IF3270 Pembelajaran Mesin

Perceptron

Tim Pengajar IF3270

Review

Logistic Regression & SVM → 1 Neuron

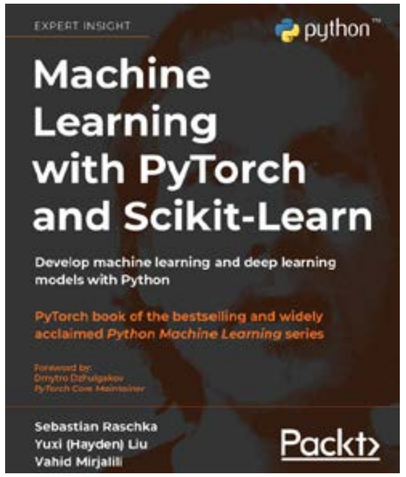
- Machine learning (ML) overview
    - AI vs ML
    - Learning Type:
        - Supervised Learning: Classification, Regression
        - Unsupervised Learning
        - Reinforcement Learning

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

- Today: Perceptron → Supervised Learning

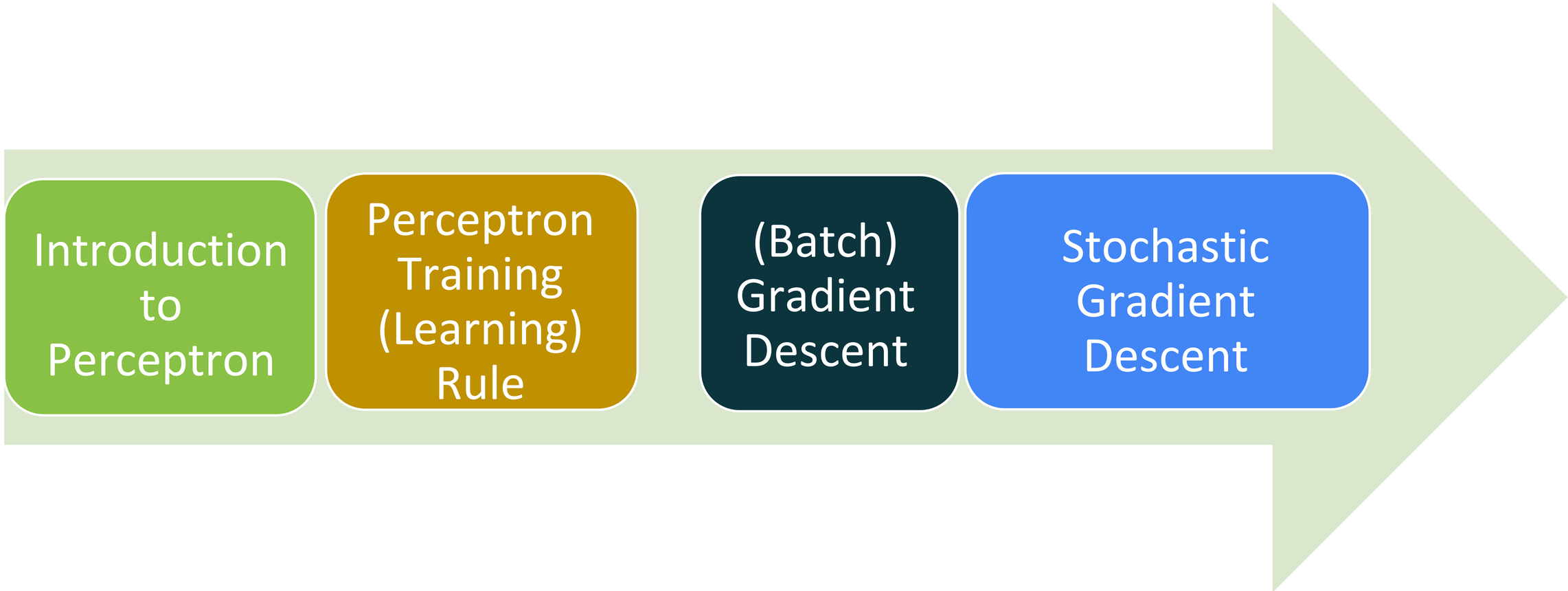
Reference

Mitchell, T.,
Machine Learning,
1997, McGraw-Hill

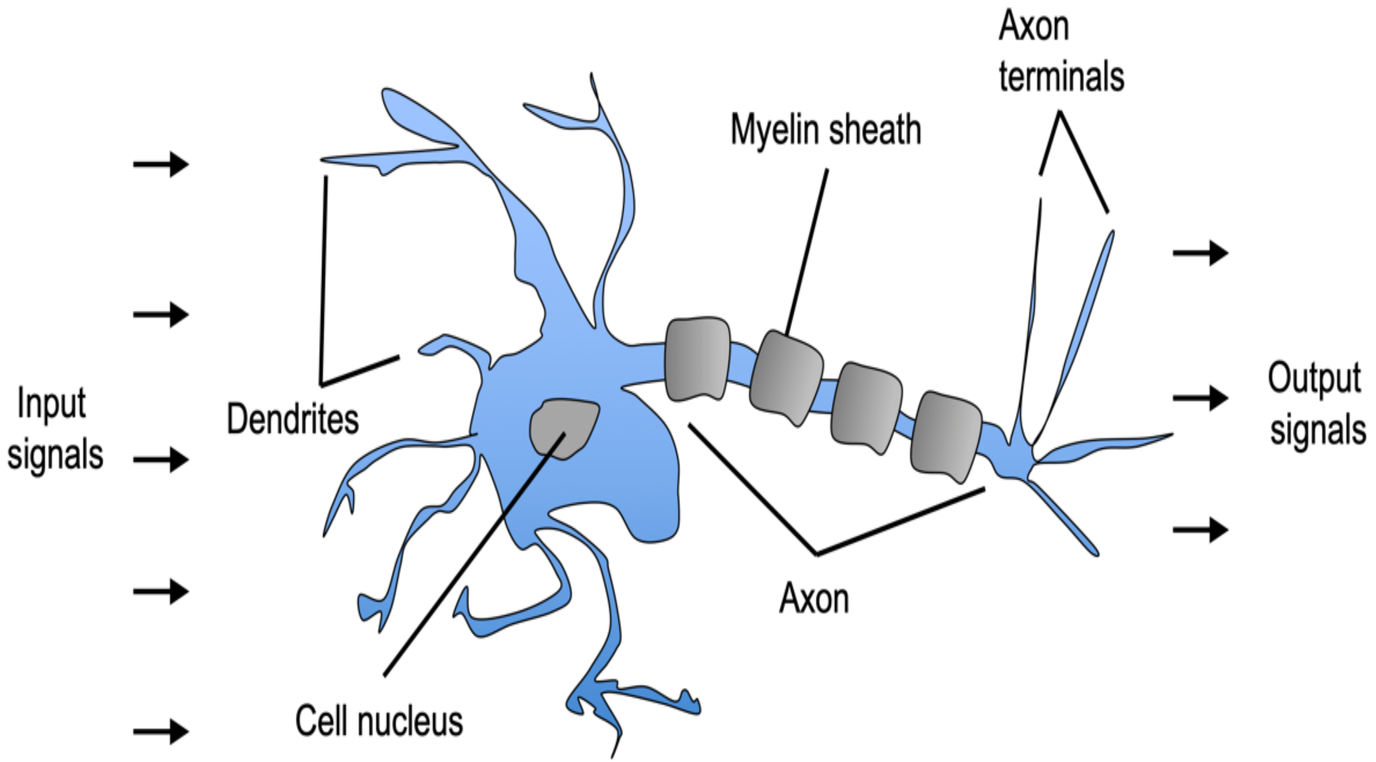


Sebastian Raschka; Yuxi (Hayden) Liu; and Vahid Mirjalili, Machine Learning with PyTorch and Scikit-Learn, 2022, Packt Publishing

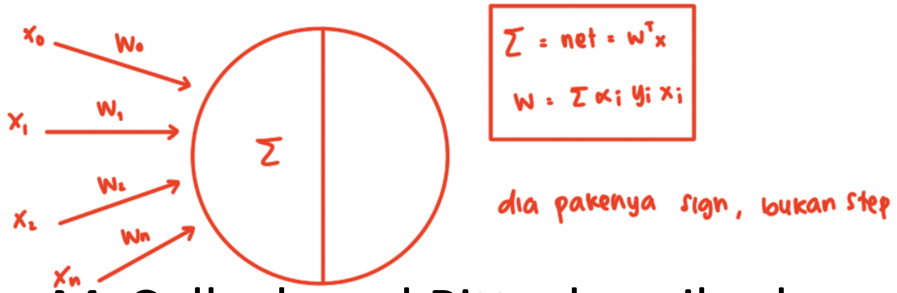
Outline


Introduction to Perceptron
Perceptron Training (Learning) Rule
(Batch) Gradient Descent
Stochastic Gradient Descent

A Simplified Brain Cell



A neuron processing chemical and electrical signals



McCulloch and Pitts described (Raschka, 2022) → a simple logic gate with binary outputs:

