Introduction of Perceptron in Python

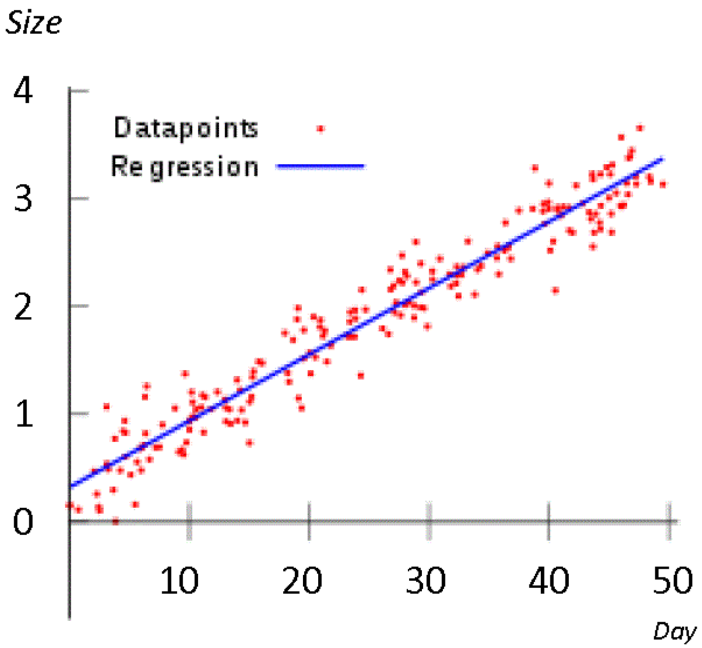
Ko, Youngjoong

Dept. of Computer Engineering,
Dong-A University

Contents

Basic Concepts of Perceptron

❖ Illustration Example (Apple Tree)

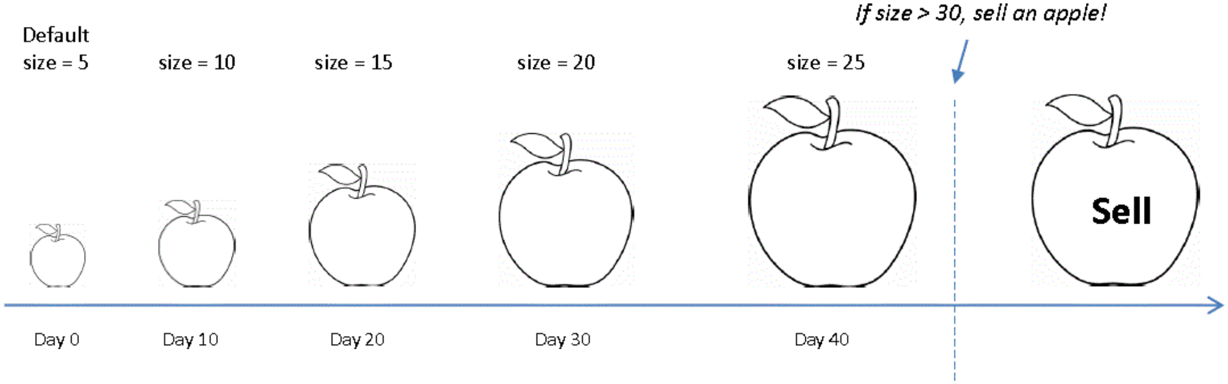


- "어떤 사과나무에 대해서 몇 년에 걸쳐 날짜 별로 사과들의 크기를 측정, 기록"
- 농부는 **특정 크기**가 넘을 때만 시장에 사과를 내다 팔 수 있다고 할 때,

- ***Q*** : *올해 Day -50 에 사과를 내다 팔 수 있을까? 없을까?*

Basic Concepts of Perceptron

❖ Illustration Example (Apple Tree)

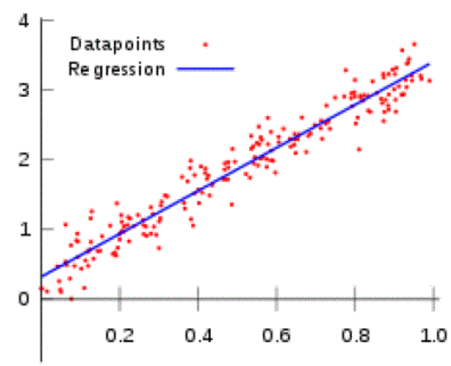


상황 1 : 작년까지 이 사과나무는 위의 경향대로 사과 열매를 맺었다.
조건 : 사과의 크기가 30이 넘으면 팔 수 있다.

Question : 올해 Day-50 에 사과를 팔 수 있을까?

