Introduction of Multilayer Perceptron in Python

Ko, Youngjoong

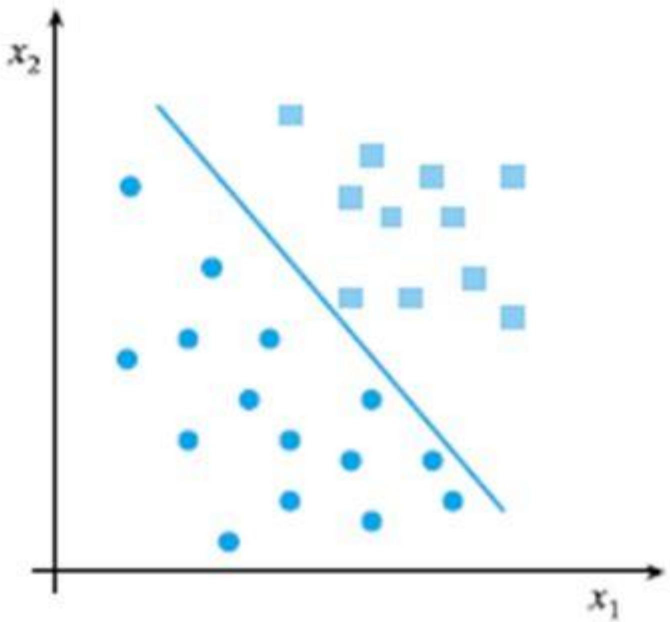
Dept. of Computer Engineering,
Dong-A University

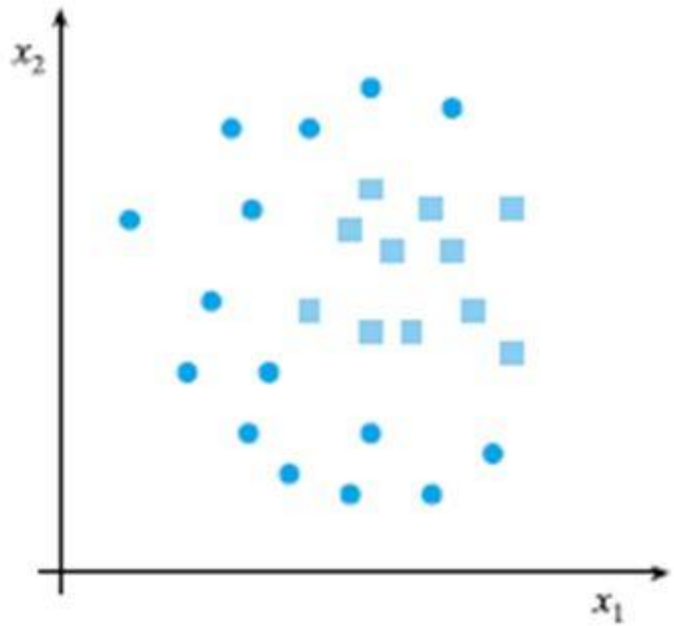
Contents

Non-linear Classification

- **Many Impossible Cases to be classified linearly**
  - Linear model: perceptron
  - Non-linear model: decision tree, nearest neighbor models

  - Explore to find a non-linear learning model from perceptron

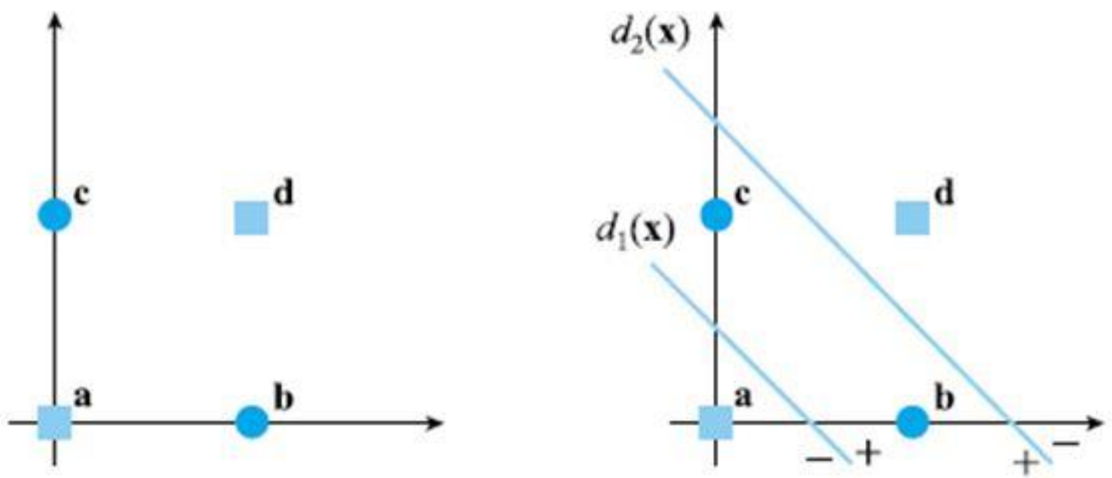




XOR Problem

- **Limitation of performance of perceptrons in XOR problem**
  - 75% Accuracy
  - Overcome this limitation by using two perceptrons

$$\left.\begin{array}{l} \mathbf{w}_1^T\mathbf{x}+b_1 > 0 \text{ 이고 } \mathbf{w}_2^T\mathbf{x}+b_2 > 0 \text{ 이면, } \mathbf{x}\in\omega_1 \\ \mathbf{w}_1^T\mathbf{x}+b_1 < 0 \text{ 이거나 } \mathbf{w}_2^T\mathbf{x}+b_2 < 0 \text{ 이면, } \mathbf{x}\in\omega_2 \end{array}\right\}$$

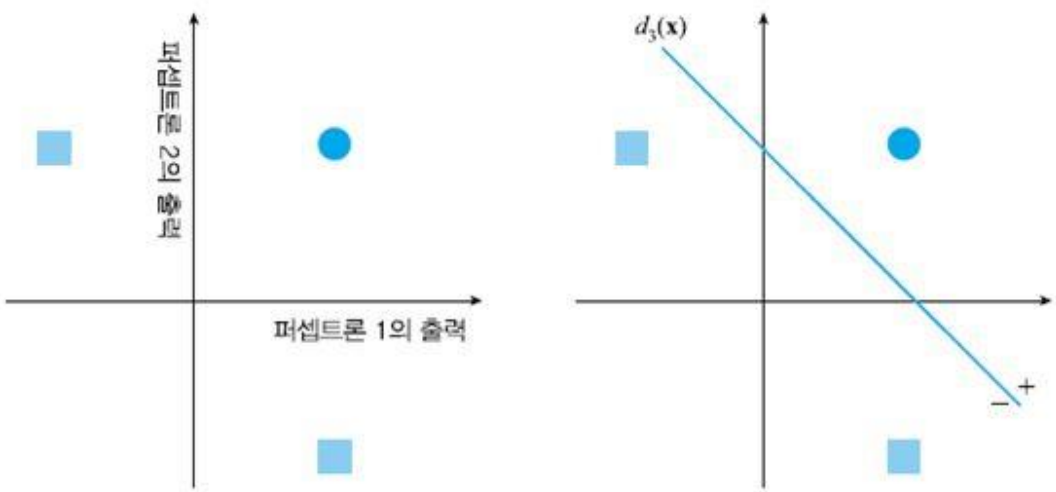


(a) XOR 분류 문제 (b) 두 개의 직선으로 해결

XOR Problem

- **Two Steps for Solution**
  - Mapping an original feature space into a new space
  - Classify in the new space

