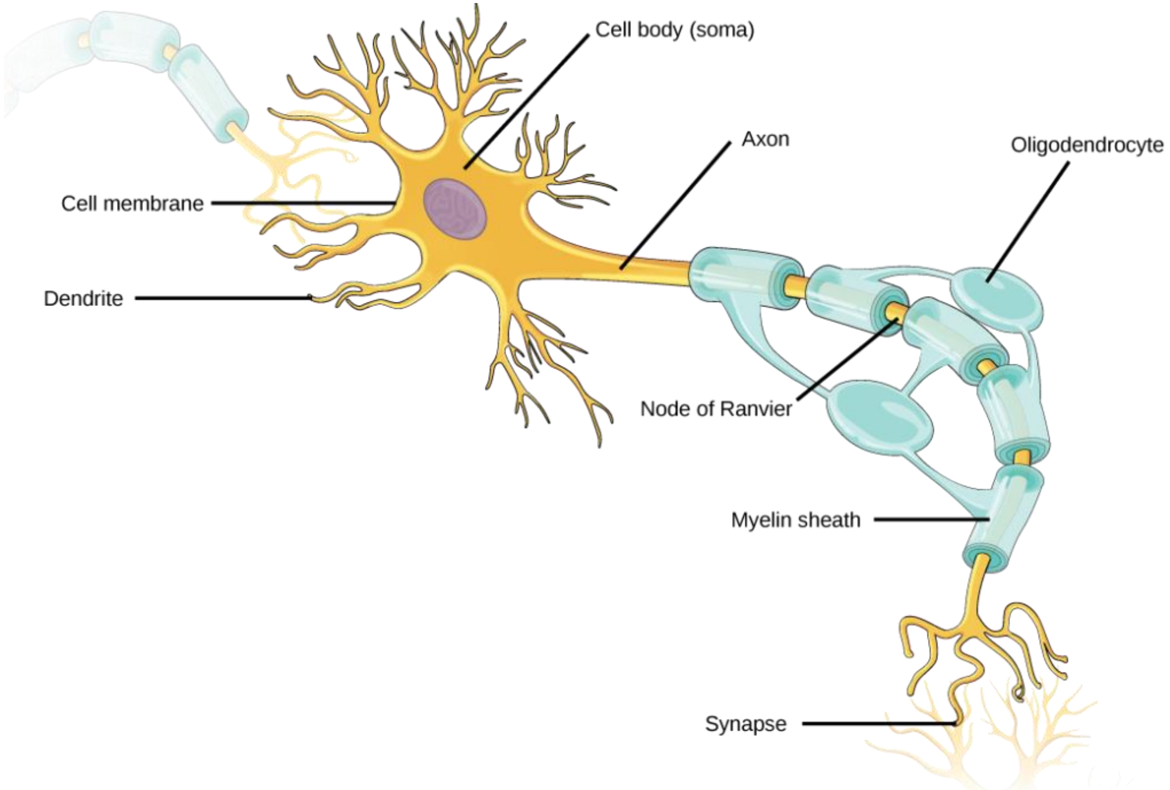
# Artificial Neural Network for Classification (1)