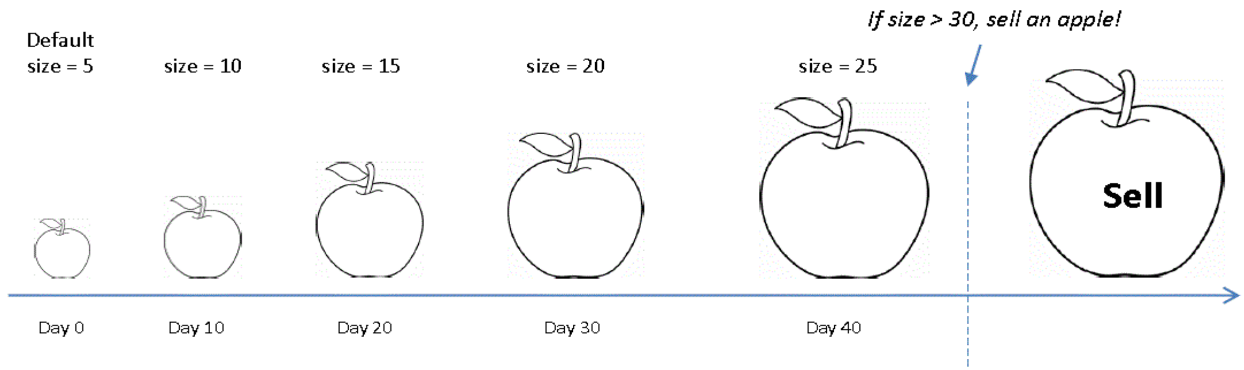
↓

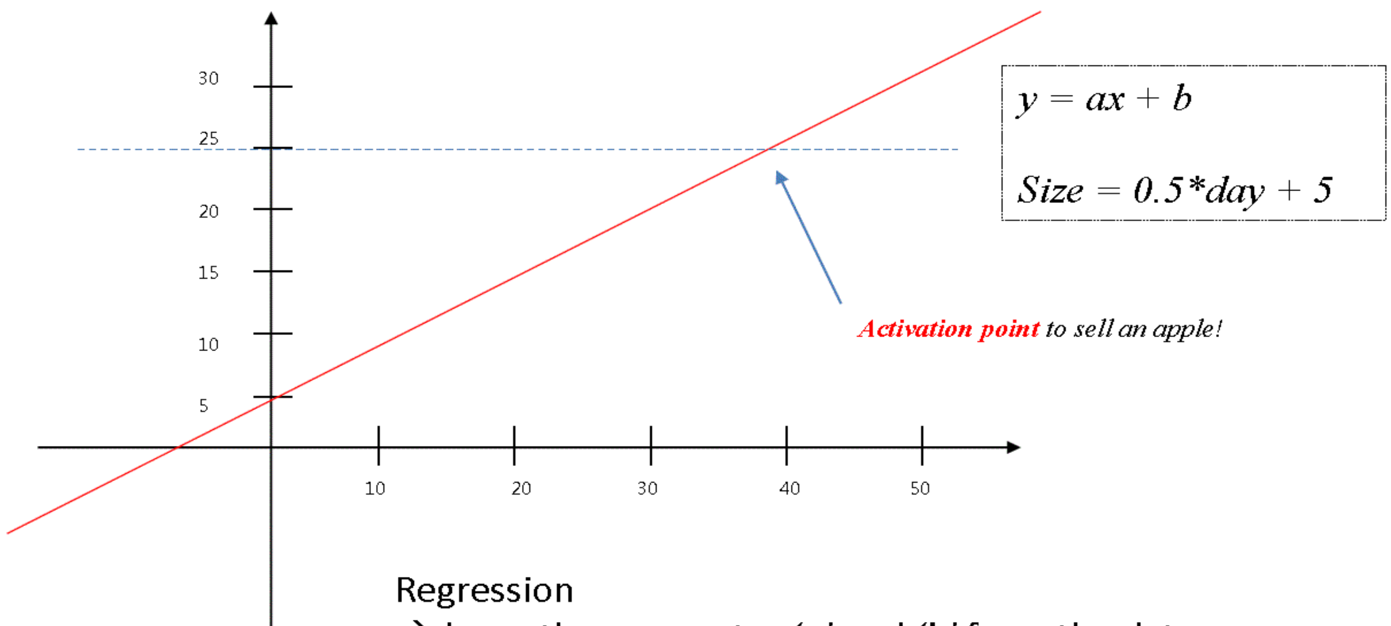
Very Typical ***Regression Problem***



Basic Concepts of Perceptron

❖ Illustration Example (Apple Tree)

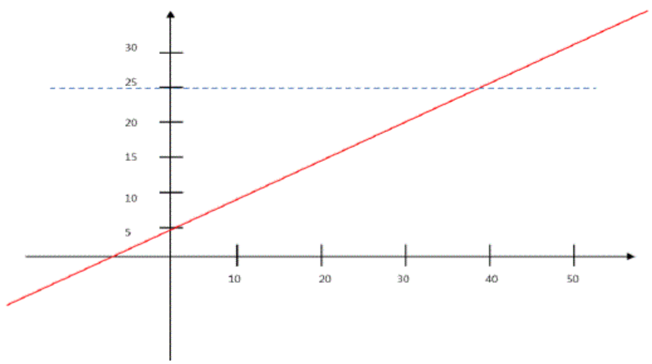


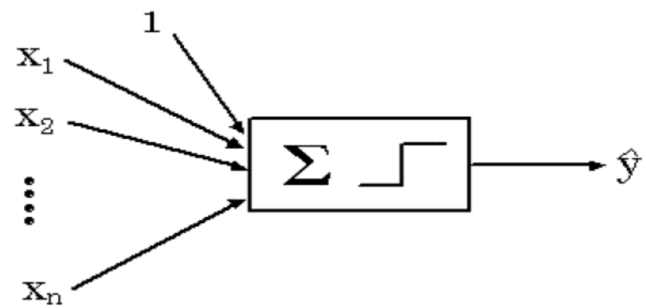


Regression
→ learn the parameter ***'a'*** and ***'b'*** from the data

Basic Concepts of Perceptron

❖ Illustration Example (Apple Tree)





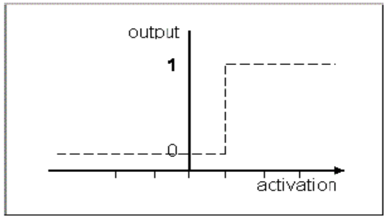
$y = ax + b$ → 입력값을 변형해 새로운 값 계산 (day → size) → $Y = WX + b$

↓

정규화

↓

If y > 30 → sell an apple → 새로운 값을 다시 해석해 최종 결과 산출 (size → 팔까(1)/말까(0)) → *Activation function*