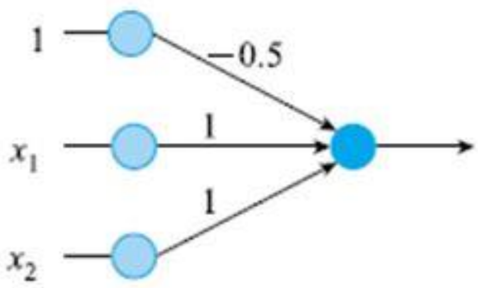
샘플	특징 벡터 (x)		첫 번째 단계		두 번째 단계
	x_1	x_2	퍼셉트론1	퍼셉트론2	퍼셉트론3
a	0	0	−1	+1	−1
b	1	0	+1	+1	+1
c	0	1	+1	+1	+1
d	1	1	+1	−1	−1



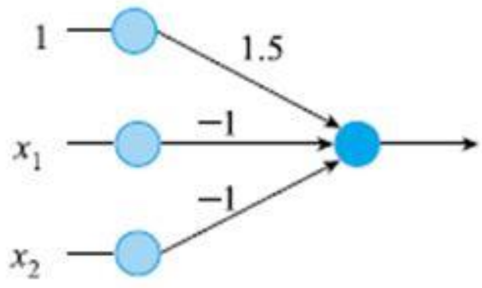
(a) 매핑 공간에서의 샘플 분포 (b) 퍼셉트론3에 의한 영역 분할

XOR Problem

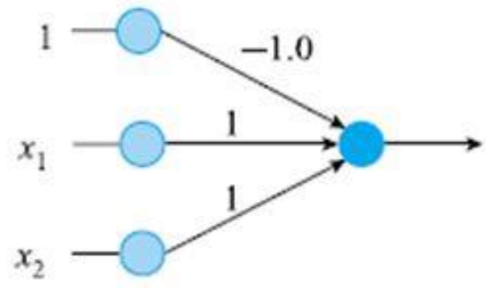
❖ **Example of Multilayer Perceptron as a solution**