- Multiple signals arrive at the dendrites,
- Integrated into the cell body,
- If the accumulated signal exceeds a certain threshold, an output signal is generated that will be passed on by the axon

Formal Definition of a Neuron (context: Binary Classification)

sign labelnya hny 0/1

- Real-valued input: **x** ($\bar{x}$) → vektor
- Corresponding weight vector (contribution of input x_i): **w**
- Threshold: θ (bias, w_0/ b)

→ Linear combination net input and bias:

$$z = w_0 + w_1\bar{x}_1 + w_2\bar{x}_2 + \ldots + w_m\bar{x}_m$$

bias

- Output: 1 if the linear combination exceed certain threshold: default utk ujian → sign function

Sign function: $\hat{y} = \sigma(z) = \begin{cases} 1, & z > 0 \\ -1, & otherwise \end{cases}$

Step function: $\hat{y} = \sigma(z) = \begin{cases} 1, & z \geq 0 \\ 0, & otherwise \end{cases}$

Perceptron

Linear model
Linearly separable dataset

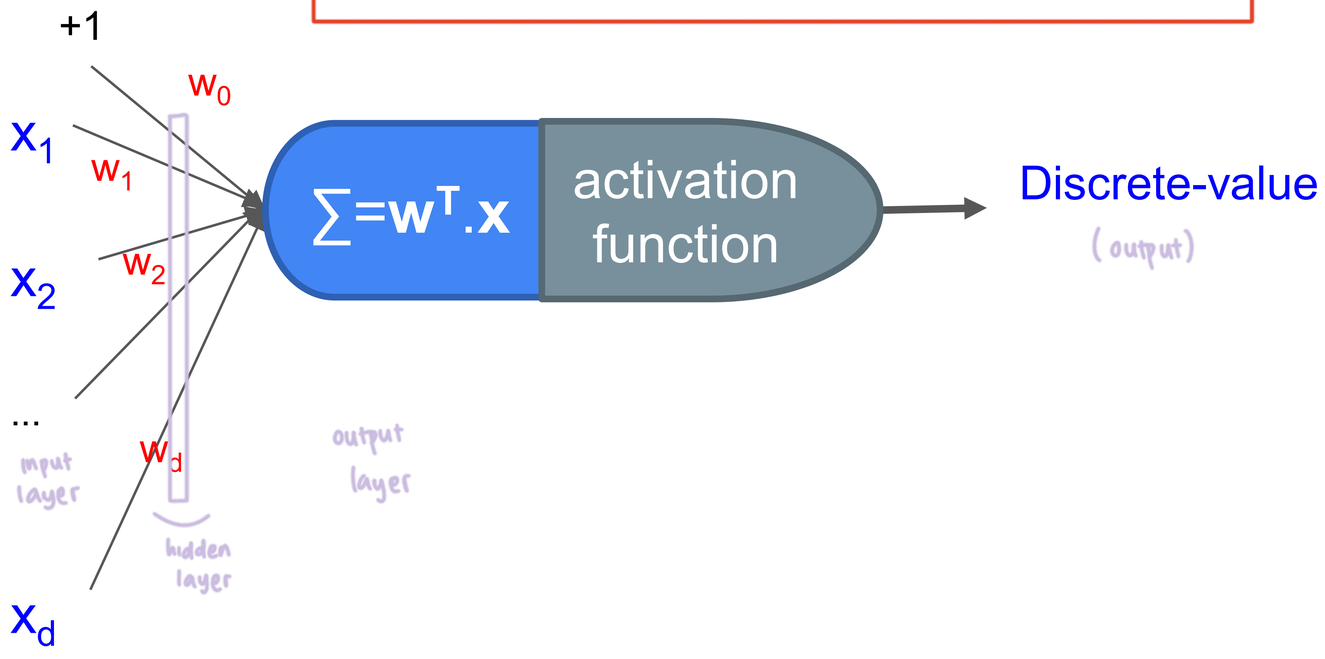
Simple Neural Network
ANN based on a unit

MLP (Multiple Layer Perceptron)