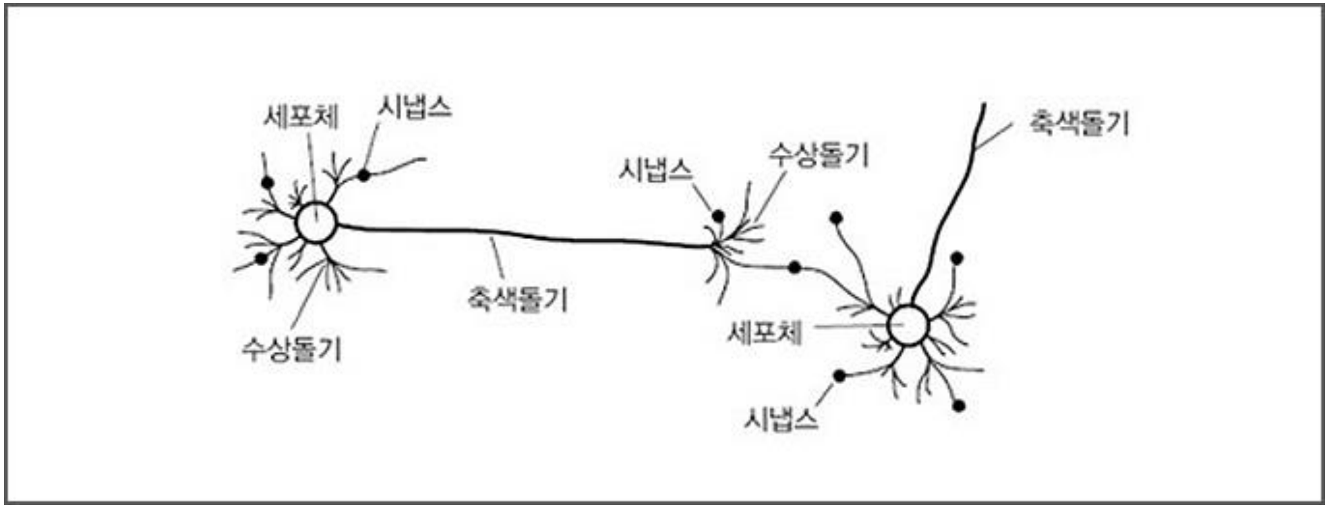
Step Function

Bio-inspired Perceptron

❖ Bio-inspired Learning

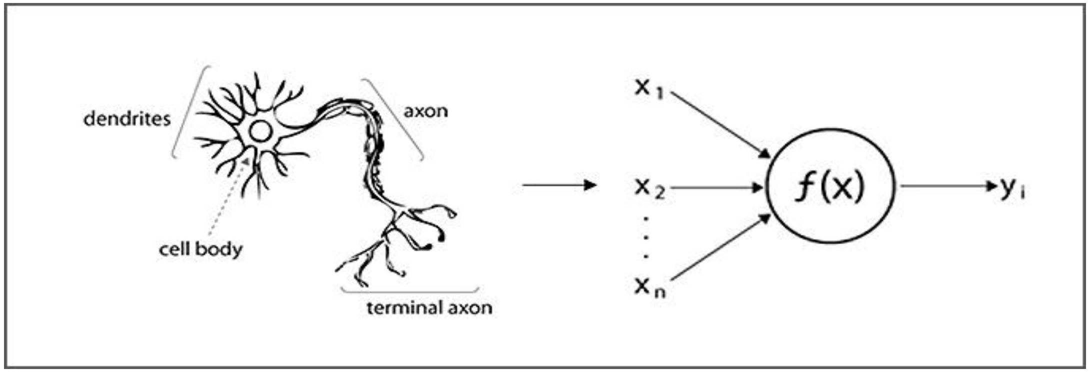
- *Our brains are made up of a bunch of little units, called neurons, that send electrical signals to one another*
  - *The rate of firing tells up how "activated" a neuron is*
  - *The incoming neurons are firing at different rates (i.e., have different activations)*

- The Goal is that we are going to think of our learning algorithm as a single neuron.