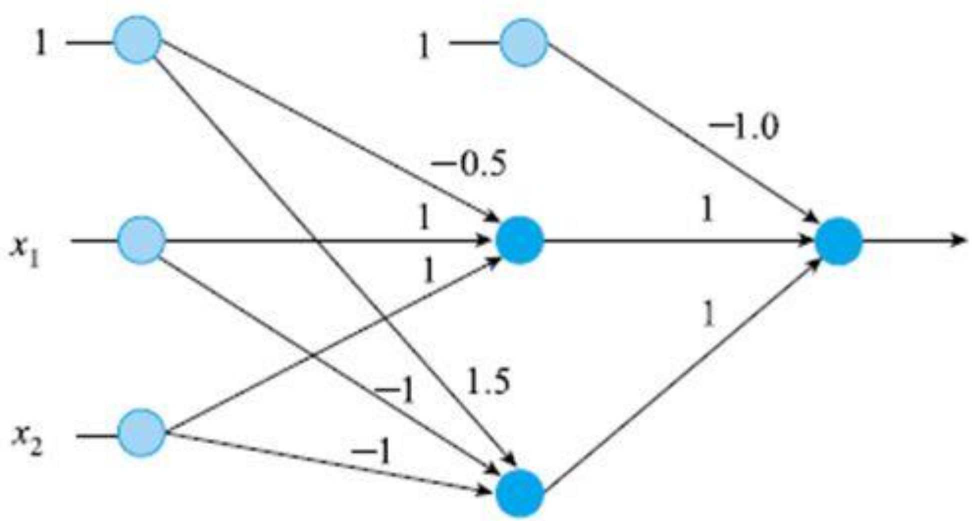
(a) 퍼셉트론1



(b) 퍼셉트론2



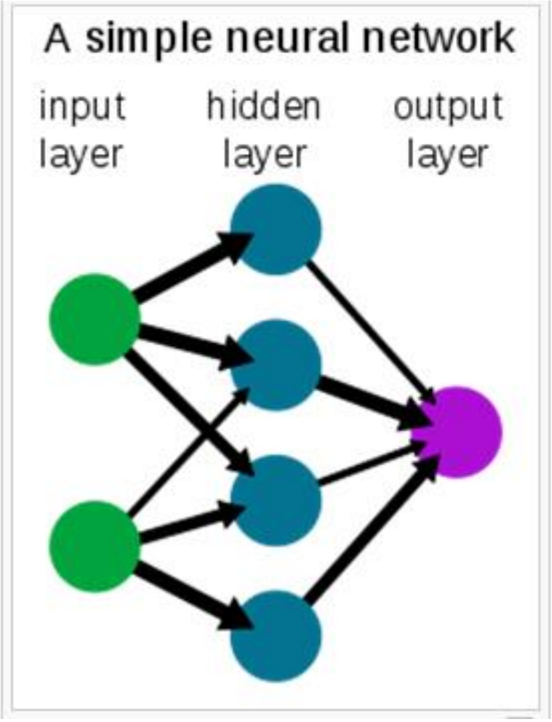
(c) 퍼셉트론3



(d) 다층 퍼셉트론

Architecture of Multilayer Perceptron

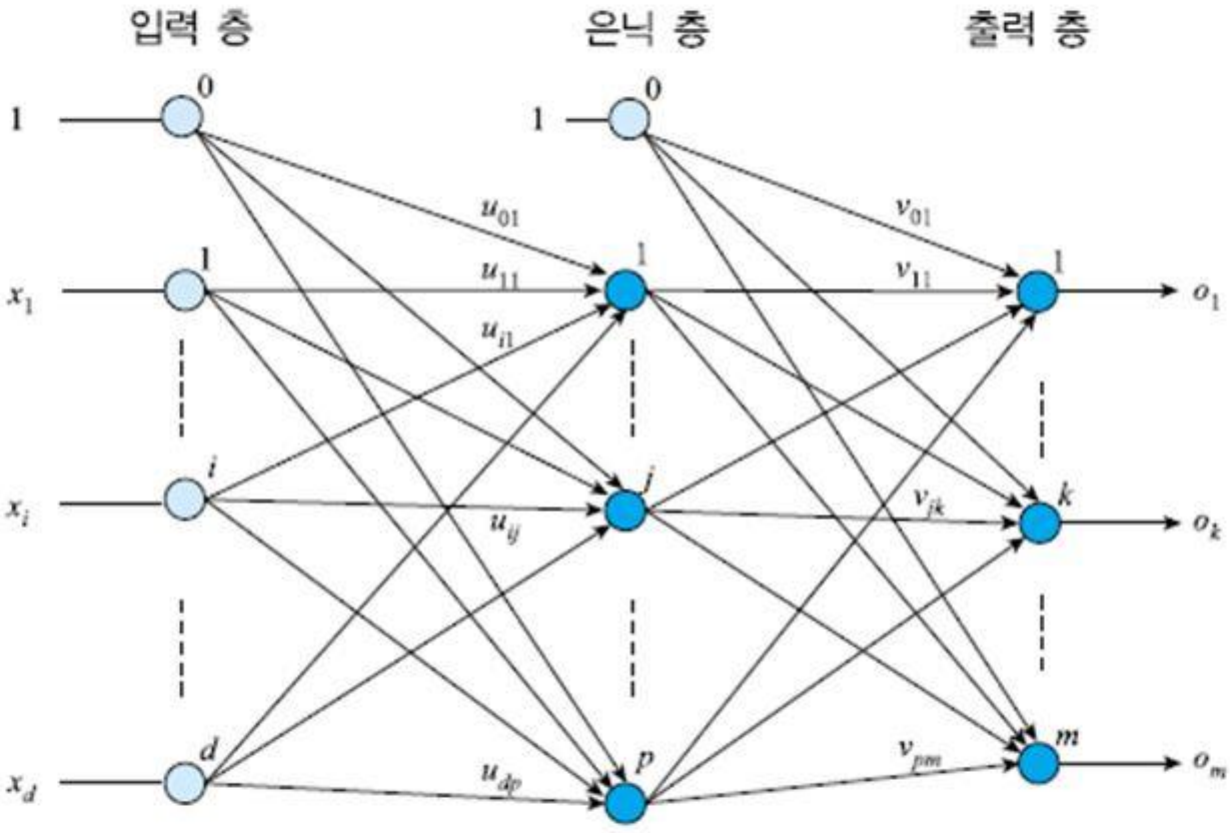
- **Multilayer Perceptron (MLP) in Neural Network**
  - To chain together a collection of perceptrons



  - Two layers (not three layers)
    - Don’t count the inputs as a real layer
    - Two layers of trained weights
  - Each edge corresponds to a different weight
    - Input -> hidden, hidden ->output

Architecture of Multilayer Perceptron

- **Multilayer Perceptron (MLP) in Neural Network**
  - Input layer, Hidden layer and Output layer
  - Weights: ***u*** and ***v***



Forward Computation

❖ **Functions in MLP**

$$\mathbf{o} = f(\mathbf{x})$$

$$\left.\begin{array}{l} \mathbf{z} = p(\mathbf{x}) \\ \mathbf{o} = q(\mathbf{z}) \\ \text{또는} \\ \mathbf{o} = q(p(\mathbf{x})) \end{array}\right\}$$

은닉 층의 j번째 노드, $1 \le j \le p$:

$$\left.\begin{array}{l} z\_sum_j = \sum_{i=1}^{n} x_i u_{ij} + u_{0j} \\ z_j = \tau(z\_sum_j) \end{array}\right\}$$

출력 층의 k번째 노드, $1 \le k \le m$:

$$\left.\begin{array}{l} o\_sum_k = \sum_{j=1}^{p} z_j v_{jk} + v_{0k} \\ o_k = \tau(o\_sum_k) \end{array}\right\}$$

Other Understanding of MLP Forward Propagation

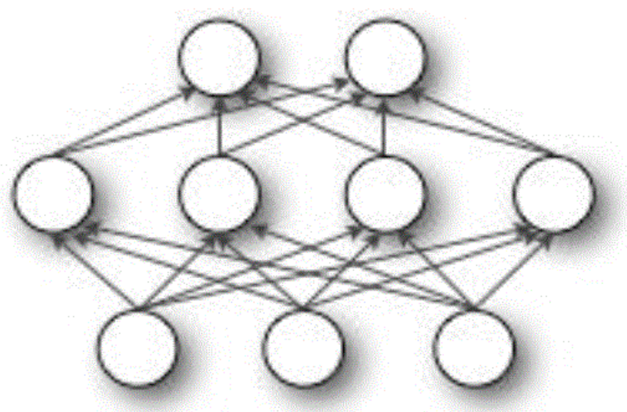
The single-hidden layer Multi-Layer Perceptron (MLP).

An ***MLP*** can be viewed as a ***logistic regressor***, where the input is first transformed using a learnt ***non-linear transformation***

[*Softmax* Function]

G : Scoring Function for top-layer

[*tanh* Function]



output layer $o(x) = G(b^{(2)} + W^{(2)} h(x))$

S : Activation Function for hidden layer

hidden layer $\boxed{h(x)} = \Phi(x) = s(b^{(1)} + W^{(1)} x)$

input layer

$\boldsymbol{x}$

$$f : R^D \rightarrow R^L$$

$$f(x) = G(b^{(2)} + W^{(2)}(s(b^{(1)} + W^{(1)}x))),$$

$\boldsymbol{D}$ is the size of input vector $\boldsymbol{x}$

$\boldsymbol{L}$ is the size of output vector ***f(x)***

Feed Forward Propagation

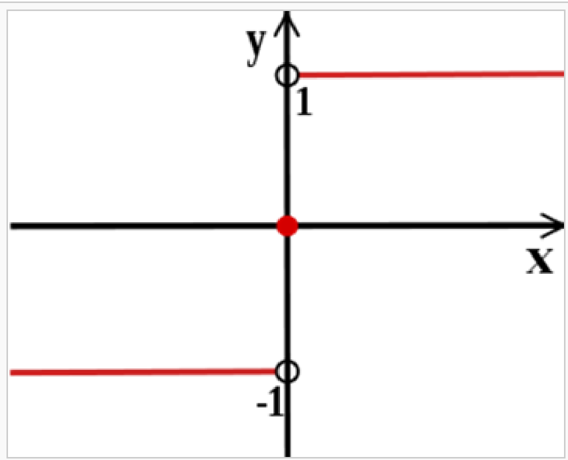
Activation Functions

❖ **Major Difference between MLP and Perceptron**

- Hidden units computes a non-linear computation of their inputs
- Activation function or Link function

$$z_j = f(u_{ij} \cdot x)$$

- One example link function
  - Sign function: Non-differential

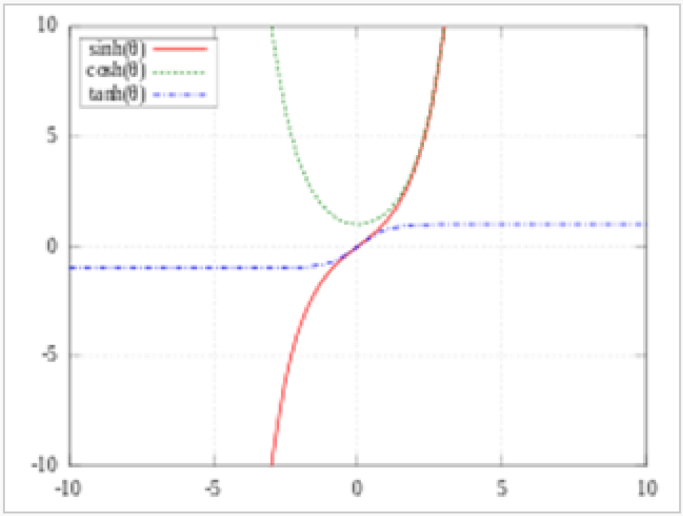


Signum function y = sgn(x)