Hanwool Jeong

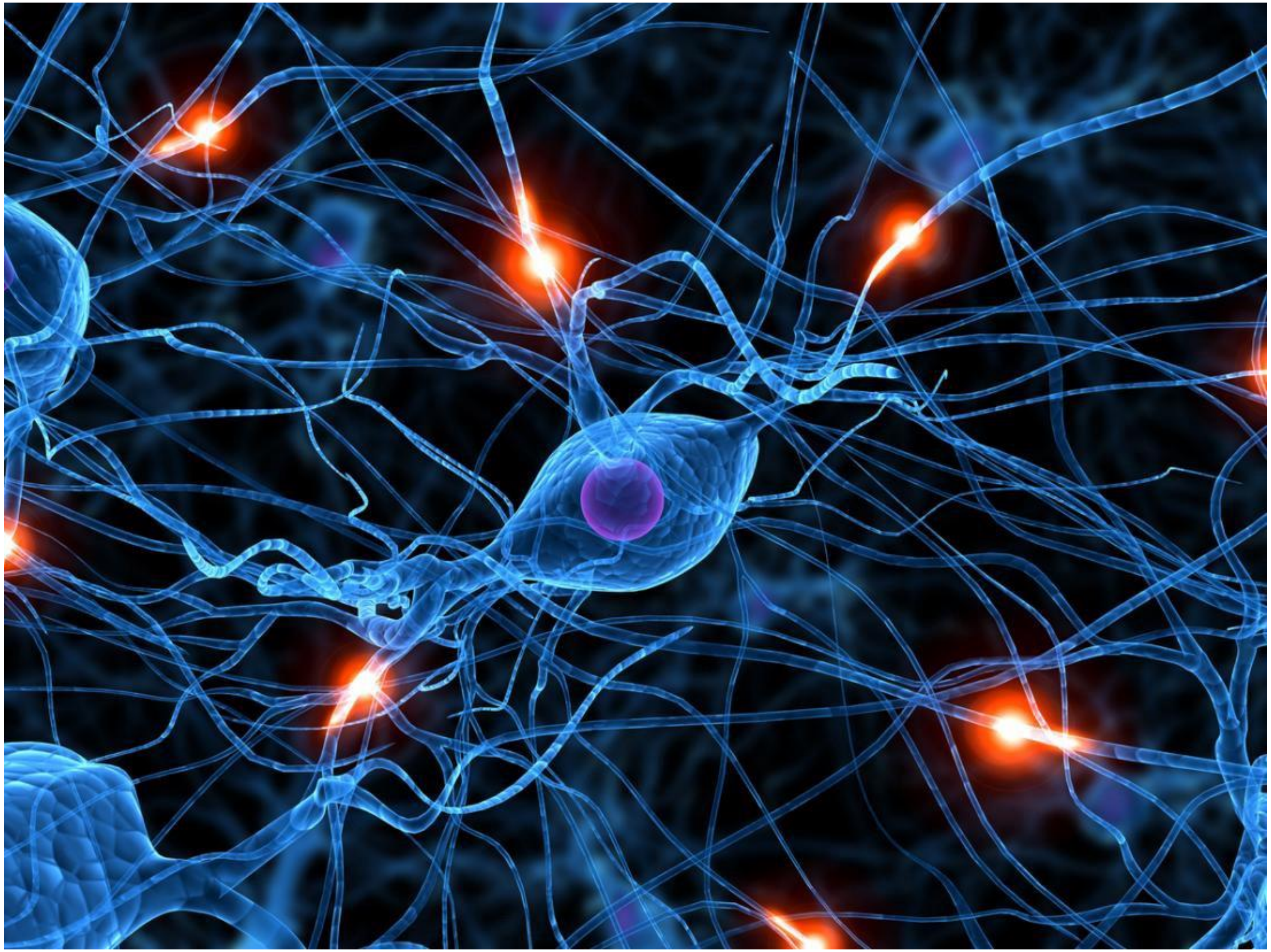
hwjeong@kw.ac.kr

# Neuron in Human Brain

- Considered as Information processing unit
  - Dendrite accepts signals from other neurons
  - Axon conducts electrical impulses away from the cell body.