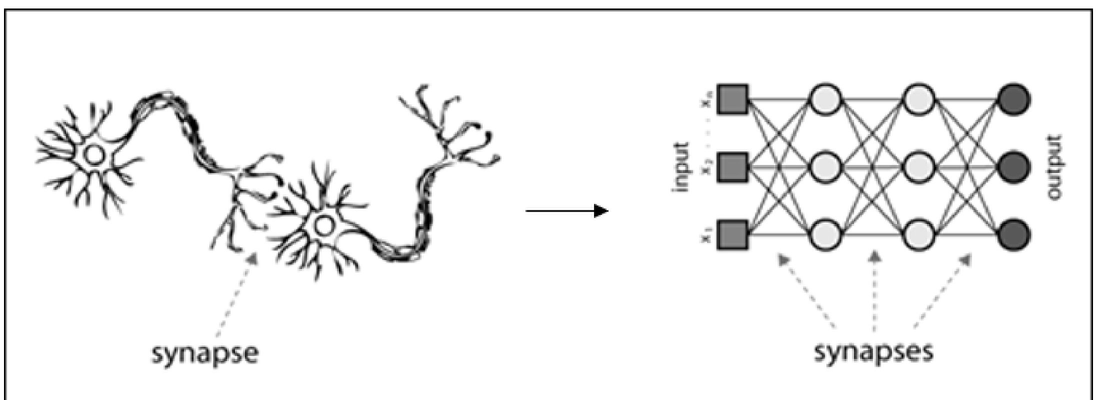


Bio-inspired Perceptron

- **Processing Unit**
  - Neuron *vs.* Node



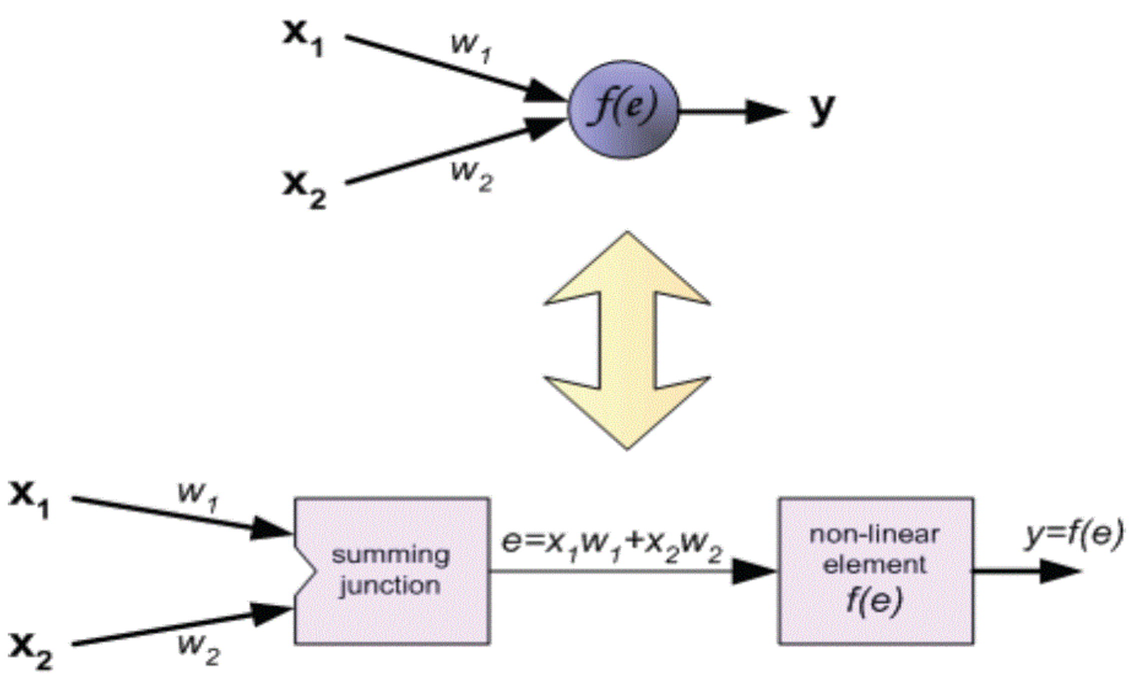
- **Connection**
  - Synapse *vs.* Weight



Structure and Computation

- **Terminology for perceptron**
  - Layer, Node, Weight, Activation function and Learning

- **A simple example**



Structure and Computation

- **The neuron receives input from *D*-many other neurons**
  - One for each input feature
  - The strength of these inputs are the feature values

- **Each incoming connection has a weight and the neuron simply sums up all the weighted inputs**
  - Based on this sum, it decides whether to “fire” or not
  - Firing is interpreted as being a positive example and not Firing is a negative example
    - If the weighted sum is positive, it “fires” and otherwise it doesn’t fire

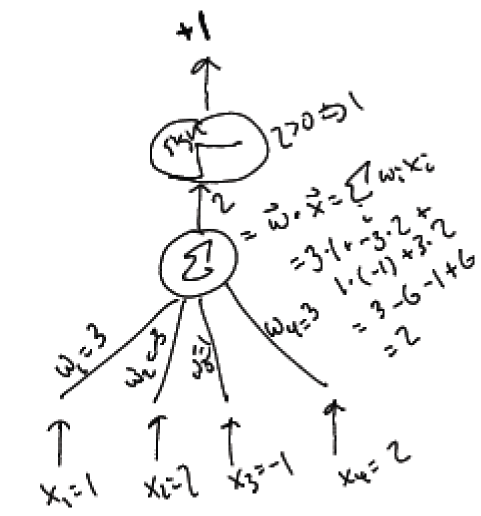
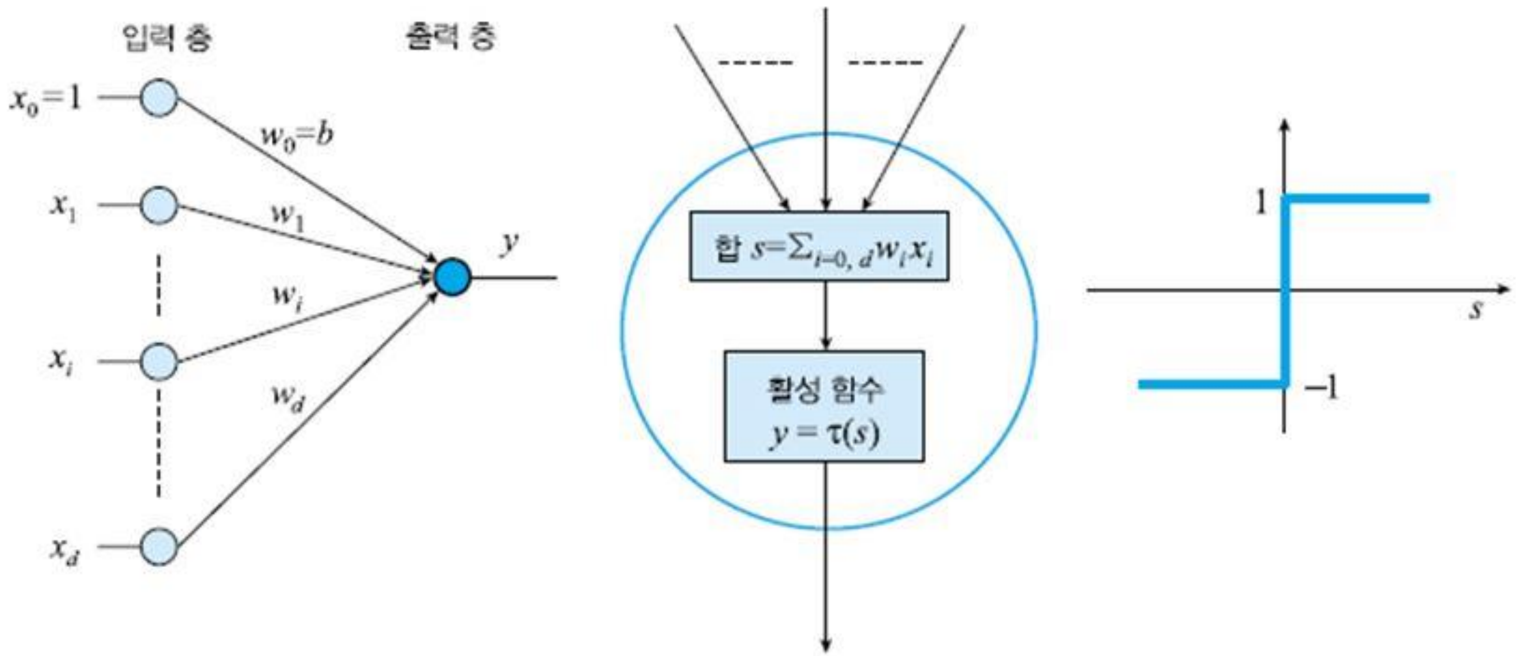