Activation Functions

- **Hyperbolic tangent function**
  - Popular link function
  - Differential: its derivative is $1-\tanh^2(x)$

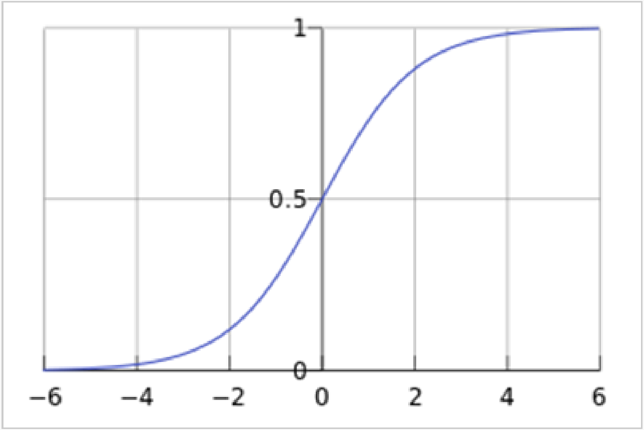
$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x}-1}{e^{2x}+1} = \frac{1-e^{-2x}}{1+e^{-2x}}$$



sinh, cosh and tanh

  - Sigmoid functions

$$S(t) = \frac{1}{1+e^{-t}}$$



The logistic curve

Activation Functions

- **Simple Two-layer MLP**

Algorithm 24 $\textsc{TwoLayerNetworkPredict}(\mathbf{W}, \boldsymbol{v}, \hat{\boldsymbol{x}})$

```
for i = 1 to number of hidden units do
    h_i ← tanh(w_i · x̂)                    // compute activation of hidden unit i
end for
return v · h                               // compute output unit
```

$$\hat{y} = \sum_i v_i \tanh(\boldsymbol{w}_i \cdot \hat{\boldsymbol{x}})$$

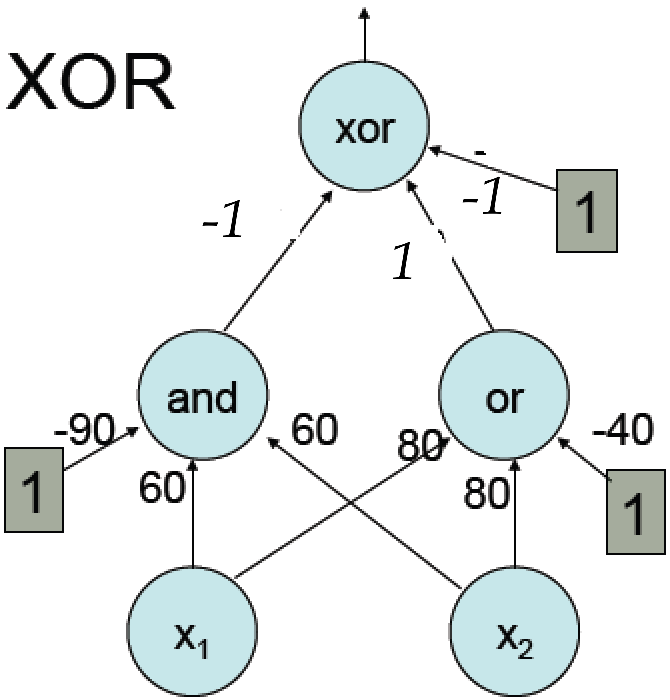
$$= \boldsymbol{v} \cdot \tanh(\mathbf{W}\hat{\boldsymbol{x}})$$

Activation Functions

❖ **Small Two-layer Perceptron to solve the XOR problem**

Net for XOR

x_1, x_2	computation	y
(1,1)	- TANH(30) + TANH(120) -1	-1
(1,0)	- TANH(-30) + TANH(40) -1	1
(0,1)	- TANH(-30) + TANH(40) -1	1
(0,0)	- TANH(-90) + TANH(-40) -1	-1



Inputs x_i are 0/1; f(a) = tanh(a)

Learning in MLP

MLP Learning

- To find {***u***, ***v***}, given $\boldsymbol{X} = \{(x_1, t_1), (x_2, t_2), \ldots, (x_N, t_N)\}$
- x_i : feature vector
- t_i : class label vector or target vector
  - $if\ x_i \in \omega_j, then\ t_i = (0, \ldots, 1, \ldots, 0)$

General Designing Steps for MLP Learning

- **Step 1**: Building up classification model
- **Step 2**: Cost function, $J(\theta)$
- **Step 3**: Design an algorithm for finding θ to optimize $J(\theta)$

Learning in MLP

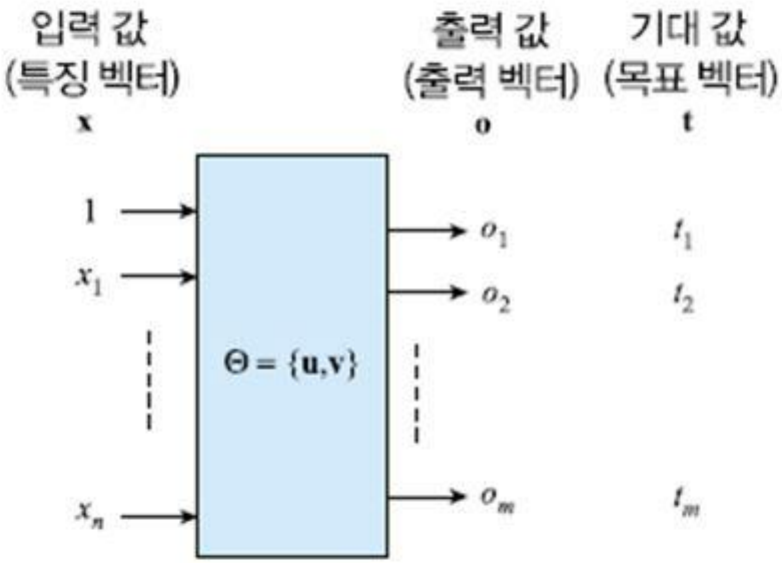
- **Step 1**
  - Parameter set: $\boldsymbol{\theta} = \{\mathbf{u}, \mathbf{v}\}$

- **Step 2**
  - Cost Function:

$$E = \frac{1}{2}\sum_{k=1}^{m}(t_k - o_k)^2$$

  - Overall objective:

$$\min_{u,v}\sum_{k=1}^{m}\frac{1}{2}\left(t_k - \tau_k\left(\sum_j v_{jk}\underbrace{\tau_j(\mathbf{u_j}\cdot\mathbf{x})}_{z_j}\right)\right)^2$$



Learning in MLP

❖ Step 3

- adjust $\boldsymbol{\theta} = \{\mathbf{u}, \mathbf{v}\}$ to reduce errors

$$\left.\begin{aligned}\mathbf{v}(h+1) &= \mathbf{v}(h) + \Delta\mathbf{v} = \mathbf{v}(h) - \rho\frac{\partial E}{\partial \mathbf{v}} \\ \mathbf{u}(h+1) &= \mathbf{u}(h) + \Delta\mathbf{u} = \mathbf{u}(h) - \rho\frac{\partial E}{\partial \mathbf{u}}\end{aligned}\right\}$$

- **How to do line 5**?