Learning output: $\mathbf{w} \in \Re^{d+1}$
Hypothesis space $H=\{\mathbf{w} \mid \mathbf{w} \in \Re^{d+1}\}$

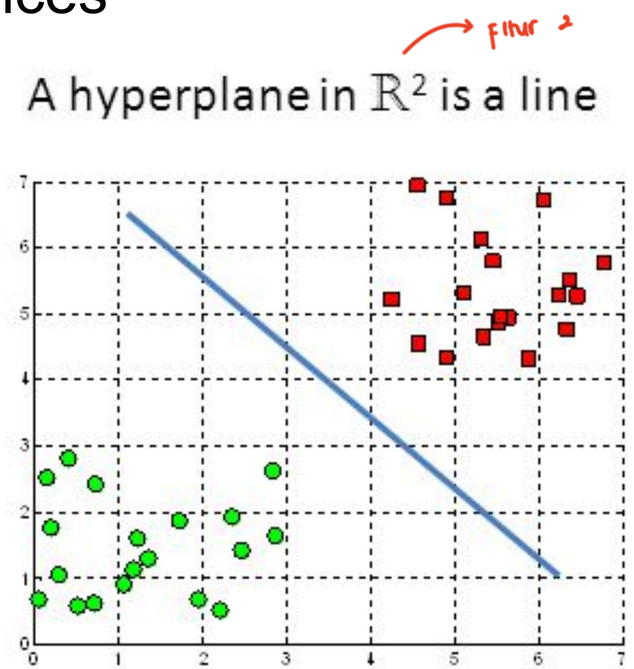
$$\boldsymbol{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$$

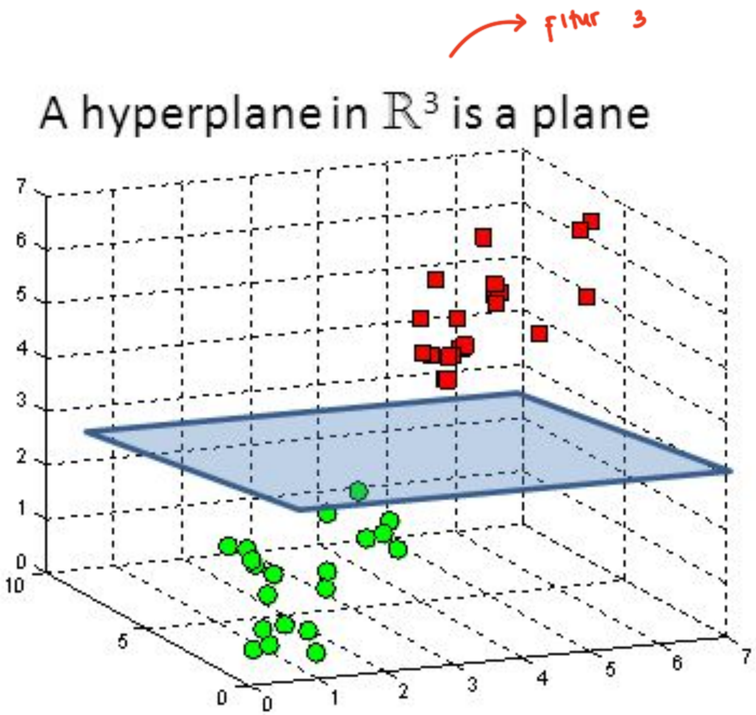
$$\hat{y} = \boldsymbol{\sigma}(\boldsymbol{w}^T \boldsymbol{x}) = \sigma(w_0 . 1 + w_1 x_1 + \cdots + w_d x_d)$$



Perceptron

Represent hyperplane decision surface in the n-dimensional space of instances





A hyperplane in $\mathbb{R}^n$ is an n-1 dimensional subspace

Sumber: http://images.slideplayer.com/5/1579281/slides/slide_32.jpg

bentuk dimensi hyperplanenya tergantung jumlah fitur

Perceptron easily represent m-of-n function

kelstimewaan perceptron

m-of-n functions: functions where at least m of the n inputs must be true

m-of-n function

AND m=n

OR m=1

fungn yg nerima boolean ky gerbang logika

Perceptron for m-of-n function

1. set all input weights to the same value (e.g., 0.5)
2. setting the threshold w_0.

Perceptron: Example

AND: sign(x1+x2-1)

OR: sign(2x1+2x2-1)

fungn estimasi

Show that perceptron with w0=-0.8 and w1=w2=0.5, represent AND function.

ada 4 chance

T T
T F
F T
F F

Mitchell, T. (1997). Machine learnig. McGraw-Hill. Chapter 4.4
https://medium.com/@stanleydukor/neural-representation-of-and-or-not-xor-and-xnor-logic-gates-perceptron-algorithm-b0275375fea1

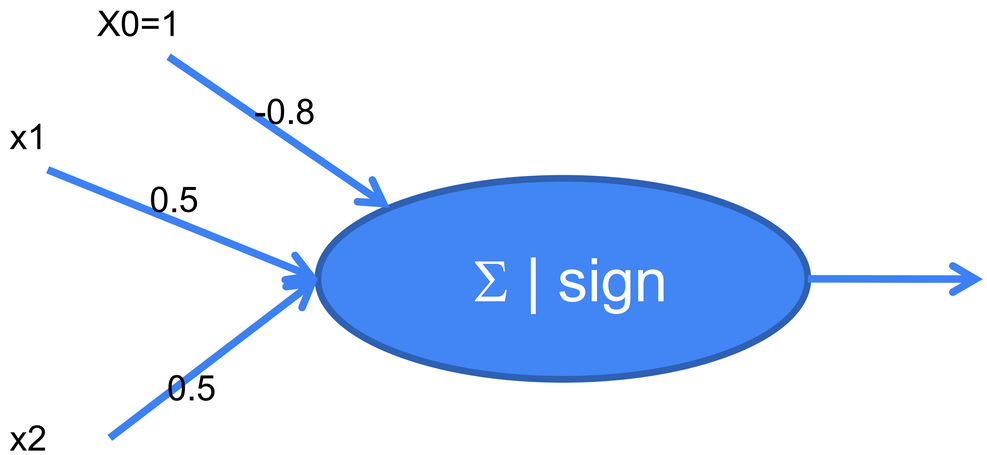
Perceptron: Example 1

Suppose we have 2 input attributes x1 and x2 $\in$ {-1,1}, and 1 output attribute o $\in$ {-1,1}. Given following training set:

x1	x2	o(x1,x2)
1	1	+1
1	-1	-1
-1	1	-1
-1	-1	-1

Then the learning problem is to find weight w1 and w2 and threshold (or bias) value such that the computed output of our network (which is given by the linear threshold function) is equal to the desired output for all examples.