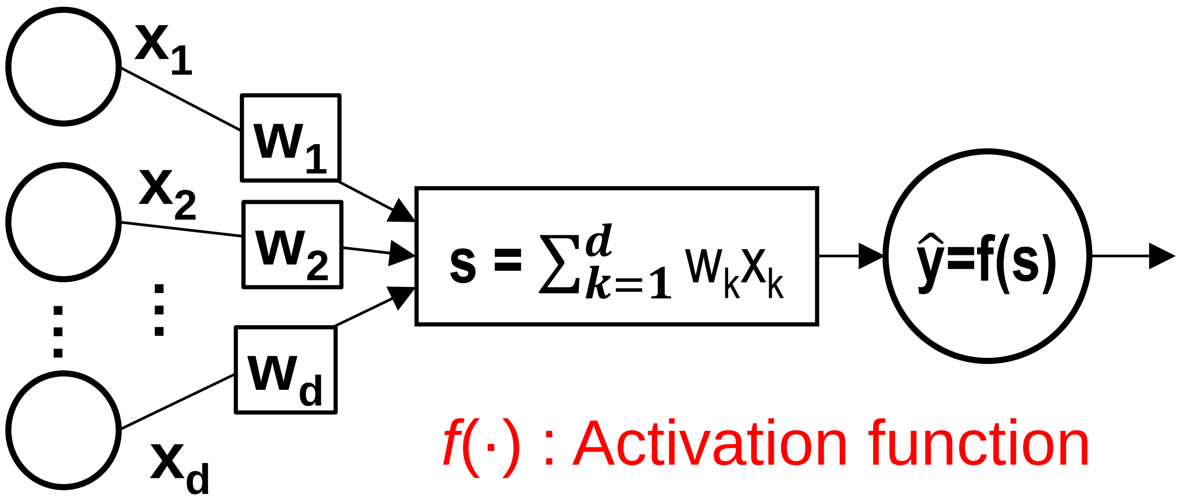


# Neural Network



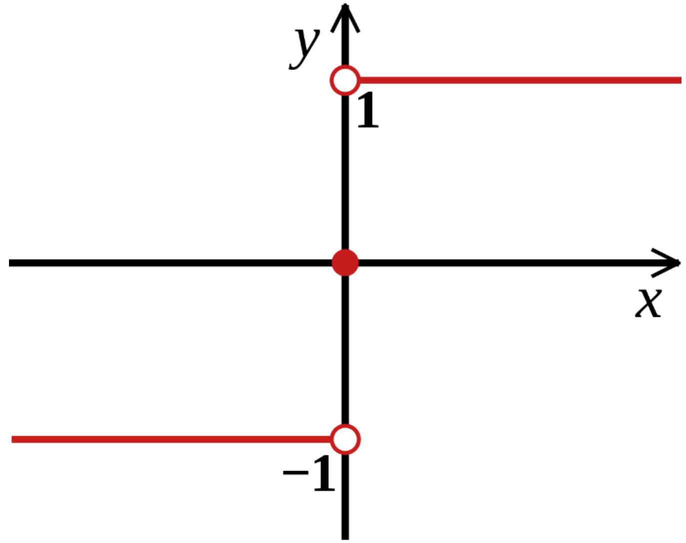
# Perceptron

- Structure and operation of perceptron




$f(\cdot)$ : Activation function

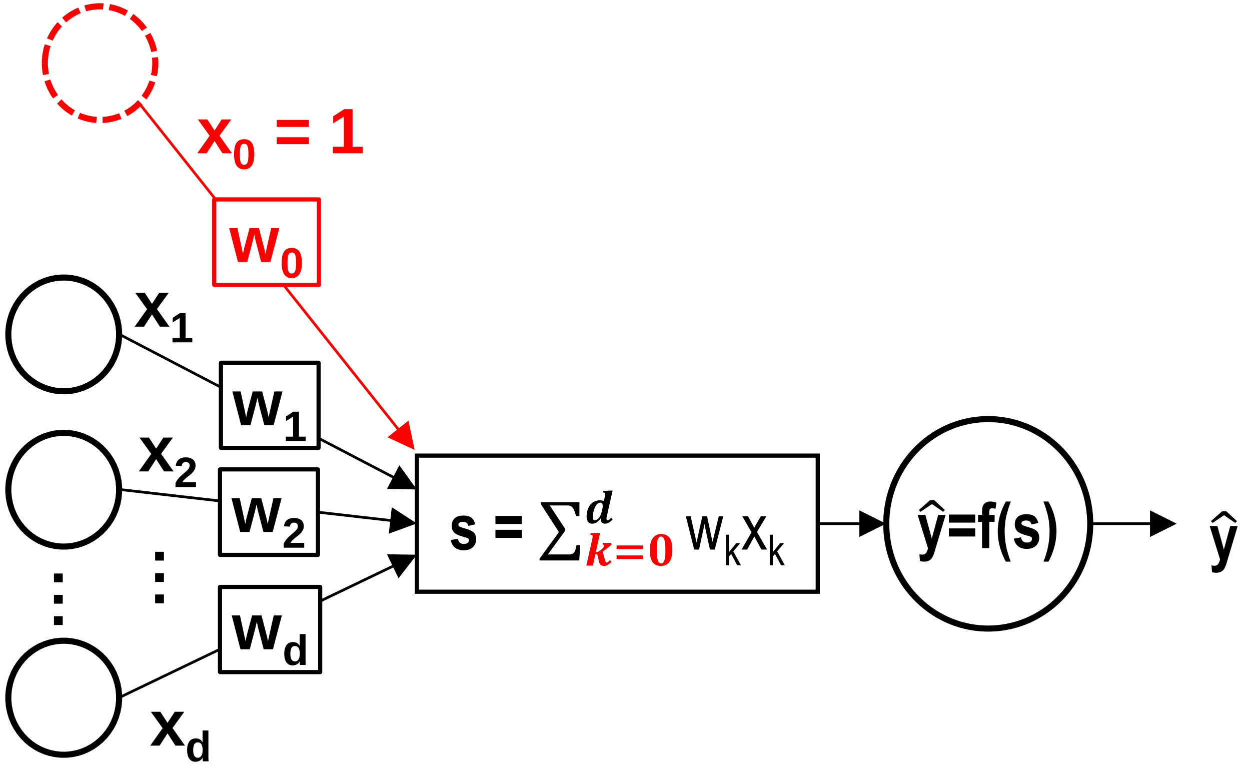
Widely used Activation Function
Sign function or sgn(x)