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

출력: 가중치 **u**와 **v**

알고리즘:

1. **u**와 **v**를 초기화한다.
2. **repeat** {
3.   **for** (X의 샘플 각각에 대해) {
4.   (4.12)와 (4.13)으로 전방 계산을 한다.
5. $\boxed{\dfrac{\partial E}{\partial \mathbf{v}} \text{와 } \dfrac{\partial E}{\partial \mathbf{u}} \text{를 계산한다.}}$
6.   (4.17)로 새로운 **u**와 **v**를 계산한다.
7.   }
8. } **until** (stop-condition);

Learning in MLP

❖ Back-propagation Algorithm

How to learn the weights??

"Backpropagation Algorithm"

최종 결과물을 얻고	**Feed Forward and Prediction**
그 결과물과 우리가 원하는 결과물과의 차이점을 찾은 후	**Cost Function**
그 차이가 무엇으로 인해 생기는 지	**Differentiation (미분)**
역으로 내려가면서 추정하여	**Back Propagation**
새로운 Parameter 값을 배움	**Weight Update**

Learning in MLP

- **Back-propagation Algorithm**
  - From v_{jk} perspective, it is just a linear model
  - Δv_{jk} calculation

$$\frac{\partial E}{\partial v_{jk}} = \frac{\partial(0.5\sum_{r=1}^{m}(t_r - o_r)^2)}{\partial v_{jk}}$$

$$= \frac{\partial(0.5(t_k - o_k)^2)}{\partial v_{jk}}$$

$$= -(t_k - o_k)\frac{\partial o_k}{\partial v_{jk}}$$

$$= -(t_k - o_k)\frac{\partial \tau(o\_sum_k)}{\partial v_{jk}}$$

$$= -(t_k - o_k)\tau'(o\_sum_k)\frac{\partial o\_sum_k}{\partial v_{jk}}$$

$$= -(t_k - o_k)\tau'(o\_sum_k)z_j$$

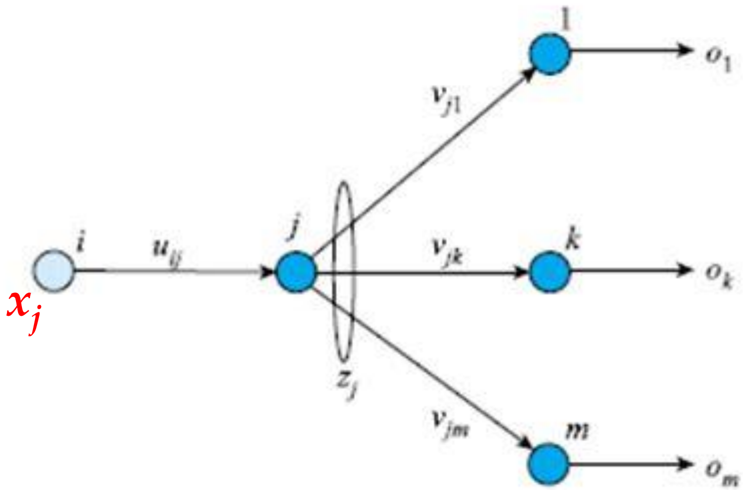
$$\downarrow$$

$$\boxed{\delta_k = (t_k - o_k)\tau'(o\_sum_k), 1 \le k \le m}$$

$$\Delta v_{jk} = -\rho\frac{\partial E}{\partial v_{jk}} = \rho\delta_k z_j, 0 \le j \le p, 1 \le k \le m$$

Learning in MLP

- **Back-propagation Algorithm**
  - Δu_{jk} calculation
  - Gradient descent + Chain rule



$$\frac{\partial E}{\partial u_{ij}} = \frac{\partial(0.5\sum_{k=1}^{m}(t_k - o_k)^2)}{\partial u_{ij}}$$

$$= -\sum_{k=1}^{m}(t_k - o_k)\frac{\partial o_k}{\partial u_{ij}}$$

$$= -\sum_{k=1}^{m}(t_k - o_k)\tau'(o\_sum_k)\frac{\partial o\_sum_k}{\partial u_{ij}}$$

$$= -\sum_{k=1}^{m}(t_k - o_k)\tau'(o\_sum_k)\frac{\partial o\_sum_k}{\partial z_j}\frac{\partial z_j}{\partial u_{ij}}$$

$$= -\sum_{k=1}^{m}(t_k - o_k)\tau'(o\_sum_k)v_{jk}\frac{\partial z_j}{\partial u_{ij}}$$