Figure 3.2: figure showing feature vector and weight vector and products and sum

Structure and Computation

❖ Structure of Perceptron

- Input layer: *(d+1)* nodes (feature vector, $\mathbf{x} = (x_1, \ldots, x_d)$)
- Output layer: *1* node (*binary* linear classifier)



(a) 전체 구조 (b) 출력 노드의 연산 (c) 활성 함수

Structure and Computation

- **The weights ($w = (w_0, \ldots, w_d)$) of these neurons are fairly easy to interpret**

  - Suppose that a feature, for instance “is this a System’s class?” gets a zero weight
    - the activation is the same regardless of the value of this feature So features with zero weight are ignored

  - Feature with positive weights are indicative of positive examples
    - Because they cause the activation to increase

  - Feature with negative weights are indicative of negative examples
    - Because they cause the activation to decrease

Structure and Computation

Computation of Perceptron

- Input layer: *Just transfer*
- Output layer: *summation and activation function*

$$\left.\begin{aligned} & y=\tau(s)=\tau(\sum_{i=1}^{d} w_i x_i + b)=\tau(\mathbf{w}^{\mathrm{T}}\mathbf{x}+b) \\ & \text{이때}\quad \tau(s)=\begin{cases} +1, s \geq 0 \\ -1, s < 0 \end{cases} \end{aligned}\right\}$$

- Binary Linear Classifier

$$\left.\begin{aligned} & d(\mathbf{x})=\mathbf{w}^{\mathrm{T}}\mathbf{x}+b>0 \text{ 이면 } \quad \mathbf{x}\in\omega_1 \\ & d(\mathbf{x})=\mathbf{w}^{\mathrm{T}}\mathbf{x}+b<0 \text{ 이면 } \quad \mathbf{x}\in\omega_2 \end{aligned}\right\}$$

Structure and Computation

❖ **Example of Perceptron Computation**

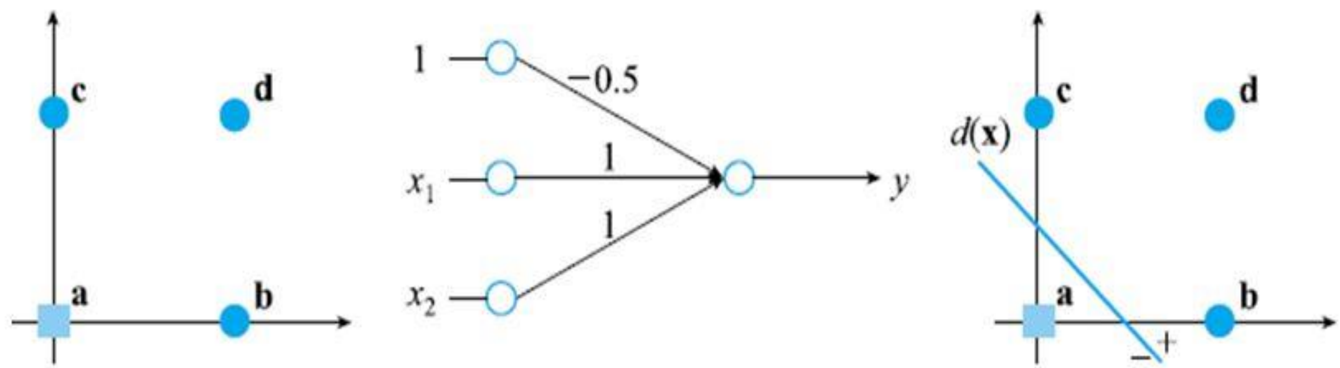
- *OR classification*
- $d(x) = x_1 + x_2 - 0.5$