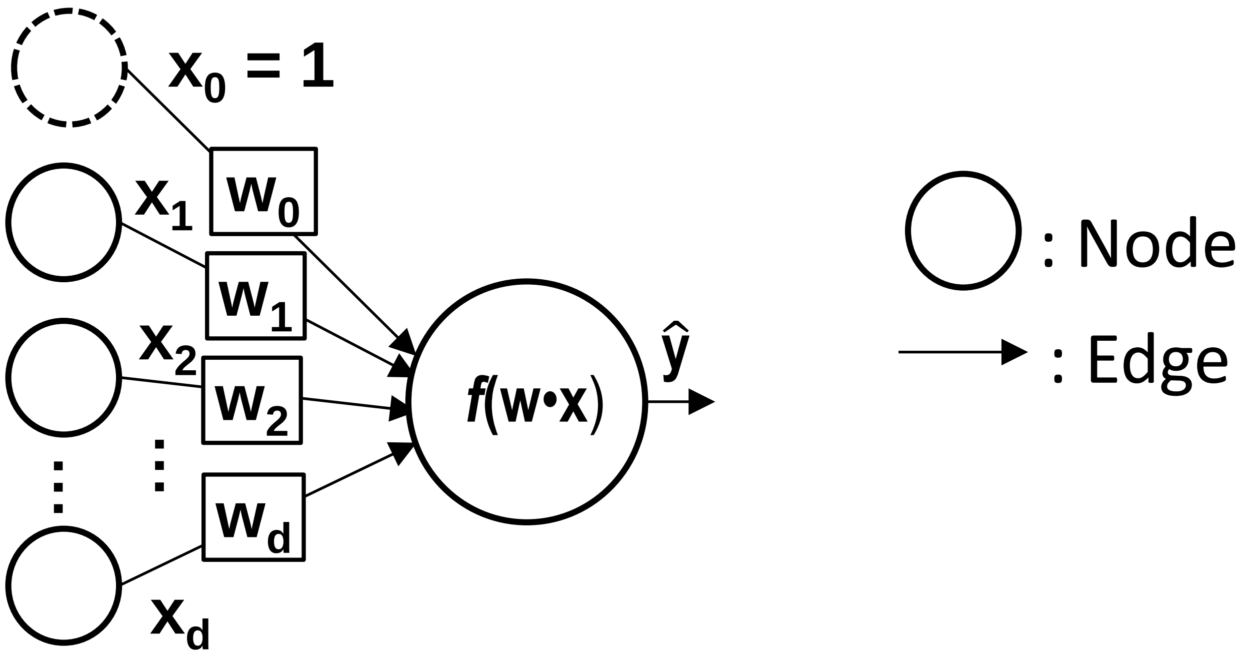

- We can represent $\sum_{k=1}^{d} \mathbf{w}_k \mathbf{x}_k$ with inner product of **w** and **x.**

