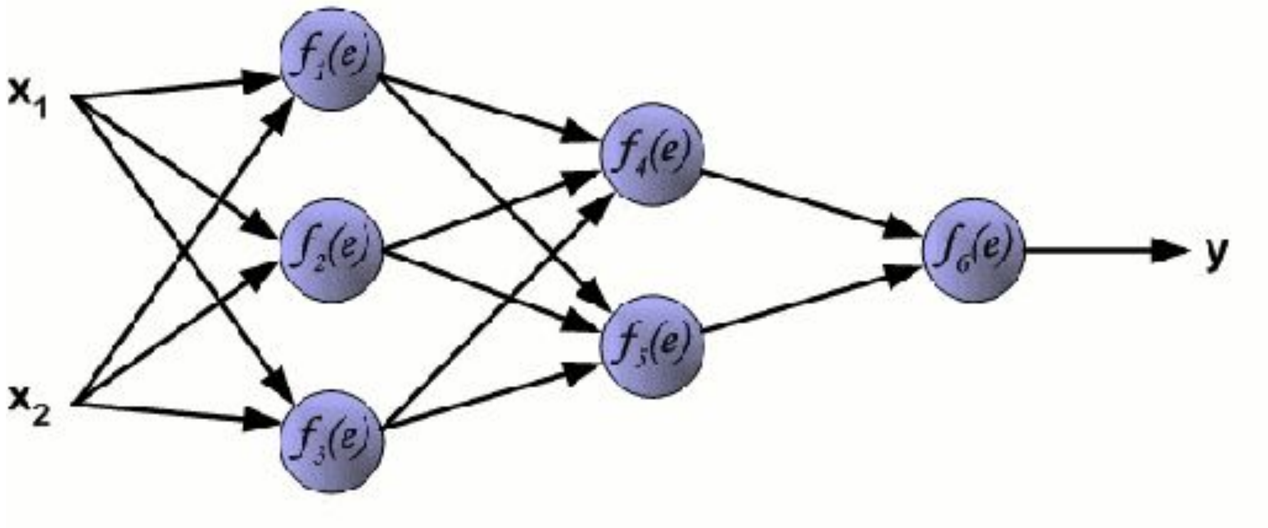
hitung output dr setiap neuron

Backward propagation: compute error term δ for each neuron (target hidden neuron is Not Available)

hitung error term utk setiap neuron

Update every weights

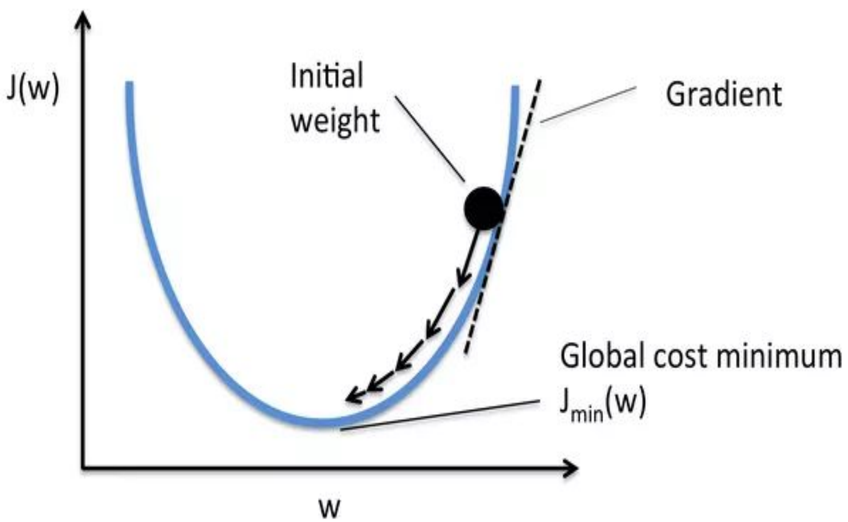
update tiap bobot yg menghubungkan neuron



Review Gradient Descent:

Steepest Descent along Error Surface

Gradient descent: a strategy for searching through a large or infinite hypothesis space that can be applied whenever (1) the hypothesis space contains continuously parameterized hypotheses (e.g., the weights in a linear unit), and (2) the error can be differentiated with respect to these hypothesis parameters.



https://medium.com/data-science-group-iitr/loss-functions-and-optimization-algorithms-demystified-bb92daff331c

Gradient

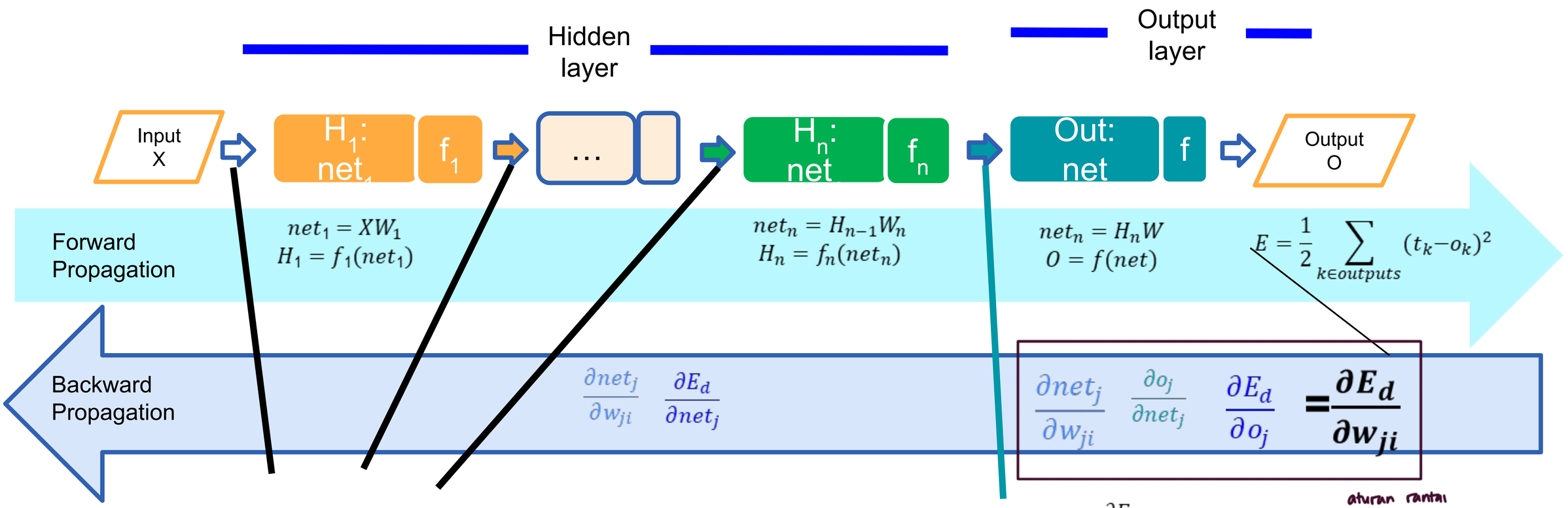
$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right]$$

$$\vec{w} \leftarrow \vec{w} + \Delta\vec{w}$$

$$\Delta\vec{w} = -\eta\nabla E(\vec{w})$$

Backpropagation for MLP

w_{ji} : weight from unit i to unit j



$$w_{ji} = w_{ji} - \eta \frac{\partial E_d}{\partial w_{ji}}$$
$$= w_{ji} - \eta[-o_j(1-o_j) \sum_{k \in outputs(j)} -\delta_k w_{kj}\ x_{ji}] = w_{ji} + \eta\delta_j x_{ji}$$

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}} = -o_j(1-o_j)\sum_{k \in outputs(j)} -\delta_k w_{kj}\ x_{ji}$$

$$w_{ji} = w_{ji} - \eta \frac{\partial E_d}{\partial w_{ji}}$$
$$= w_{ji} - \eta[-(t_j - o_j)\ o_j(1-o_j)\ x_{ji}]$$
$$= w_{ji} + \eta\delta_j x_{ji}$$