AND Perceptron: m-of-n function



$$sign(net) = \begin{cases} 1, net > 0 \\ -1, otherwise \end{cases}$$

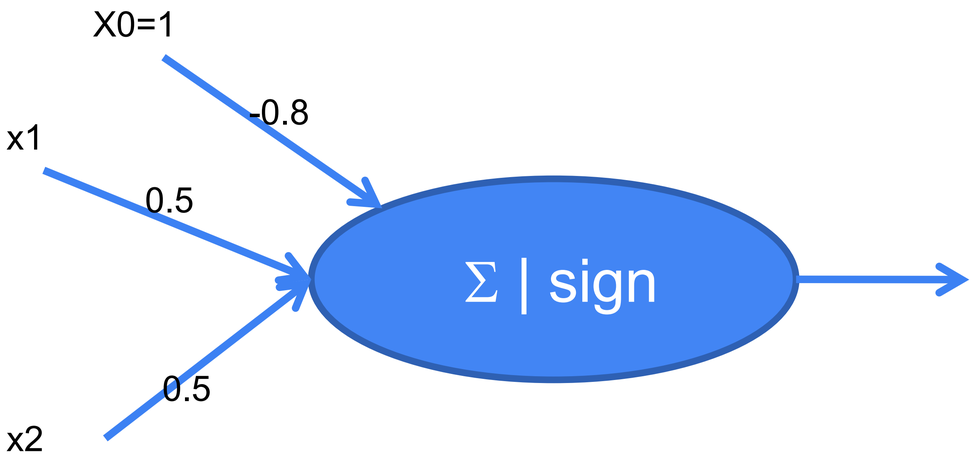
AND: m=n=2

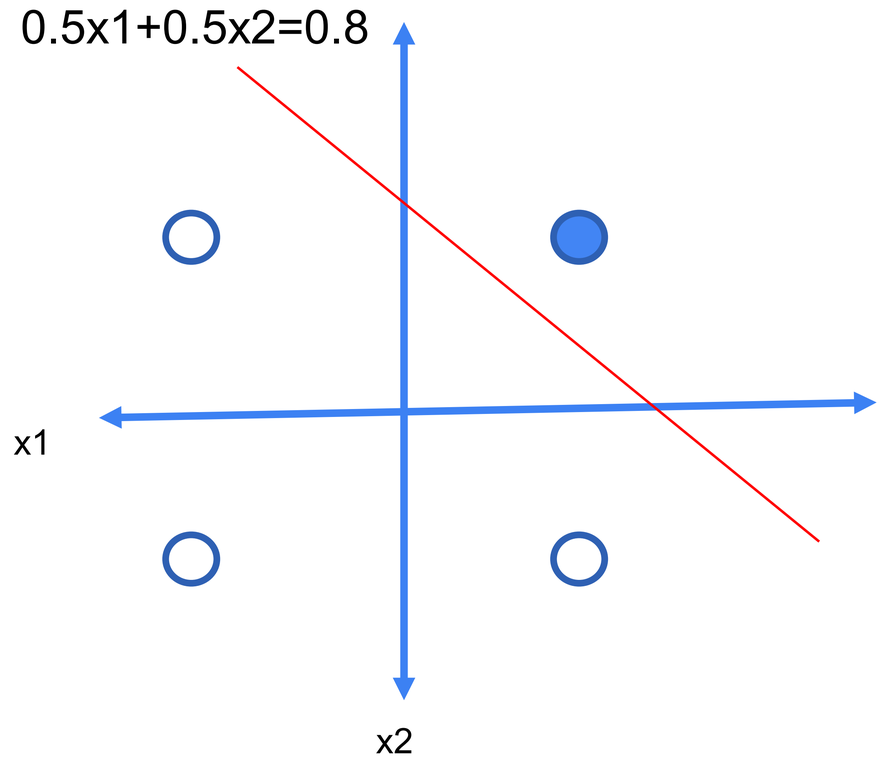
x0	x1	x2	∑	sign
1	1	1	0.2	+1
1	1	-1	-0.8	-1
1	-1	1	-0.8	-1
1	-1	-1	-1.8	-1

set all input weights to the same value and then setting the threshold wo.
e.g., w1=w2=0.5 → w0=-0.8

AND Perceptron

x0	x1	x2	Σ	sign
1	1	1	0.2	+1
1	1	-1	-0.8	-1
1	-1	1	-0.8	-1
1	-1	-1	-1.8	-1





$$o(x1, x2) = \begin{cases} 1, 0.5x1 + 0.5x2 > 0.8 \\ -1, otherwise \end{cases}$$

Perceptron: Learning algorithm

Perceptron Training Rule

proven to be converged if training data is linearly separable and learning rate (η) sufficiently small.