$\mathbf{a} = (0,0)^T,\ t_a = -1$

$\mathbf{b} = (1,0)^T,\ t_b = 1$

$\mathbf{c} = (0,1)^T,\ t_c = 1$

$\mathbf{d} = (1,1)^T,\ t_d = 1$



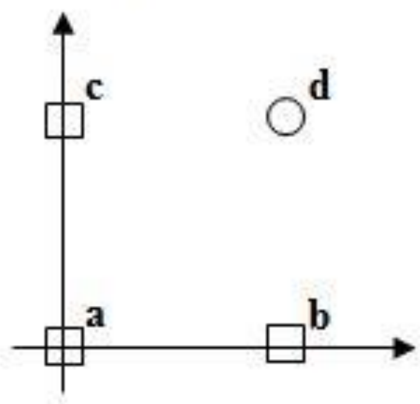
(a) OR 분류 문제 (b) OR 분류기로서 퍼셉트론 (c) 퍼셉트론은 선형 분류기

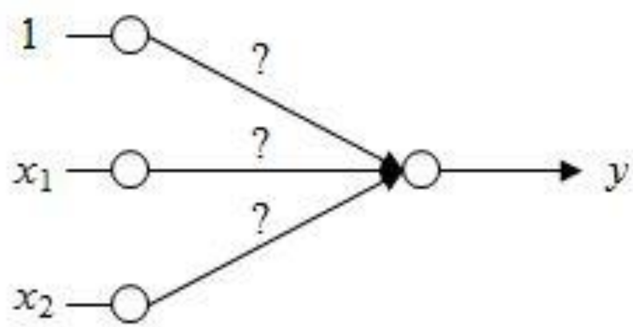
Learning of Perceptron

- **Perceptron Learning**
  - Training set: $\boldsymbol{X} = \{ (x_1, t_1), (x_2, t_2), \ldots (x_N, t_N)\}$, $t_i = 1$ or -1
  - *Try to look for* $\mathbf{w} = (w_0, \ldots, w_d)$ and b
  - ***Ex) And Problem***

$\mathbf{a}=(0,0)^T$ $\mathbf{b}=(1,0)^T$ $\mathbf{c}=(0,1)^T$ $\mathbf{d}=(1,1)^T$

$t_a = -1$ $t_b = -1$ $t_c = -1$ $t_d = 1$





Learning of Perceptron

- **General Designing Steps for Learning in Pattern Recognition**

  - **Step 1:** Building up Classification Model
  - **Step 2**: Cost function, $J(\theta)$
  - **Step 3**: Finding θ to optimize $J(\theta)$

  - **This problem is changed into an Optimization Problem !**

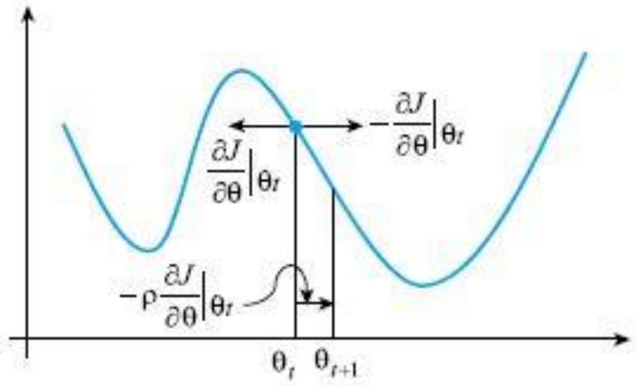
Learning of Perceptron

- **Step 1**
  - Parameter Set: $\theta = \{w, b\}$

- **Step 2**
  - Cost Function: Y is a set of error training examples

$$J(\Theta) = \sum_{\mathbf{x}_k \in Y} (-t_k)(\mathbf{w}^{\mathrm{T}} \mathbf{x}_k + b)$$

- **Step 3**
  - *Gradient Descent Method*
  - *Move* $-\frac{\partial J}{\partial \theta}$ *direction*
  - *Learning Rate:*



Learning of Perceptron

Sketch of algorithm

- Setting up Initial Parameters for $\theta = \{w, b\}$

$$\Theta(h+1) = \Theta(h) - \rho(h)\frac{\partial J(\Theta)}{\partial \Theta}$$

$$\left.\begin{aligned}\frac{\partial J(\Theta)}{\partial \mathbf{w}} &= \sum_{\mathbf{x}_k \in Y}(-t_k)\mathbf{x}_k \\ \frac{\partial J(\Theta)}{\partial b} &= \sum_{\mathbf{x}_k \in Y}(-t_k)\end{aligned}\right\}$$

$$\downarrow$$

$$\left.\begin{aligned}\mathbf{w}(h+1) &= \mathbf{w}(h) + \rho(h)\sum_{\mathbf{x}_k \in Y} t_k \mathbf{x}_k \\ b(h+1) &= b(h) + \rho(h)\sum_{\mathbf{x}_k \in Y} t_k\end{aligned}\right\}$$

또는

$$\hat{\mathbf{w}}(h+1) = \hat{\mathbf{w}}(h) + \rho(h)\sum_{\mathbf{x}_k \in Y} t_k \hat{\mathbf{x}}_k$$

Learning of Perceptron

❖ Perceptron Learning in Batch Mode

입력: 훈련 집합 $X = \{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \cdots, (\mathbf{x}_N, t_N)\}$, 학습률 ρ

출력: 퍼셉트론 가중치 $\mathbf{w}$, b

알고리즘:

1. $\mathbf{w}$와 b를 초기화한다.
2. **repeat** {
3. $\quad Y = \varnothing$;
4. $\quad$ **for** ($i = 1$ **to** N) {
5. $\quad\quad y = \tau(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b)$; // (4.2)로 분류를 수행함
6. $\quad\quad$ **if** ($y \neq t_i$) $Y = Y \cup \mathbf{x}_i$; // 오분류된 샘플 수집
7. $\quad$ }
8. $\quad \mathbf{w} = \mathbf{w} + \rho \sum_{\mathbf{x}_k \in Y} t_k \mathbf{x}_k$; // (4.7)로 가중치 갱신
9. $\quad b = b + \rho \sum_{\mathbf{x}_k \in Y} t_k$;
10. } **until** ($Y = \varnothing$);
11. $\mathbf{w}$와 b를 저장한다.