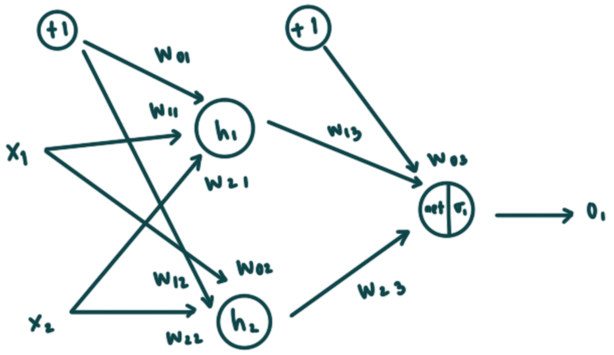
buat mau ke output doang

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}} = -(t_j - o_j)\ o_j(1-o_j)\ x_{ji}$$

$$X \longrightarrow h_1 \longrightarrow h_2 \longrightarrow h_3 \longrightarrow \dots \longrightarrow h_n \longrightarrow O$$

$$E = \frac{1}{2} \sum (t_k - o_k)^2$$

$$\sigma'(net) = \underbrace{\sigma(net)}_{o_1} \cdot \underbrace{(1 - \sigma(net))}_{(1-o_1)}$$



$$h_1 = \sigma(\underbrace{w_{01} + w_{11} \cdot x_1 + w_{21} \cdot x_2}_{net_{11}})$$

$$h_2 = \sigma(\underbrace{w_{02} + w_{12} \cdot x_1 + w_{22} \cdot x_2}_{net_{12}})$$

$$o_1 = \sigma(\underbrace{w_{03} + w_{13} \cdot h_1 + w_{23} \cdot h_2}_{net})$$

$$\frac{\partial E}{\partial w_{23}} = \frac{\partial E}{\partial o_1} \cdot \frac{\partial o_1}{\partial net} \cdot \frac{\partial net}{\partial w_{23}} = -(t_1 - o_1) \cdot o_1 (1 - o_1) \cdot h_2$$

$$\frac{\partial E}{\partial o_1} = \frac{1}{2} \cdot 2 \cdot (t_1 - o_1) \cdot -1 = -(t_1 - o_1)$$

$$\frac{\partial o_1}{\partial net} = o_1 (1 - o_1)$$

$$\frac{\partial E}{\partial w_{13}} = -(t_1 - o_1) \cdot o_1 (1 - o_1) \cdot h_1$$

$$\frac{\partial net}{\partial w_{23}} = h_2$$

$$\frac{\partial E}{\partial net} \quad \delta_1$$

$$\frac{\partial E}{\partial w_{23}} = \frac{\partial E}{\partial o_1} \cdot \frac{\partial o_1}{\partial net} \cdot \frac{\partial net}{\partial w_{23}} = -(t_1 - o_1) \cdot o_1 (1 - o_1) \cdot h_2$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial net_k} \cdot \frac{\partial net_k}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ji}}$$

$\frac{\partial net_j}{\partial w_{ji}} = x_{ji}$

$net_o = \sum w_{oi} \cdot x_{ji}$

net_j	$o_j \longrightarrow net_k$	o_j
w_{ji}		

Backpropagation for MLP with Sigmoid Unit

Steepest gradient descent proposes new weight with negative gradient and Sum of Squared Error (SSE).