$$\sum_{k=1}^{d} w_k x_k = \mathbf{w} \bullet \mathbf{x}$$

# Adding Bias

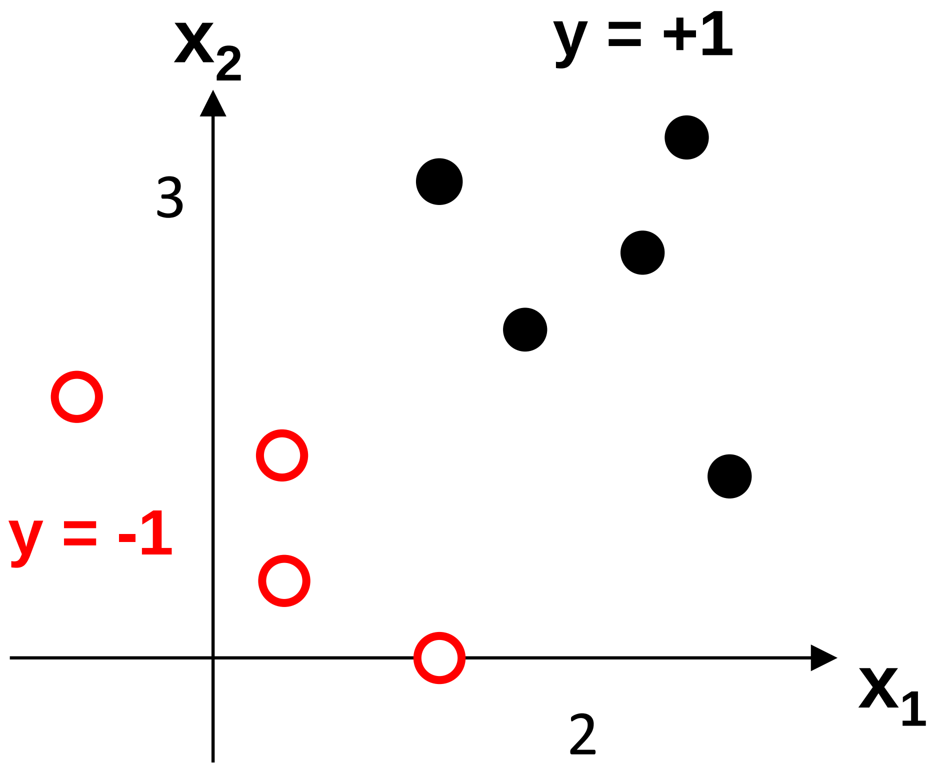


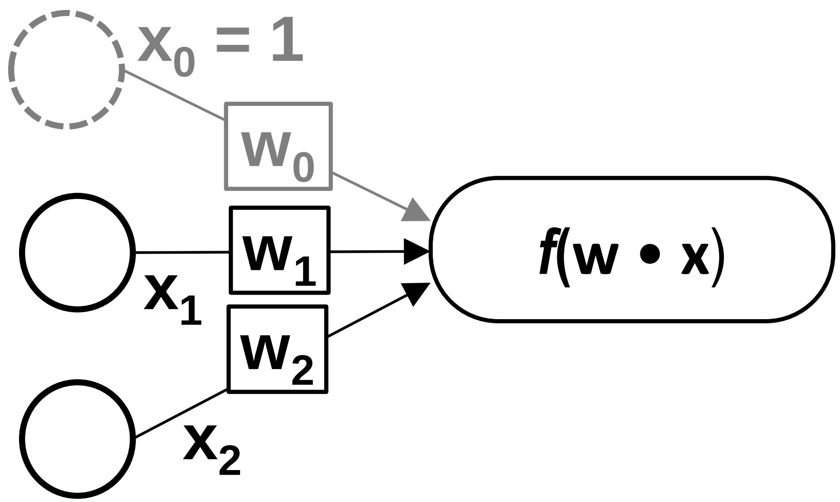
# Structure of Perceptron



# Simplest Example

- Can you apply perceptron to classification shown below?
  ➔ How should you determine $w_0$, $w_1$, $w_2$?
- That is, to determine the parameter set or prediction model.