paling sederhana

Batch Gradient Descent

kompleks krn turunan

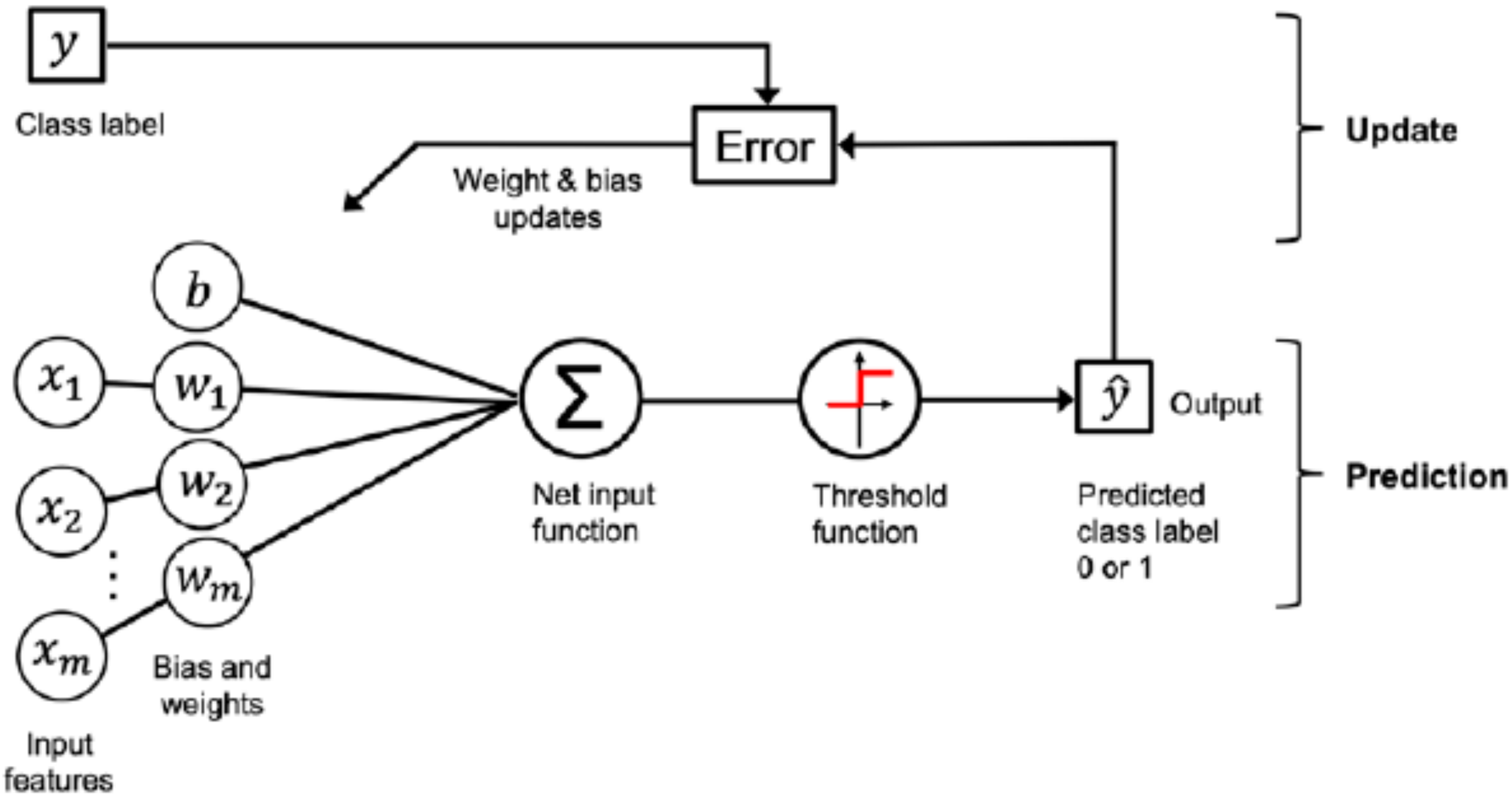
Stochastic Gradient Descent

kompleks krn turunan

Perceptron Training (Learning) Rule

selalu forward propagation baru backward

Perceptron Training (Learning) Rule



Perceptron Training rule

input: Training data D={<x_1,y_1>…<x_n,y_n>}; learning rate η;
activation function: sign

D: {<[1,0,1],-1>,<[0,-1,-1],+1>}; η=0.1

data 3 dimensi → w = 4
data 4 dimensi → w = 5

biar perceptron, selalu 1

x_0	x1	x2	x3	y
1	1	0	1	-1
1	0	-1	-1	+1

Initialize randomly for **w**

repeat

E←0 setiap awal iterasi error selalu diset 0 karena dia konvergen

For each example <xi,yi>:

oi=prediction for xi using the current coefficients **w**

If oi==yi then do nothing
else #update
$w_j \leftarrow w_j + \Delta w_j$ dgn $\Delta w_j = \eta (yi-oi) x_{ij}$
(weight sblmnya; learning rate)

E←E+(yi-oi)^2/2 #update error

Until E==0 bakal iterasi terus sampe E = 0

Return **w**

w=[0,0,0,0]
#w0=0; w1=0; w2=0, w3=0

epoch 1: E=0
<[1,0,1],-1>:
oi=sign(0)=-1 → do nothing
<[0,-1,-1],+1>:
oi=sign(0)=-1
w0=0+0.1(1-(-1))*1=0.2
w1=0+0.1(1-(-1))*0=0
w2=0+0.1(1-(-1))*-1= -0.2
w3=0+0.1(1-(-1))*-1= -0.2
E=2
epoch 2: E=0

Perceptron: Example

- Suppose we have 3 input attributes x1, x2, x3 $\in \Re$, and 1 output attribute o $\in$ {-1,1}. Given following training set:

x1	x2	x3	o(x1,x2)
1	0	1	-1
0	-1	-1	+1
-1	-0.5	-1	+1

Then the learning problem is to find weight w0, w1, w2, w3 values such that the computed output of our network (which is given by the sign function) is equal to the desired output for all examples.

Perceptron Learning Rule

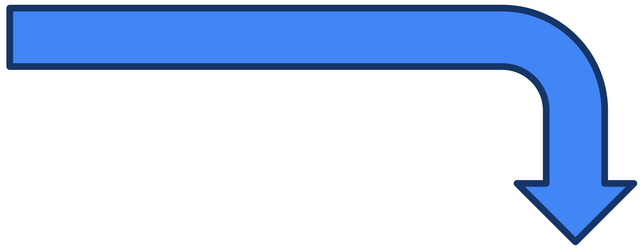
learning rate				0.1							
Fungsi aktivasi				sign							
Perhitungan error sepanjang iterasi											
Iterasi 1	x1	x2	x3	Target	w0	w1	w2	w3	sigma	sign	Err
	1	0	1	-1	0.00	0.00	0.00	0.00	0.00	-1	0
	0	-1	-1	1	0.00	0.00	0.00	0.00	0.00	-1	2
	-1	-0.5	-1	1	0.20	0.00	-0.20	-0.20	0.50	1	2
					0.20	0.00	-0.20	-0.20			
Iterasi 2	x1	x2	x3	Target	w0	w1	w2	w3	sigma	sign	Err
	1	0	1	-1	0.20	0.00	-0.20	-0.20	0.00	-1	0
	0	-1	-1	1	0.20	0.00	-0.20	-0.20	0.60	1	0
	-1	-0.5	-1	1	0.20	0.00	-0.20	-0.20	0.50	1	0
					0.20	0.00	-0.20	-0.20			