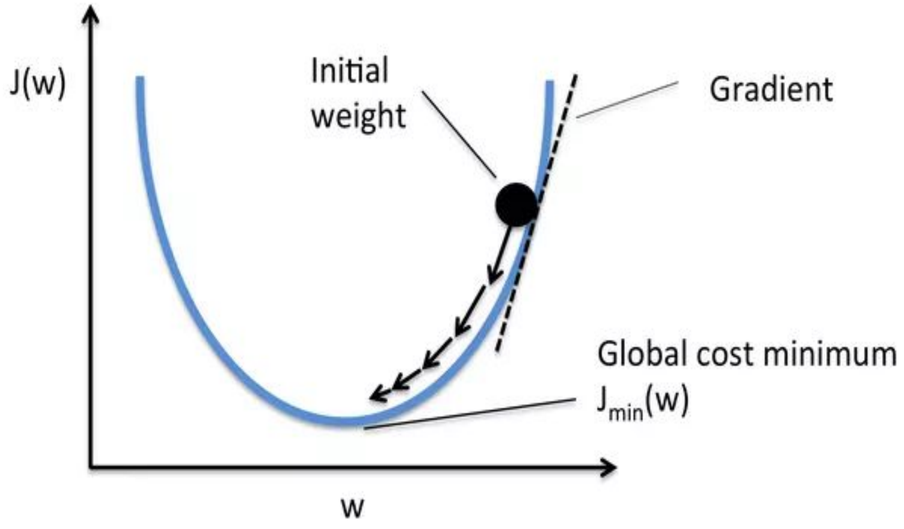
$$w_{ji} = w_{ji} + \Delta w_{ji} = w_{ji} - \eta \frac{\partial E_d}{\partial w_{ji}} = w_{ji} + \eta \delta_j x_{ji}$$

Error term for each output unit j:

$$\delta_j = o_j(1 - o_j)(t_j - o_j)$$

Error term for each hidden unit k:

$$\delta_h = o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k$$



https://medium.com/data-science-group-iitr/loss-functions-and-optimization-algorithms-demystified-bb92daff331c

Computing Gradient for Output Unit Weights

Chain rule to compute gradient:

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}} = -(t_j - o_j)\, o_j(1-o_j)\, x_{ji}$$

$$\frac{\partial net_j}{\partial w_{ji}} = x_{ji}\,; net_j = \sum_{i \in [0..nj]} x_{ji} w_{ji}$$

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial \sigma(net_j)}{\partial net_j} = o_j(1-o_j)$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j}\frac{1}{2}\sum_{k \in outputs}(t_k - o_k)^2 = \frac{1}{2}.2.(t_j - o_j).-1 = -(t_j - o_j)$$

Computing Gradient for Output Unit Weights

Chain rule to compute gradient:

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}} = -(t_j - o_j)\, o_j(1-o_j)\, x_{ji}$$

gradien

$$\delta_j = o_j(1-o_j)(t_j - o_j)$$

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}} = -\delta_j x_{ji}$$

Computing Gradient for Hidden Unit Weights

Chain rule to compute gradient:

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}} = o_j(1-o_j)\sum_{k\in outputs(j)} -\delta_k w_{kj}\, x_{ji}$$

$$\frac{\partial net_j}{\partial w_{ji}} = x_{ji}\,; net_j = \sum_{i\in[0..nj]} x_{ji} w_{ji}$$

$$\frac{\partial E_d}{\partial net_j} = \sum_{k\in outputs(j)} \frac{\partial E_d}{\partial net_k}\frac{\partial net_k}{\partial o_j}\frac{\partial o_j}{\partial net_j} = \sum_{k\in outputs(j)} -\delta_k w_{kj} o_j(1-o_j)$$

$$\frac{\partial E_d}{\partial net_j} = -\delta_j$$

Computing Gradient for Hidden Unit Weights

Chain rule to compute gradient:

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}} = -o_j(1-o_j)\sum_{k\in outputs(j)}\delta_k w_{kj}\, x_{ji}$$

$$\delta_h = o_h(1-o_h)\sum_{k\in outputs(h)}\delta_k w_{kh}$$

$$\frac{\partial E_d}{\partial w_{ji}} = -\delta_h x_{ji}$$

BACKPROPAGATION($training\_examples, \eta, n_{in}, n_{out}, n_{hidden}$)

Each training example is a pair of the form $\langle \vec{x}, \vec{t} \rangle$, where $\vec{x}$ is the vector of network input values, and $\vec{t}$ is the vector of target network output values.

η is the learning rate (e.g., .05). n_{in} is the number of network inputs, n_{hidden} the number of units in the hidden layer, and n_{out} the number of output units.

The input from unit i into unit j is denoted x_{ji}, and the weight from unit i to unit j is denoted w_{ji}.