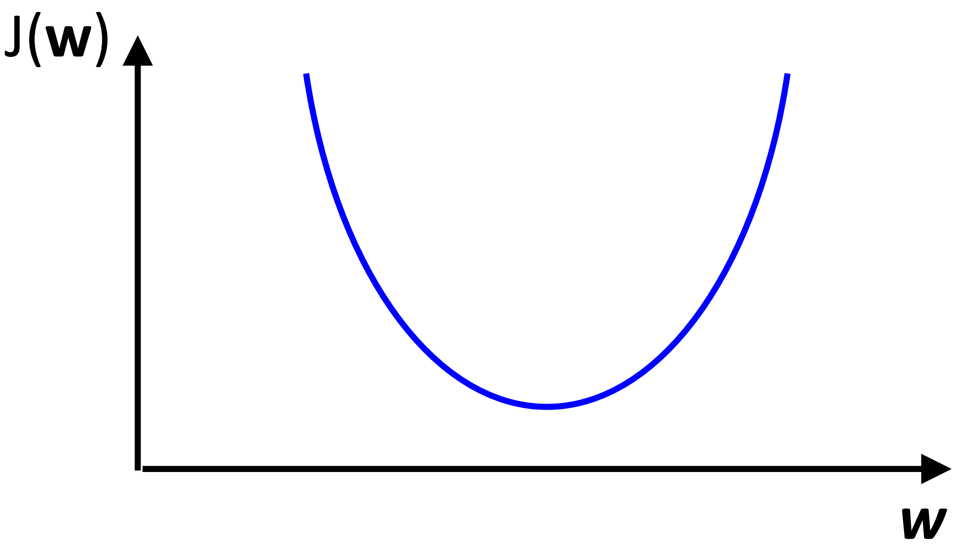
# Perceptron as a Linear Classifier

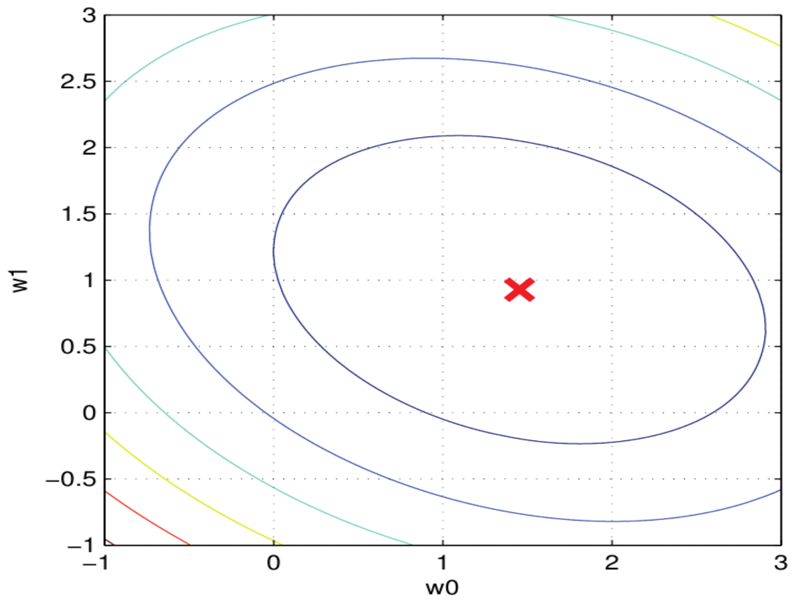
- For 2D feature space, it is line
- How about for 3D feature space? How about larger than 3D?
- We need generalized cost function and minimization process.
  ➔ How do we need to define the cost function?
- Cost function would be dependent to **w** and our objective is to find out the **w*** that minimizes the cost.