$$= -\sum_{k=1}^{m}(t_k - o_k)\tau'(o\_sum_k)v_{jk}\tau'(z\_sum_j)x_i$$

$$= -\sum_{k=1}^{m}\delta_k v_{jk}\tau'(z\_sum_j)x_i$$

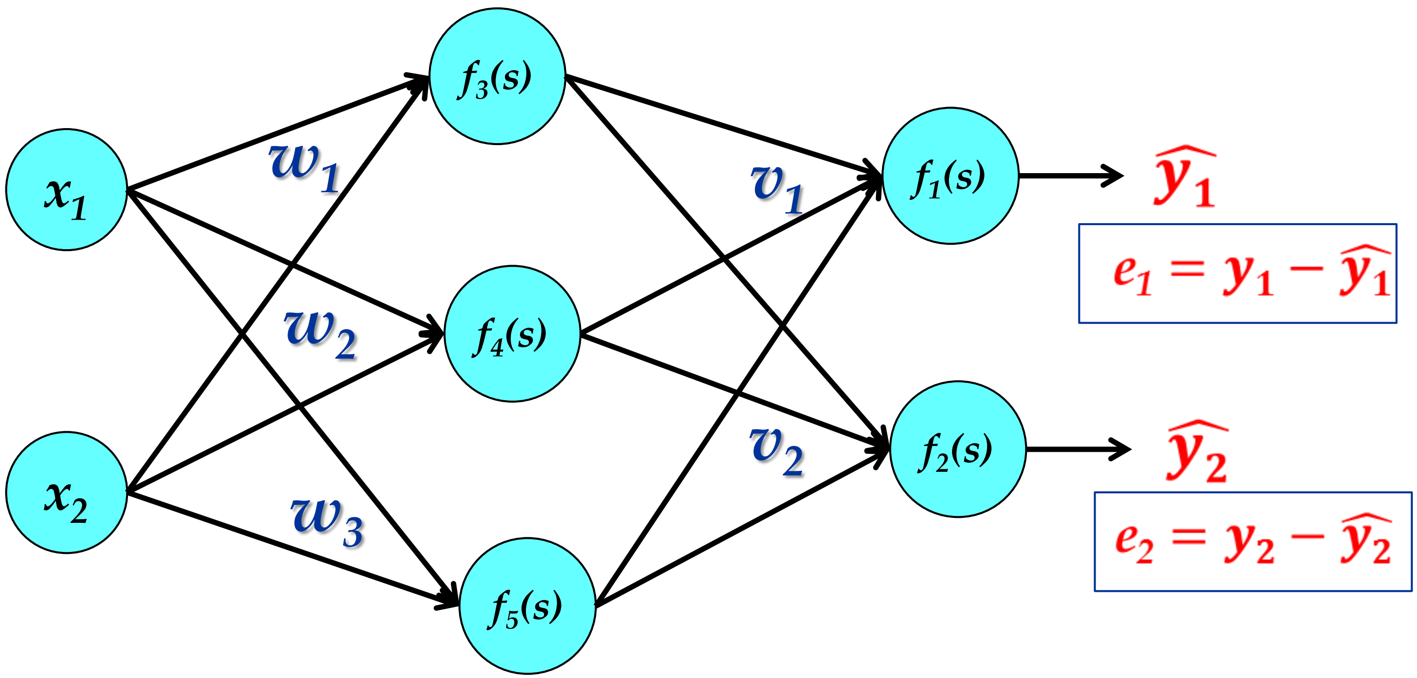
$$\downarrow$$

$$\eta_j = \tau'(z\_sum_j)\boxed{\sum_{k=1}^{m}\delta_k v_{jk}}, 1 \le j \le p$$

$$\Delta u_{ij} = -\rho\frac{\partial E}{\partial u_{ij}} = \rho\eta_j x_i, 0 \le i \le d, 1 \le j \le p$$

Learning in MLP

- **Understanding back-propagation on a simple example**
  - Two layers MLP and No activation function in the output layer



Learning in MLP

- **Understanding back-propagation on a simple example**

$$\mathcal{L}(\mathbf{W}) = \frac{1}{2}\left(y - \sum_i v_i f(\boldsymbol{w}_i \cdot \boldsymbol{x})\right)^2$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_i} = \frac{\partial \mathcal{L}}{\partial f_i}\frac{\partial f_i}{\partial \boldsymbol{w}_i}$$

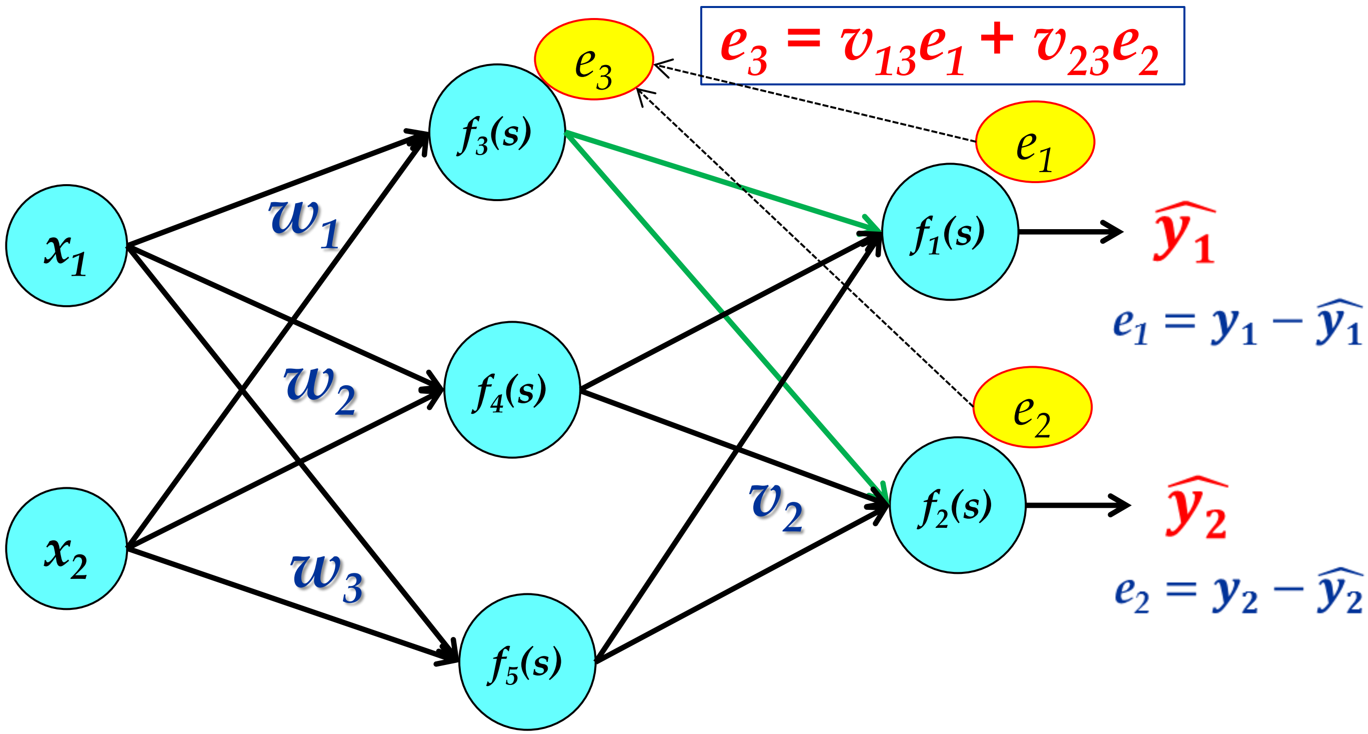
$$\frac{\partial \mathcal{L}}{\partial f_i} = -\left(y - \sum_i v_i f(\boldsymbol{w}_i \cdot \boldsymbol{x})\right) v_i = -e v_i$$

$$\frac{\partial f_i}{\partial \boldsymbol{w}_i} = f'(\boldsymbol{w}_i \cdot \boldsymbol{x})\boldsymbol{x}$$

$$\boxed{\nabla_{\boldsymbol{w}_i} = -e v_i f'(\boldsymbol{w}_i \cdot \boldsymbol{x})\boldsymbol{x}}$$