- Create a feed-forward network with n_{in} inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers (e.g., between $-.05$ and .05).
- Until the termination condition is met, Do
  - For each $\langle \vec{x}, \vec{t} \rangle$ in *training_examples*, Do

Propagate the input forward through the network:

1. Input the instance $\vec{x}$ to the network and compute the output o_u of every unit u in the network.

Propagate the errors backward through the network:

2. For each network output unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k) \qquad \text{(T4.3)}$$

3. For each hidden unit h, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k \qquad \text{(T4.4)}$$

4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

where

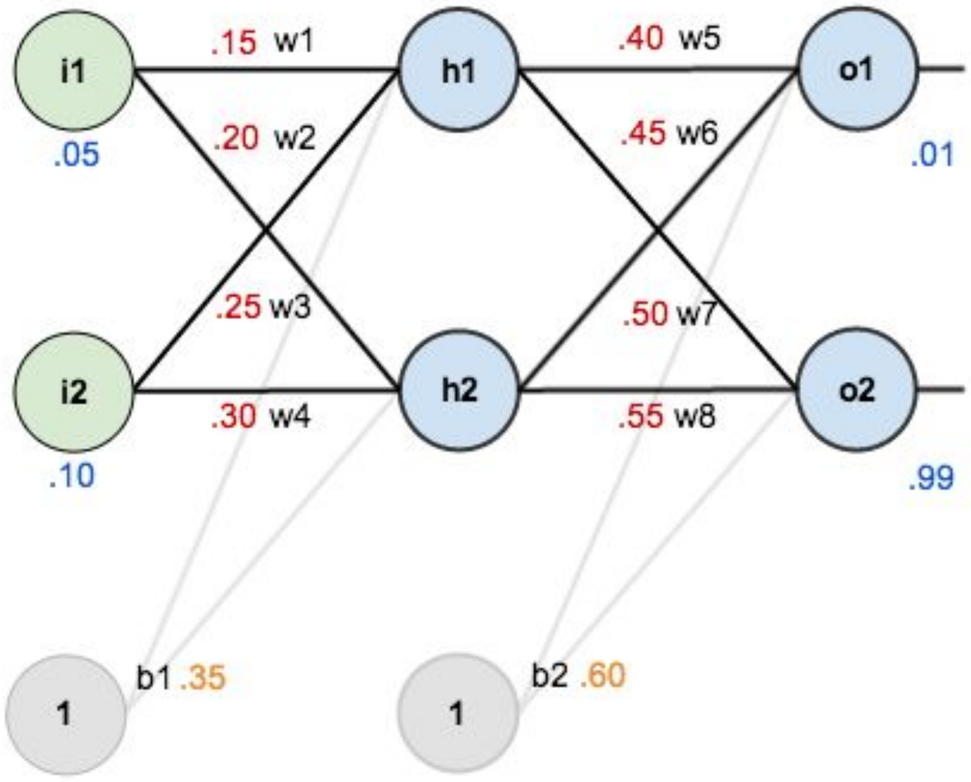
$$\Delta w_{ji} = \eta \, \delta_j \, x_{ji} \qquad \text{(T4.5)}$$

Backpropagation Algorithm (Mitchell, 1997)