terminate

(Batch) Gradient Descent

Gradient Descent

If the data is not linearly separable → perceptron rule can fail to converge



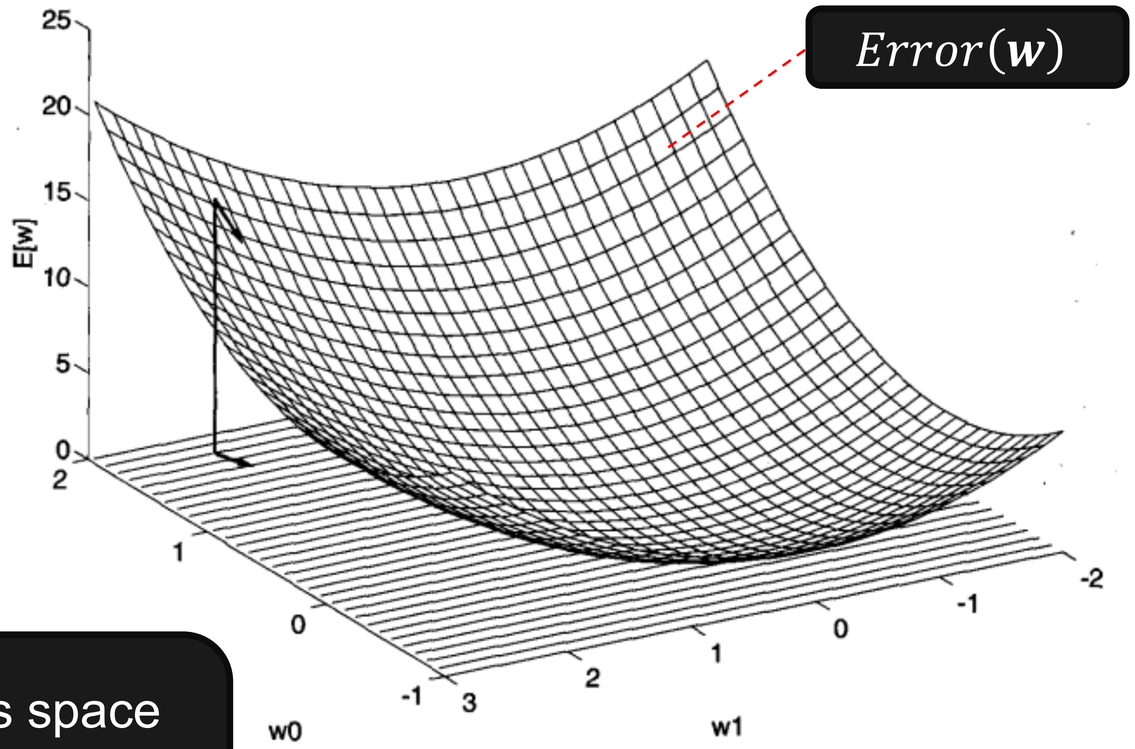
Delta Rule

If the training examples are not linearly separable → converges toward a best-fit approximation to the target concept

Use ***gradient descent*** to search the hypothesis space

Gradient Descent: What

maunya dia makin lama makin turun sampai nilai terendah, misal error = 0



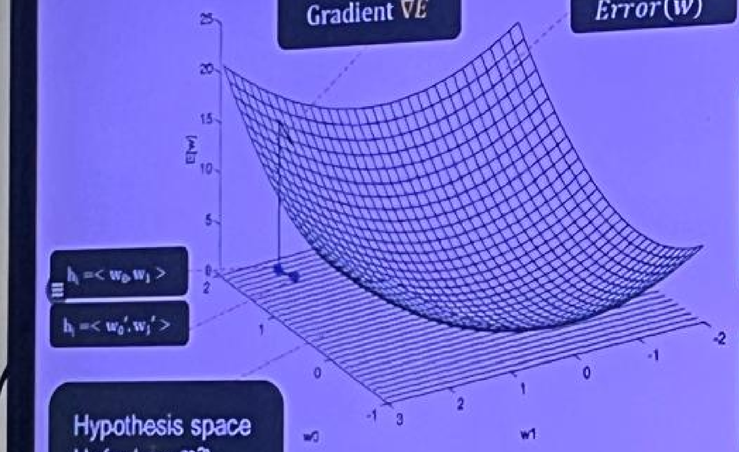
Hypothesis space
H={**w** | **w**

5040-0522

Gradient Descent: What

Gradient ∇E

$Error(w)$



$h_i = < w_0, w_1 >$

$h_i = < w_0', w_1' >$

Hypothesis space
$H=\{w \mid w \in \Re^2\}$
continuously parameterized hypotheses

Searching strategy using gradient of landscape to find minimum error

Gradient gives magnitude and direction of steepest slope

Gradient: partial derivatives

$$\nabla E(w) = [\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_d}]$$

Training rule: $w \leftarrow w + \Delta w$

$$\Delta w = -\eta \nabla E(w)$$

Toolbar

21

batch → utk semua data dia cm hitung Δw_k

Gradient Descent: Training rule

$$\nabla E(\boldsymbol{w}) = [\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_d}]$$

$$\frac{\partial E}{\partial w_k} = \frac{\partial}{\partial w_k} \frac{1}{2} \sum_{d \epsilon D} (y_d - o_d)^2$$

$$= \frac{1}{2} \sum_{d \epsilon D} \frac{\partial}{\partial w_k} (y_d - o_d)^2$$

$$= \frac{1}{2} \sum_{d \epsilon D} 2(y_d - o_d) \frac{\partial}{\partial w_k} (y_d - o_d)$$

$$= \sum_{d \epsilon D} (y_d - o_d) \frac{\partial}{\partial w_k} (y_d - \mathbf{w}.\mathbf{x}_d)$$

$$\frac{\partial E}{\partial w_k} = \sum_{d \epsilon D} (y_d - o_d)(-x_{dk})$$

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

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

$$\Delta w_k = -\eta \frac{\partial E}{\partial w_k}$$

$$= -\eta \sum_{d \epsilon D} (y_d - o_d)(-x_{dk})$$

$$= \eta \sum_{d \epsilon D} (y_d - o_d)(x_{dk})$$

Batch gradient descent

η : gradient descent step size

Batch Gradient Descent for perceptron training

input: Training data D={<x_1,y_1>…<x_n,y_n>}; learning rate η , Error threshold θ

D: {<[1,0,1],-1>,<[0,-1,-1],+1>}; η=0.3; θ=0.01

Initialize each w_k to some small random value

Until the termination condition is met, do

- Initialize each Δw_k to zero ; E←0
- For each example <$\mathbf{x}_i$,y_i> do
  - o_i←prediction for $\mathbf{x}_i$ using current w
  - For each linear unit weight w_k, do
    $\Delta w_k \leftarrow \Delta w_k + \eta (y_i - o_i) x_{ik}$
    hmy dihitung & tdk lgsg update
  - E←E+(yi-oi)^2/2 #update error
- For each linear unit weight w_k, do
  $w_k \leftarrow w_k + \Delta w_k$