Learning of Perceptron

- **Perceptron Learning in Pattern Mode**

Algorithm 5 PERCEPTRONTRAIN($\mathbf{D}$, $MaxIter$)

$w_d \leftarrow 0$, for all $d = 1 \dots D$ // initialize weights
$b \leftarrow 0$ // initialize bias
for $iter = 1 \dots MaxIter$ **do**
  for all $(x,y) \in \mathbf{D}$ **do**
    $a \leftarrow \sum_{d=1}^{D} w_d \, x_d + b$ // compute activation for this example
    if $ya \leq 0$ **then**
      $w_d \leftarrow w_d + yx_d$, for all $d = 1 \dots D$ // update weights
      $b \leftarrow b + y$ // update bias
    end if
  end for
end for
return $w_0, w_1, \dots, w_D, b$

Algorithm 6 PERCEPTRONTEST($w_0, w_1, \dots, w_D, b, \hat{x}$)

$a \leftarrow \sum_{d=1}^{D} w_d \, \hat{x}_d + b$ // compute activation for the test example
return SIGN(a)

❖ An Example

$\mathbf{w}(0)=(-0.5,0.75)^T$, $b(0)=0.375$

① $d(\mathbf{x})=-0.5x_1+0.75x_2+0.375$

$Y=\{\mathbf{a},\mathbf{b}\}$

$$\mathbf{w}(1)=\mathbf{w}(0)+0.4(t_a\cdot\mathbf{a}+t_b\cdot\mathbf{b})=\begin{pmatrix}-0.5\\0.75\end{pmatrix}+0.4\left[-\begin{pmatrix}0\\0\end{pmatrix}+\begin{pmatrix}1\\0\end{pmatrix}\right]=\begin{pmatrix}-0.1\\0.75\end{pmatrix}$$

$$b(1)=b(0)+0.4(t_a+t_b)=0.375+0.4*0=0.375$$

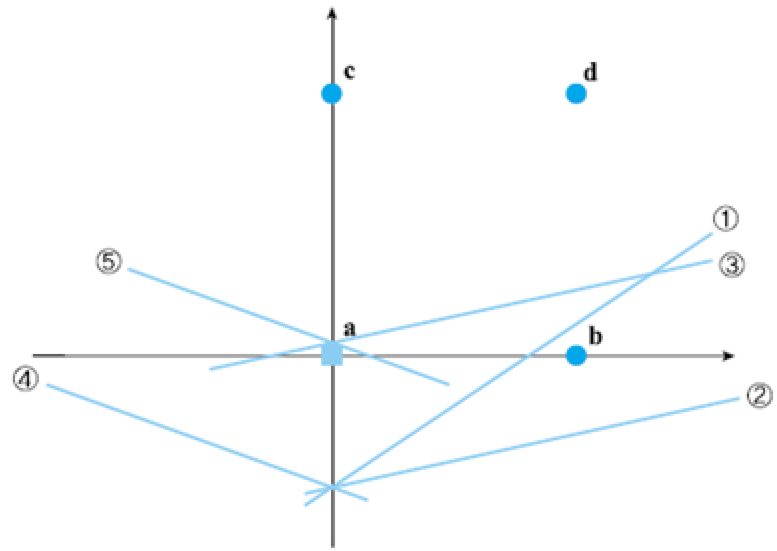
② $d(\mathbf{x})=-0.1x_1+0.75x_2+0.375$

$Y=\{\mathbf{a}\}$

$$\mathbf{w}(2)=\mathbf{w}(1)+0.4(t_a\mathbf{a})=\begin{pmatrix}-0.1\\0.75\end{pmatrix}+0.4\left[-\begin{pmatrix}0\\0\end{pmatrix}\right]=\begin{pmatrix}-0.1\\0.75\end{pmatrix}$$

$$b(2)=b(1)+0.4(t_a)=0.375-0.4=-0.025$$

⋮



Learning of Perceptron

- **Why this particular update achieves better job**
  - Some current set of parameters ***w, b***
  - An example (x_i, t_i), suppose this is a positive example, so $t_i = 1$
  - compute an activation a, and make an error ($a < 0$)

$$
\begin{aligned}
a' &= \sum_{d=1}^{D} w'_d x_d + b' \\
&= \sum_{d=1}^{D} (w_d + x_d) x_d + (b+1) \\
&= \sum_{d=1}^{D} w_d x_d + b + \sum_{d=1}^{D} x_d x_d + 1 \\
&= a + \sum_{d=1}^{D} x_d^2 + 1 \quad > \quad a
\end{aligned}
$$

$$w_d x_d + x_d x_d$$

Geometric Interpretation

- **What does the decision boundary of a perceptron look like?**
  - The sign of the activation, a, changes from -1 to +1
  - The set of points ***x*** achieves zero activation
    - The points are not clearly positive nor negative