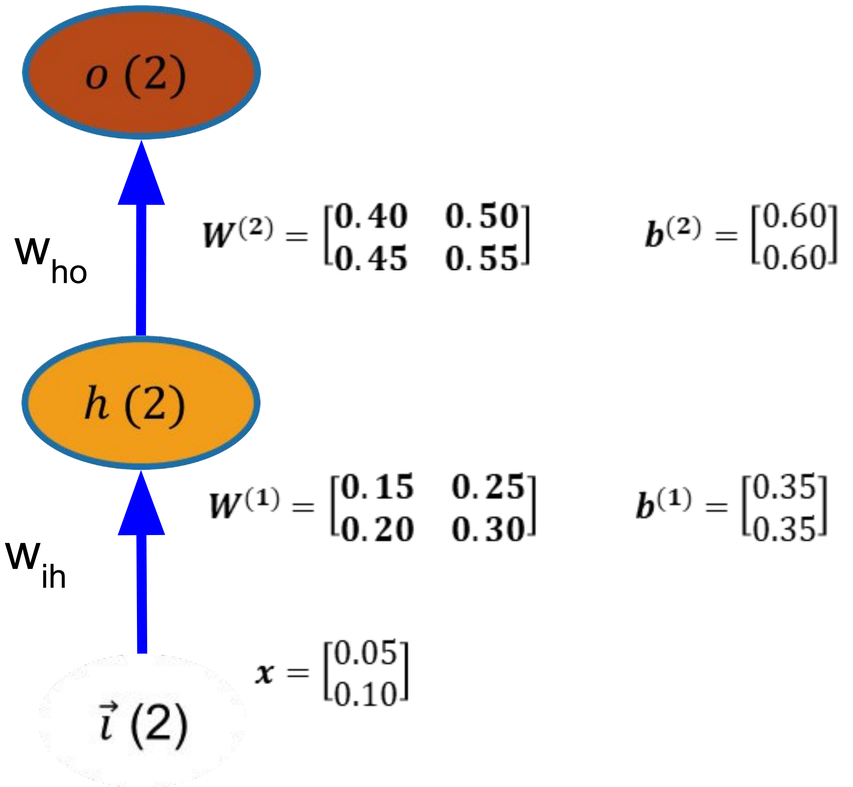
Termination condition:
- a fixed number of iterations
- the error on the training examples < threshold
- the error on a separate validation set of examples meets some criterion

Backpropagation: Example

(from http://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/)



Compact style: sigmoid model



Backpropagation: Example

$Learning\ rate: \eta = 0.5$

k	h
0	$\begin{bmatrix}0.05\\0.10\end{bmatrix}$
1	$\begin{bmatrix}0.5933\\0.5969\end{bmatrix}$
2	$\begin{bmatrix}0.7514\\0.7729\end{bmatrix}$

$\delta_{o_1} = 0.7514 * (1 - 0.7514) * (0.01 - 0.7514) = -0.1385$

$\delta_{o_2} = 0.7729 * (1 - 0.7729) * (0.99 - 0.7729) = 0.0381$

$\delta_{h_1} = 0.5933 * (1 - 0.5933) * [w_5 * \delta_{o_1} + w_7 * \delta_{o_2}]$

$= 0.2413 * [0.4 * (-0.1385) + 0.5 * 0.0381]$

Calculate also for δ_{h_2}

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k$$

Example updating for w5 ($w_{o_1 h_1}$)