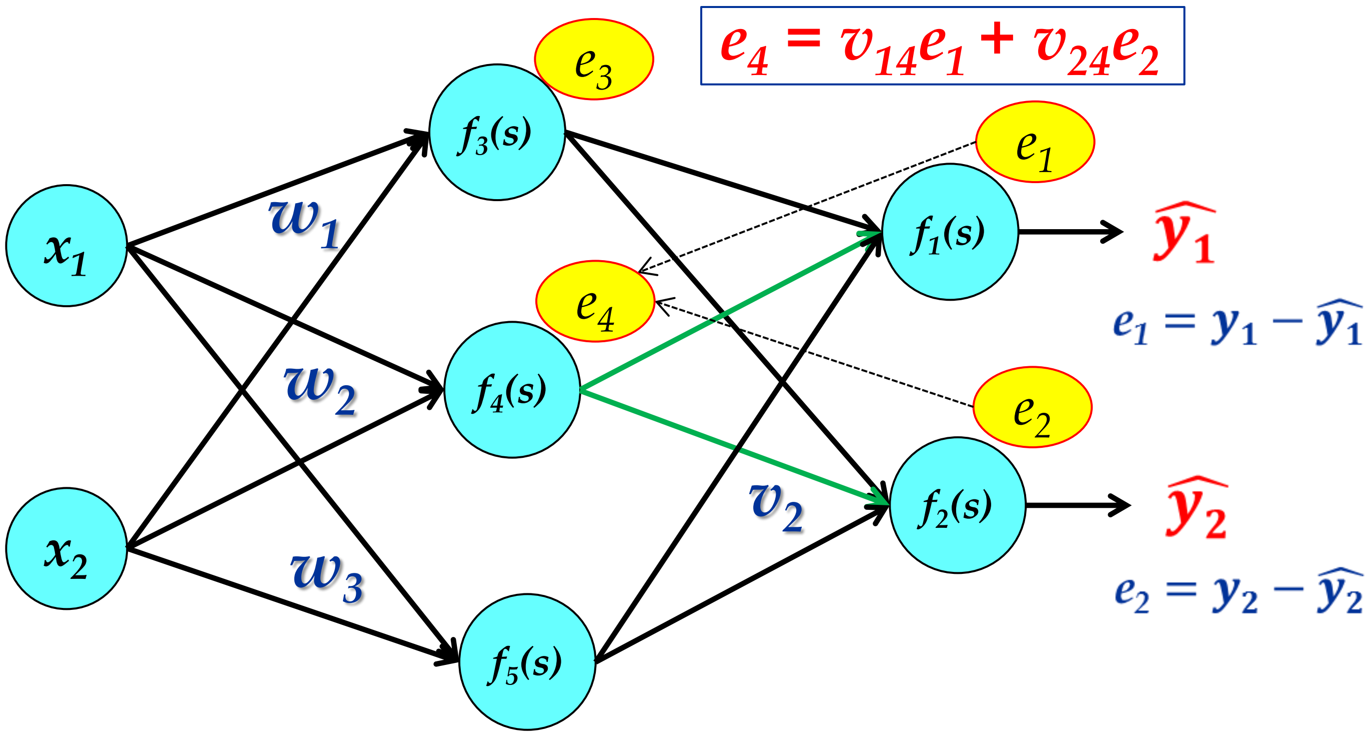
Learning in MLP

❖ **Understanding back-propagation on a simple example**



Learning in MLP

❖ **Understanding back-propagation on a simple example**

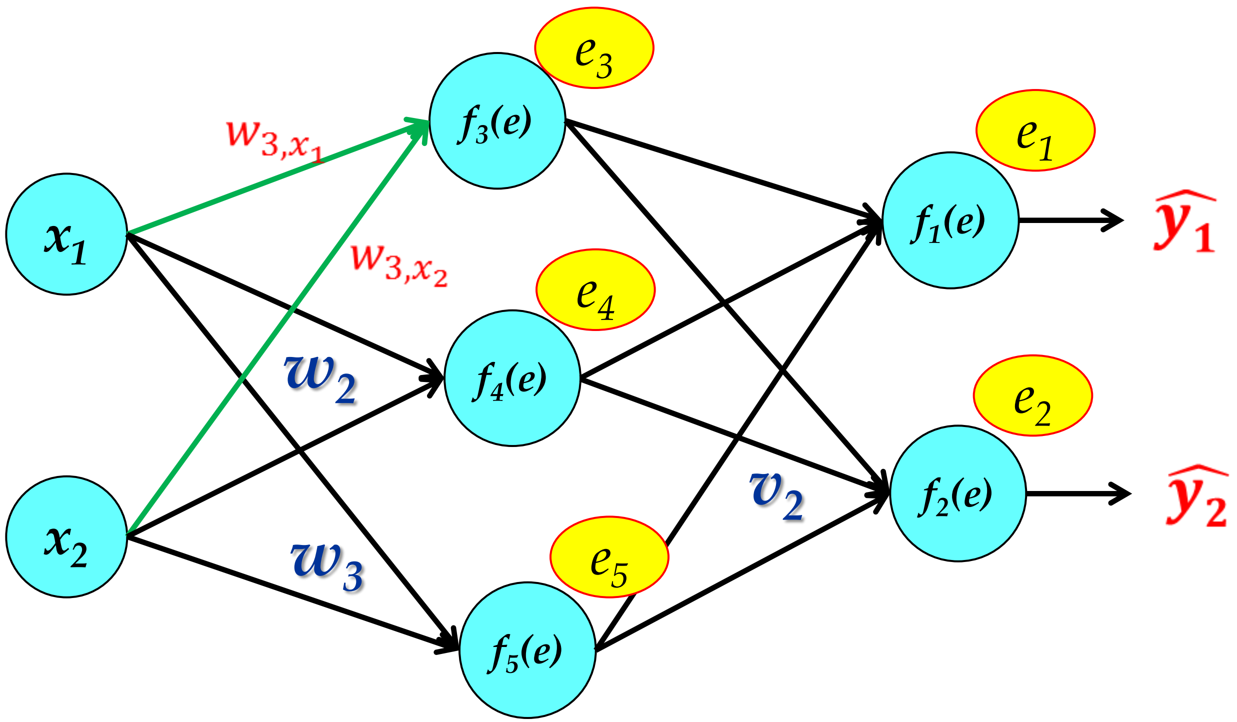


Learning in MLP

❖ **Understanding back-propagation on a simple example**

$$w'_{3,x_1} = w_{3,x_1} + \rho e_3 \frac{d f_3(s)}{ds} x_1$$

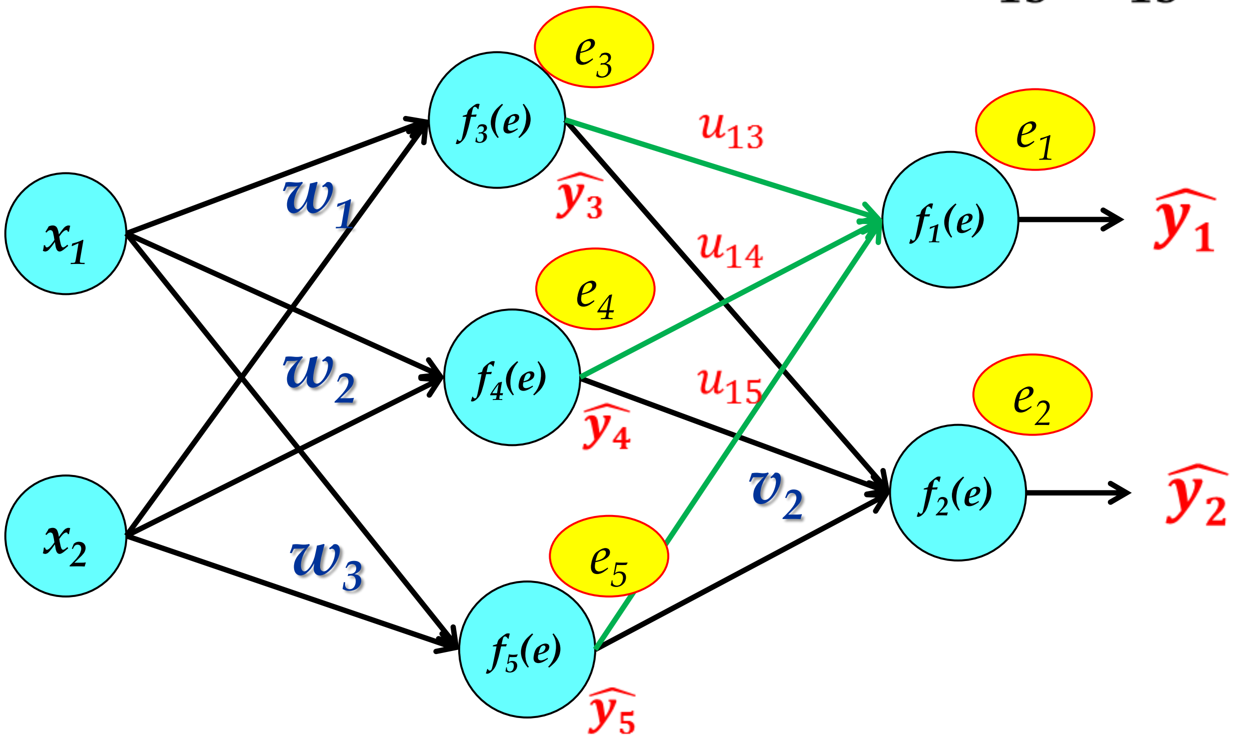
$$w'_{3,x_2} = w_{3,x_2} + \rho e_3 \frac{d f_3(s)}{ds} x_2$$