# Revisit Gradient Descent

- Given J(**w**), $\frac{\partial J}{\partial \boldsymbol{w}}$ = 0 or $\nabla_w$J = 0 is the solution for minimum/maximum.
- However, it is highly complex or impossible to find solution analytically for many cases.
- Gradient descent method can find the above numerically by repeatedly doing:

$$\boldsymbol{w}_{\text{next}} = \boldsymbol{w}_{\text{present}} - \eta \nabla J(\boldsymbol{w}) \quad \leftarrow \eta : \text{learning rate}$$



# Training for Perceptron

- Cost function is defined as

$$\mathrm{J}(\mathbf{w}) = -\sum_{x_i \in A} \mathbf{y}_\mathrm{i}(\mathbf{w} \bullet \mathbf{x_i})$$

where $A$ is a set of samples that wrongly classified given **w**
➔ Why do we use **w • x** instead of f(**w • x**)?

$$\frac{\partial J}{\partial w_k} = -\frac{\partial}{\partial w_k}\sum_{x_i \in A} \mathbf{y}_\mathrm{i}(\mathbf{w} \bullet \mathbf{x_i}) = -\frac{\partial}{\partial wk}\sum_{x_i \in A} \mathbf{y}_\mathrm{i}\left(\sum_{\boldsymbol{k=0}}^{\boldsymbol{d}} \mathrm{w_k x_{ik}}\right)$$

$$= -\sum_{x_i \in A} \mathbf{y}_\mathrm{i}\mathrm{x_{ik}}$$

- We can find kth component of **w*** by

$$\mathrm{w_{k,next}} = \mathrm{w_{k,present}} + \eta\sum_{x_i \in A} \mathrm{y_i x_{ik}}$$ ← $\eta$ : learning rate

- In vector representation,

$$\mathbf{w}_\mathrm{next} = \mathbf{w}_\mathrm{present} + \eta\sum_{x_i \in A} \mathrm{y_i}\mathbf{x}_i$$

# Training Algorithm

Initialize **w** randomly
while (A ≠ Φ)
    A = Φ
    for i =(1,…,N):
        $\hat{y}_i$ = f(**w** • $\mathbf{x_i}$)
        if($\hat{y}_i$ ≠ $y_i$):
            A ∪ {$\mathbf{x_i}$}
    If(A ≠ Φ):
        $\mathbf{w}_{\text{next}} = \mathbf{w}_{\text{present}} + \eta \sum_{x_i \in A} y_i \mathbf{x}_i$
**w*** = **w**

← While (A ≠ Φ) can be inefficient

Soft decision can be implemented using "No change in A"

# Batch vs. Stochastic Training

- **Batch training** is training on all samples.
- **Stochastic training** is performing training on one randomly selected sample at a time.
- **Mini-batch training** is training on a part of the overall samples
- Batch training makes the algorithm too slow. Thus, stochastic or mini-batch training is used.

# Applying Stochastic Training

Initialize **w** randomly
while (A ≠ Φ)
    A = Φ
    for i =(1,…,N):
        $\hat{y}_i$ = f(**w** • **x**$_i$)
        if($\hat{y}_i$ ≠ $y_i$):
            A ∪ {**x**$_i$}
    If(A ≠ Φ):
        $\mathbf{w}_{next} = \mathbf{w}_{present} + \eta \sum_{x_i \in A} y_i \mathbf{x}_i$
**w*** = **w**

Initialize **w** randomly
while (A ≠ Φ)
    Shuffle **X**
    for i =(1,…,N):
        $\hat{y}_i$ = f(**w** • **x**$_i$)
        if($\hat{y}_i$ ≠ $y_i$):
            $\mathbf{w}_{next} = \mathbf{w}_{present} + \eta \cancel{\sum_{x_i \in A}} y_i \mathbf{x}_i$
**w*** = **w**

Randomly select ONE sample by
1) Shuffle 2) then pick first error sample

# Checkpoints

- ✓ Perceptron is formed mimicking human neural network.
- ✓ Classification for linearly separable data can be done by Linearly boundary (inner product) + Non-linear (sign func.)
- ✓ Gradient descent (batch/stochastic/mini-batch) for training **θ**(=w).
- ✓ Is there any way to apply the perceptron to linearly inseparable data?

**Like XOR data?**