$w_5 = 0.4 + (0.5 * \delta_{o_1} * h_1$

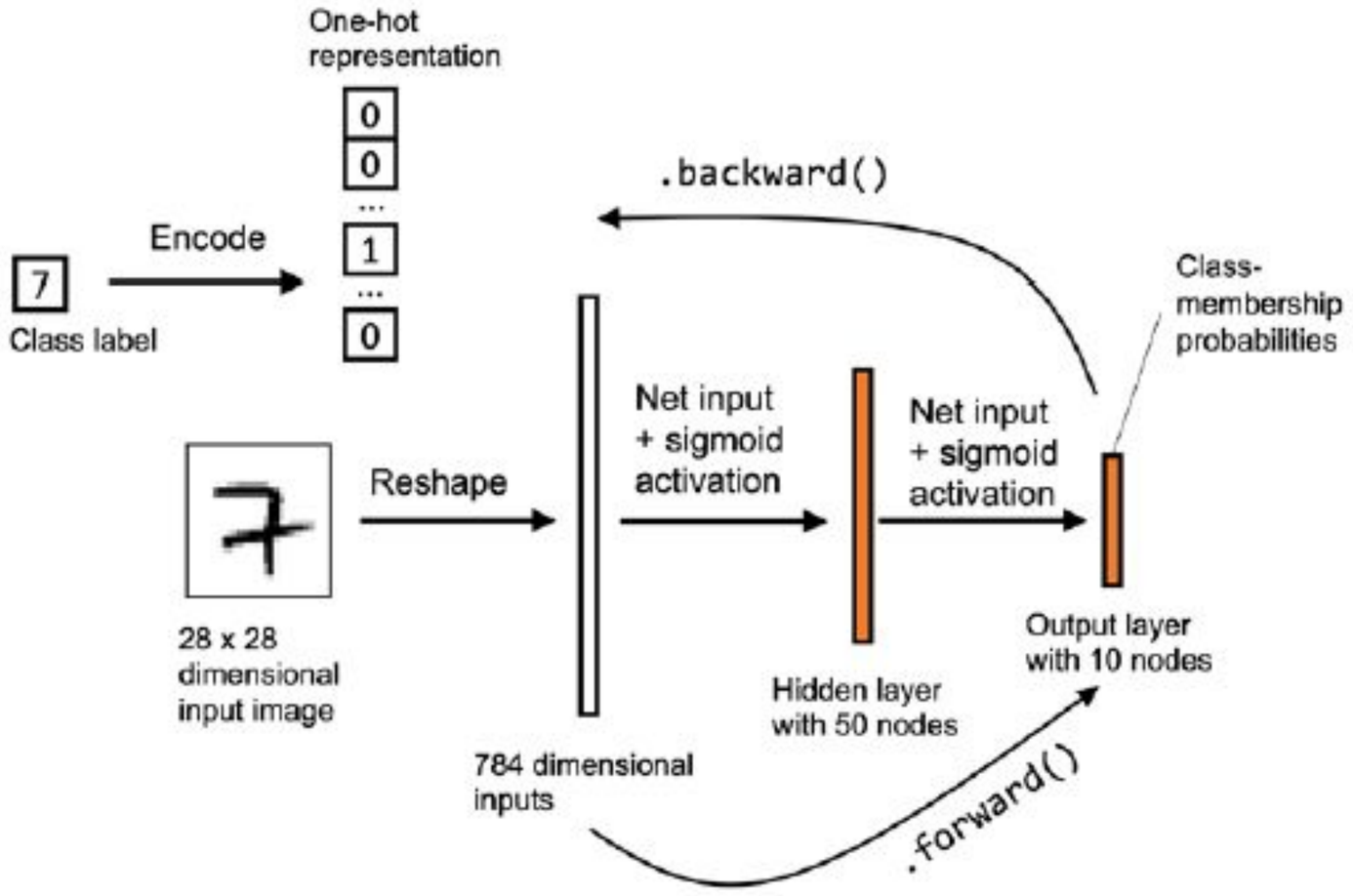
$= 0.4 + (0.5 * (-0.1385) * 0.5933)$

$= 0.3589$

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

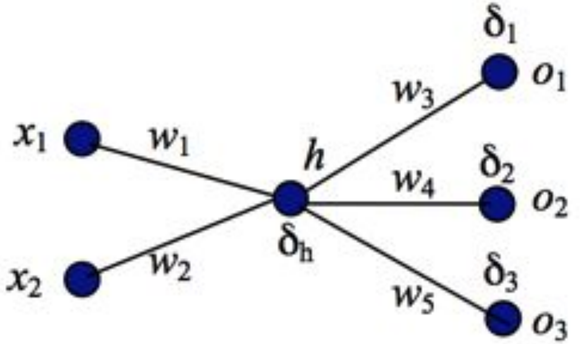
$$\Delta w_{ji} = \eta\, \delta_j\, x_{ji}$$

Example of NN Architecture for labeling handwritten digits



Raschka, *Figure 11.6: The NN architecture for labeling handwritten digits*

Latihan 1



Diketahui sebuah arsitektur jaringan Neural Network sebagai berikut yang memiliki dua unit input *x*, tiga unit output *o* dan satu unit hidden *h*, serta menggunakan fungsi aktivasi Sigmoid di setiap unitnya. Proses pembelajaran menerapkan algoritma Backpropagation.

a. Tuliskan formula untuk menghitung luaran *h* dan o_1

b. Jika δ_1, δ_2 dan δ_3 adalah *error* pada ketiga unit output, δ_h adalah *error* pada hidden unit, dan t_1 adalah nilai target untuk unit output yang pertama, tuliskan formula untuk menghitung δ_1 dan δ_h.

c. Jika α adalah *learning rate*, tuliskan formula untuk *update* bobot w_1 dan w_3.