- **Consider the case where there is no "bias" term**
  - The decision boundary ***B*** is :

$$\mathcal{B} = \left\{ x \ : \ \sum_d w_d x_d = 0 \right\}$$

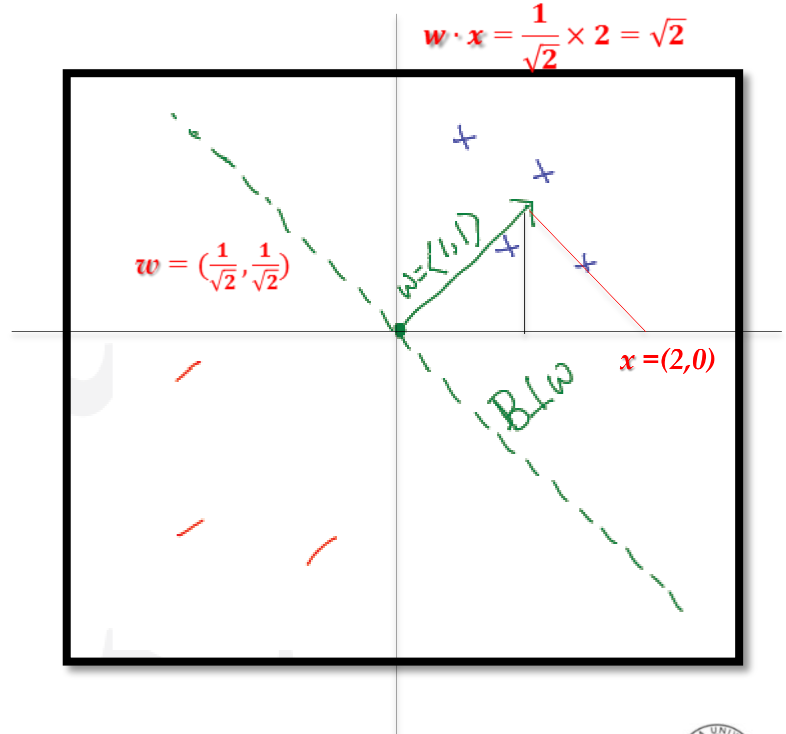
  - If two vectors have a zero dot product, they are perpendicular
  - The decision boundary: the plane perpendicular to ***w***

Geometric Interpretation

- **The scale of the weight vector is irrelevant from the perspective of classification**
  - Work with normalized weight vector ***w***, ||***w***|| = 1

- **The value $\boldsymbol{w} \cdot \boldsymbol{x}$ is just the distance of *x* from the origin when projected onto the vector *w***

- **This distance along *w* is exactly the activation of that example, with no bias**



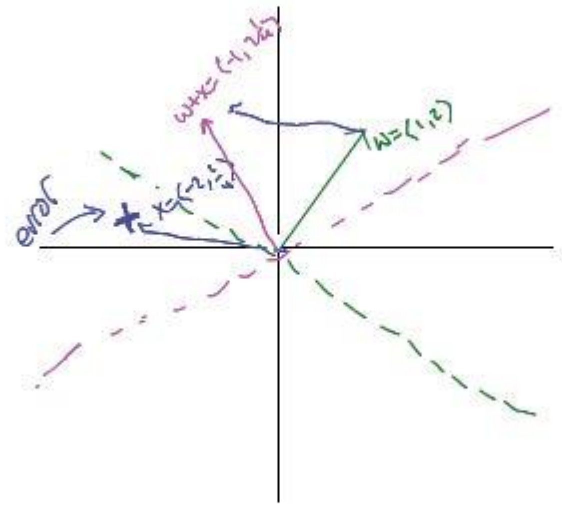
Geometric Interpretation

- **The role of the bias term**
  - Previously, the threshold would be at zero
  - The bias simply moves this threshold
  - Bias term ***b*** is added to get the overall activation
    - The projection plus ***b*** is then compared against **zero**
  - From a geometric perspective, the role of the bias is to shift the decision boundary away from the origin, in the direction of ***w***
  - It is shifted exactly ***b*** units
    - ***b*** is positive, the boundary is shifted away from ***w***
    - ***b*** is negative, the boundary is shifted toward ***w***
  - A positive bias means that more examples should be classified positive
    - By moving the decision boundary in the negative direction, more space yields a positive classification

Geometric Interpretation

- The perceptron update can also be considered geometrically

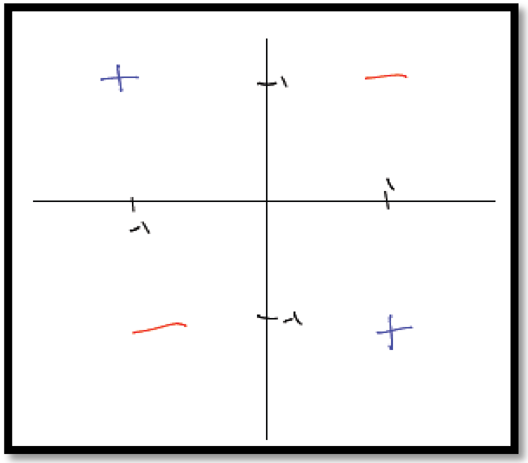
- Here, we have a current guess as to the hyperplane, and positive example comes in that is currently mis-classified



- The weights are updated : ***w*** = ***w*** + ***xt***
  - The weight vector is changed enough so this training example is now correctly classified

Limitations of Perceptron

- **The limitation is that its decision boundaries can only be linear**
  - XOR problem
- **You might ask is: "Do XOR-like problems exist in the real world?"**
  - The answer is "**YES**."



- **Two alternative approaches to taking key ideas from the perceptron and generating classifiers with non linear decision boundaries**
  - Neural Networks: combine multi-layer perceptrons in a single framework
  - Kernels: find computationally efficient ways of doing feature mapping in a computationally and statistically efficient way

Python Code and Practice

- **You should install Python 2.7 and Numpy**
- **Download from: http://nlpmlir.blogspot.kr/2016/01/perceptron.html**
- **Homework**

References

- 오일석. *패턴인식*. 교보문고.

- **Sangkeun Jung.** “Introduction to Deep Learning.” *Natural Language Processing Tutorial*, 2015.

- http://ciml.info/

Thank you for your attention!

http://web.donga.ac.kr/yjko/

고 영 중