Return **w**

w=[0,0,0,0] #w0=0; w1=0; w2=0, w3=0

epoch 1: Δw_k=0;k∈[0..3];E=0

<[1,0,1],-1>:
oi=0 gapake sign
Δw_0=0+0.3(-1-0)*1=-0.3
Δw_1=0+0.3(-1-0)*1=-0.3
Δw_2=0+0.3(-1-0)*0=0
Δw_3=0+0.3(-1-0)*1=-0.3
E=0+(-1-0)^2/2=0.5
<[0,-1,-1],+1>:
oi=0 gapake sign
Δw_0=-0.3+0.3(1-0)*1=0
Δw_1=-0.3+0.3(1-0)*0=-0.3
Δw_2=0+0.3(1-0)*-1=-0.3
Δw_3=-0.3+0.3(1-0)*-1=-0.6
E=0.5+(1-0)^2/2=1
w=[0,-0.3,-0.3,-0.6]

epoch 2: Δw_i=0;i∈[0..3]

Batch Gradient Descent: Example

learning rate	0.3
Error threshold	0.010
Fungsi aktivasi	linear

batch → semua data dipake dulu ttr dpt Δw di akhir iterasi

Iterasi 1	x1	x2	x3	Target	w0	w1	w2	w3	sigma	dw0	dw1	dw2	dw3	Err
	1	0	1	-1	0.00	0.00	0.00	0.00	0.00	-0.30	-0.30	0.00	-0.30	0.500
	0	-1	-1	1	0.00	0.00	0.00	0.00	0.00	0.00	-0.30	-0.30	-0.60	1.000
	-1	-0.5	-1	1	0.00	0.00	0.00	0.00	0.00	0.30	-0.60	-0.45	-0.90	1.500
					0.30	-0.60	-0.45	-0.90						

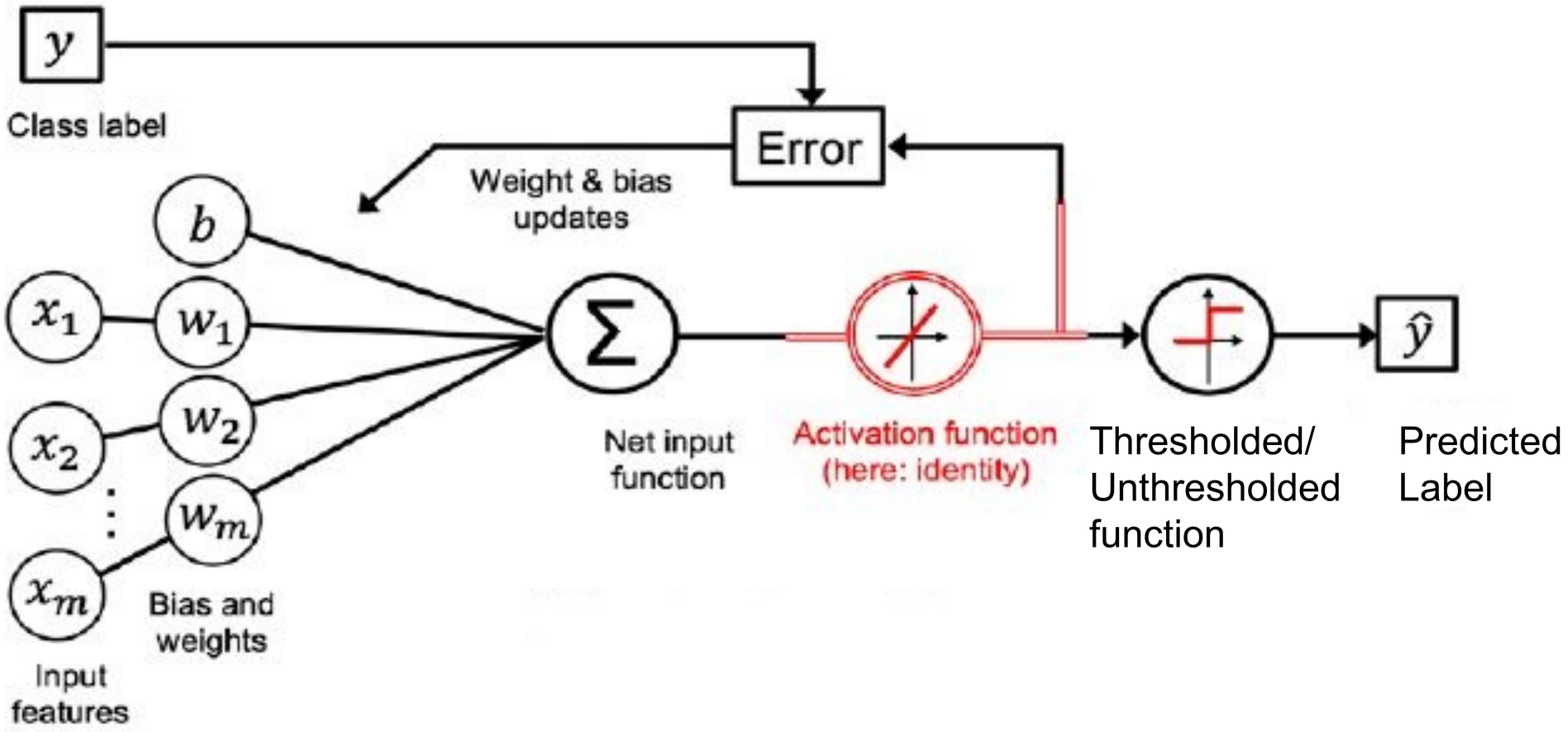
Iterasi 2	x1	x2	x3	Target	w0	w1	w2	w3	sigma	dw0	dw1	dw2	dw3	Err
	1	0	1	-1	0.30	-0.60	-0.45	-0.90	-1.20	0.06	0.06	0.00	0.06	0.020
	0	-1	-1	1	0.30	-0.60	-0.45	-0.90	1.65	-0.14	0.06	0.20	0.26	0.231
	-1	-0.5	-1	1	0.30	-0.60	-0.45	-0.90	2.03	-0.44	0.37	0.35	0.56	0.757
					-0.14	-0.23	-0.10	-0.34						

Iterasi 10	x1	x2	x3	Target	w0	w1	w2	w3	sigma	dw0	dw1	dw2	dw3	Err
	1	0	1	-1	-0.03	-0.29	-0.37	-0.69	-1.01	0.00	0.00	0.00	0.00	0.000
	0	-1	-1	1	-0.03	-0.29	-0.37	-0.69	1.03	0.00	0.00	0.01	0.01	0.000
	-1	-0.5	-1	1	-0.03	-0.29	-0.37	-0.69	1.13	-0.04	0.04	0.03	0.05	0.009
					-0.07	-0.25	-0.34	-0.64						

↳ pake sigma, bkn sign

Delta Rule with Gradient Descent

Also known as: Least Mean Square (LMS) rule, Adaptive Linear Neuron (Adaline) rule, Widrow-Hoff rule