Catatan: bias selalu ada sebagai input tiap layer, simbol bebas

$$h = \sigma(w_0 + w_1 \cdot x_1 + w_2 \cdot x_2)$$

$$o_1 = \sigma(w_6 + h \cdot w_3)$$

$$\delta_k = o_k (1 - o_k)(t_k - o_k)$$

$$\delta_h = o_k (1 - o_k) \cdot \sum w_{kh} \cdot \delta_k$$

$$\delta_1 = o_1 (1 - o_1) \cdot (t_1 - o_1)$$

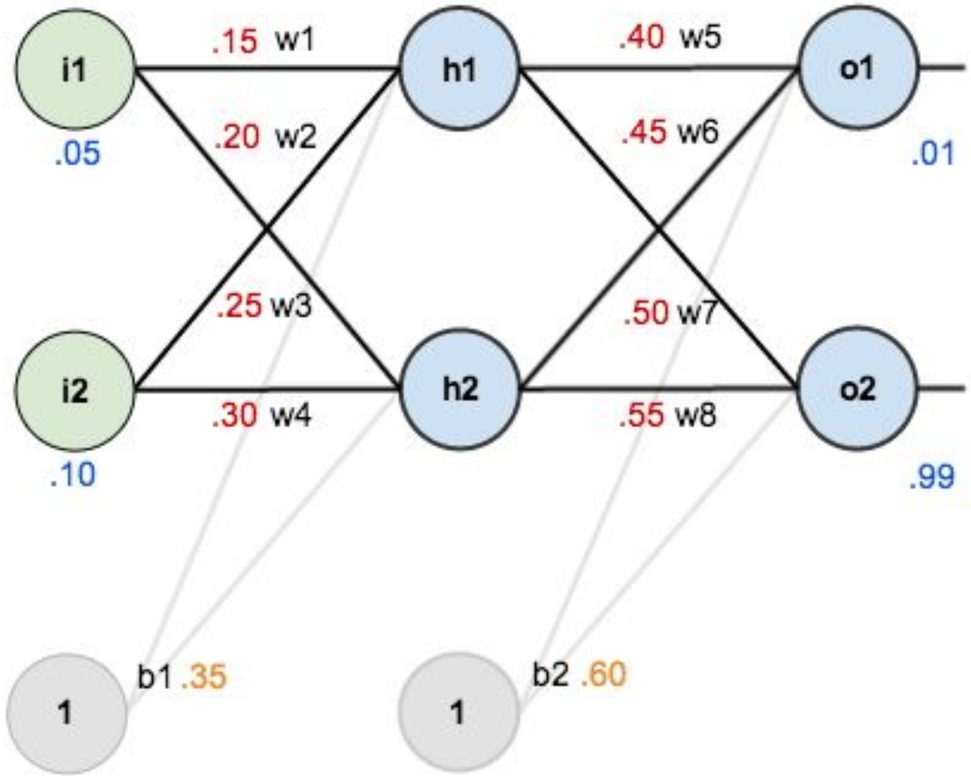
$$\delta_h = o_1 (1 - o_1) \left(w_3 \cdot \delta_1 + w_4 \cdot \delta_2 + w_5 \cdot \delta_3 \right)$$

$$w_3 = w_3 + \eta \cdot \delta_1 \cdot h$$

$$w_1 = w_1 + \eta \cdot \delta_h \cdot x_1$$

Latihan 2

- Lakukan pembelajaran (backpropagation) untuk ANN gambar di samping untuk mendapatkan nilai semua bobot yang baru, lengkap dengan perhitungan.
- Terapkan bobot yang baru pada data input i1= 0.05 dan i2 = 0.1
- Hitung error setelah update bobot, dan bandingkan dengan sebelum update bobot



Fungsi Aktivasi: Sigmoid
Learning Rate: 0.5

Thank you