Batch Gradient Descent → konvergen ke global minimum

local search → konvergen ke lokal minimum (hoping it's global minimum)

- Computes weight updates after summing over all the training examples in D
- If the training examples are not linearly separable, it converges toward a best-fit approximation to the target concept, given a sufficiently small learning rate η (step size) is used

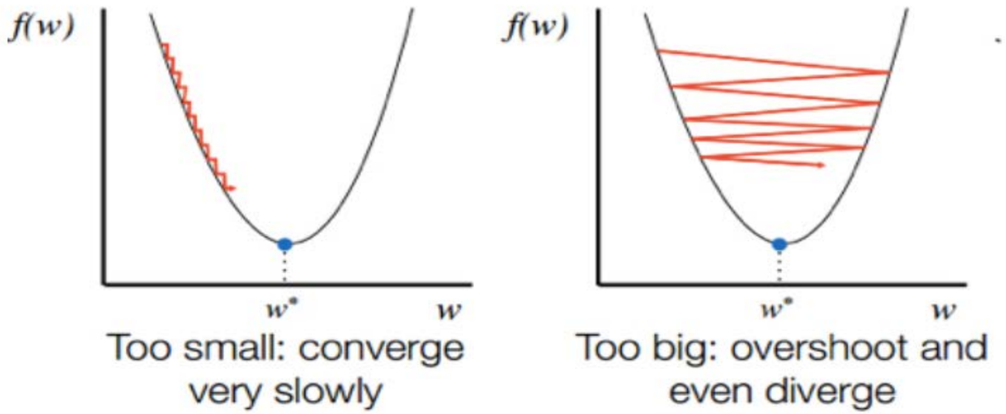


Image Source: https://srdas.github.io/DLBook/GradientDescentTechniques.html

Difficulties

- Converging to a local minimum can sometimes be quite slow
- No guarantee find global minimum if there are multiple local minima in the error surface

Stochastic Gradient Descent (SGD)

Batch vs Stochastic Gradient Descent

Both: (1) Linear Function; (2) best-fit approximation

setiap epoch update
incremental
lbh cpt konvergennya

Batch

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{d \epsilon D}(y_d - \mathrm{o}_d)^2$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \Delta \boldsymbol{w}$$
$$\Delta \boldsymbol{w} = -\eta \nabla E(\boldsymbol{w})$$
$$\Delta w_k = -\eta \frac{\partial E}{\partial w_k} = \eta \sum_{d \epsilon D} (y_d - \mathrm{o}_d)(\mathrm{x}_{d\mathrm{k}})$$

Batch weight update

Stochastic

$$\mathrm{E}_d(\boldsymbol{w}) = \frac{1}{2}(y_d - \mathrm{o}_d)^2$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \Delta \boldsymbol{w}$$
$$\Delta \boldsymbol{w} = -\eta \nabla E(\boldsymbol{w})$$
$$\Delta w_k = -\eta \frac{\partial E}{\partial w_k} = \eta (y_d - \mathrm{o}_d)(\mathrm{x}_{d\mathrm{k}})$$

Incremental weight update

Stochastic Gradient Descent → default di library

input: Training data D={<x_1,y_1>…<x_n,y_n>}; learning rate η , Error threshold θ

Initialize each w_k to some small random value

Until the termination condition is met, do

- E←0
- For each example <$\mathbf{x}_i$, y_i> do
  - o_i←prediction for $\mathbf{x}_i$ using current w
  - For each linear unit weight w_k, do
    $w_k \leftarrow w_k + \eta(y_i - o_i)x_{ik}$ tidak ada update di luar epoch
  - E←E+(yi-oi)^2/2 #update error

Return **w**

w=[0,0,0,0] #w0=0; w1=0; w2=0, w3=0

epoch 1: E=0

<[1,0,1],-1>:
o1=0
w_0=0+0.3(-1-0)*1=-0.3
w_1=0+0.3(-1-0)*1=-0.3
w_2=0+0.3(-1-0)*0=0
w_3=0+0.3(-1-0)*1=-0.3
E=0+(-1-0)^2/2=0.5
<[0,-1,-1],+1>:
o2=0
w_0=-0.3+0.3(1-0)*1=0
w_1=-0.3+0.3(1-0)*0=-0.3
w_2=0+0.3(1-0)*-1=-0.3
w_3=-0.3+0.3(1-0)*-1=-0.6
E=0.5+(1-0)^2/2=1

epoch 2: E=0

Stochastic Gradient Descent: Example

learning rate	0,3
Error threshold	0,010
Fungsi aktivasi	linear

$0 + 0,3\,(-1) = 0$

	x1	x2	x3	Target	w0	w1	w2	w3	sigma	linear	Err
Iterasi 1	1	0	1	-1	0,00	0,00	0,00	0,00	0,00	0,00	0,5
	0	-1	-1	1	-0,30	-0,30	0,00	-0,30	0,00	0,00	1
	-1	-0,5	-1	1	0,00	-0,30	-0,30	-0,60	1,05	1,05	1
					-0,01	-0,29	-0,29	-0,59			1

	x1	x2	x3	Target	w0	w1	w2	w3	sigma	linear	Err
Iterasi 2	1	0	1	-1	-0,01	-0,29	-0,29	-0,59	-0,89	-0,89	0,01
	0	-1	-1	1	-0,05	-0,32	-0,29	-0,62	0,86	0,86	0,02
	-1	-0,5	-1	1	-0,01	-0,32	-0,33	-0,66	1,14	1,14	0,03
					-0,05	-0,28	-0,31	-0,62			0,03

	x1	x2	x3	Target	w0	w1	w2	w3	sigma	linear	Err
Iterasi 10	1	0	1	-1	-0,09	-0,21	-0,39	-0,69	-0,99	-0,99	0,000
	0	-1	-1	1	-0,09	-0,22	-0,39	-0,69	0,98	0,98	0,000
	-1	-0,5	-1	1	-0,09	-0,22	-0,39	-0,70	1,02	1,02	0,000
					-0,09	-0,21	-0,39	-0,69			0,000

Mini-batch Stochastic Gradient Descent

Epoch
Iterasi untuk semua training data atau semua mini-batch.

Batch size
parameter ukuran mini-batch (hasil split training data).
Jika batch size = 1, proses incremental/true SGD
Jika batch size = |training data|, proses batch GD
Jika 1<batch size<|training data|, proses mini-batch SGD

gradient descent

Step
Pemrosesan satu mini batch diakhiri dengan update bobot

Training Data

Fitur dan Label
mini-batch_1
mini-batch_2
...
mini-batch_n

berhenti di step ke-n,
bukan epoch ke-n

Thank you