Learning in MLP

❖ **Understanding back-propagation on a simple example**

$$u'_{13}=u_{13}+\rho e_1 \frac{df_1(s)}{ds}\widehat{y_3}$$
$$u'_{14}=u_{14}+\rho e_1 \frac{df_1(s)}{ds}\widehat{y_4}$$
$$u'_{15}=u_{15}+\rho e_1 \frac{df_1(s)}{ds}\widehat{y_5}$$



Learning in MLP

❖ **Back-propagation Algorithm**

알고리즘 [4.5] 다층 퍼셉트론 (MLP) 학습을 위한 오류 역전파 알고리즘 (패턴 모드)

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

출력: 가중치 **u**와 **v**

알고리즘:

// 초기화

u와 **v**를 초기화한다.

$x_0 = z_0 = 1$; // 바이어스

repeat {

for (X의 샘플 각각에 대해) {

현재 샘플을 $\mathbf{x} = (x_1, x_2, \cdots, x_d)^{\mathrm{T}}$와 $\mathbf{t} = (t_1, t_2, \cdots, t_m)^{\mathrm{T}}$으로 표기한다.

// 전방 계산

for ($j = 1$ **to** p) $\{ z\_sum_j = \sum_{i=0}^{d} x_i u_{ij};\quad z_j = \tau(z\_sum_j); \}$ // (4.12)

for ($k = 1$ **to** m) $\{ o\_sum_k = \sum_{j=0}^{p} z_j v_{jk};\quad o_k = \tau(o\_sum_k); \}$ // (4.13)

// 오류 역전파

for ($k = 1$ **to** m) $\delta_k = (t_k - o_k)\tau'(o\_sum_k);$ // (4.18)

for (모든 $v_{jk}, 0 \le j \le p, 1 \le k \le m$ 에 대해) $\Delta v_{jk} = \rho \delta_k z_j;$ // (4.19)

for (j=1 **to** p) $\eta_j = \tau'(z\_sum_j) \sum_{k=1}^{m} \delta_k v_{jk};$ // (4.20)

for (모든 $u_{ij}, 0 \le i \le d, 1 \le j \le p$ 에 대해) $\Delta u_{ij} = \rho \eta_j x_i;$ // (4.21)

// 가중치 갱신

for (모든 $v_{jk}, 0 \le j \le p, 1 \le k \le m$ 에 대해) $v_{jk} = v_{jk} + \Delta v_{jk};$ // (4.17)

for (모든 $u_{ij}, 0 \le i \le d, 1 \le j \le p$ 에 대해) $u_{ij} = u_{ij} + \Delta u_{ij};$ // (4.17)

}

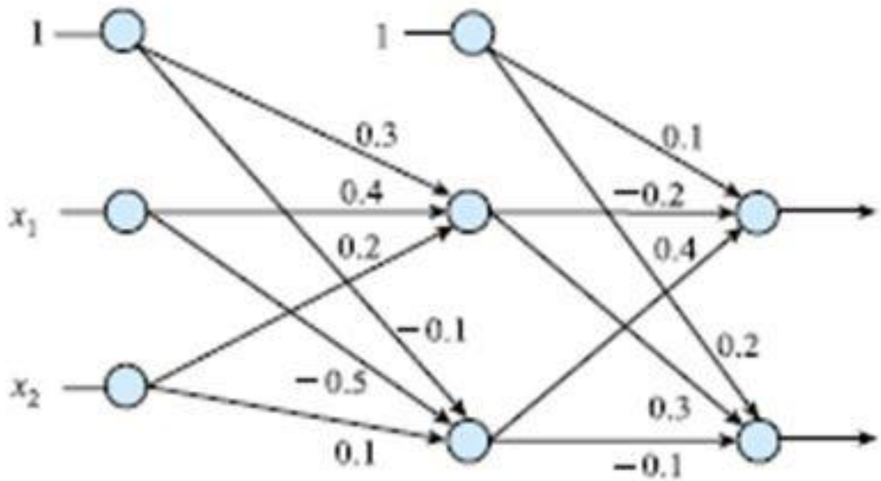
} **until** (stop-condition);

u와 **v**를 저장한다.

Learning in MLP

- **Simple Example in MLP Learning**

$$\mathbf{x} = (0.7,\ 0.2)^T,\ \mathbf{t} = (-1, 1)^T$$



Learning in MLP

- **Forward Computation**
  - Activation function: $\tau_2(x) = \dfrac{2}{1+e^{-\alpha x}} - 1$, $\alpha = 1,\ \rho = 0.2$
  - Line 6:

$$z\_sum_1 = 1*0.3+0.7*0.4+0.2*0.2 = 0.62000$$

$$z\_sum_2 = 1*(-0.1)+0.7*(-0.5)+0.2*0.1 = -0.43000$$

$$z_1 = \tau_2(0.62000) = 2/(1+e^{-0.62000})-1 = 0.30044$$

$$z_2 = \tau_2(-0.43000) = 2/(1+e^{0.43000})-1 = -0.21175$$

  - Line 7:

$$o\_sum_1 = 1*0.1+0.30044*(-0.2)+(-0.21175)*0.4 = -0.04479$$

$$o\_sum_2 = 1*0.2+0.30044*0.3+(-0.21175)*(-0.1) = 0.31131$$

$$o_1 = \tau_2(-0.04479) = -0.02239$$

$$o_2 = \tau_2(0.31131) = 0.15441$$

  - $x = (0.7, 0.2)^T,\ o = (-0.02239, 0.15441)^T,\ t = (-1, 1)^T$

  - ***Error***:

$$E = 0.5*((-1.0-(-0.02239))^2 + (1.0-0.15441)^2) = 0.83537$$

Learning in MLP

❖ Back-propagation

➢ Line 8:

$$\delta_1 = (-1.0+0.02239)\tau_2'(-0.04479) = -0.97761*0.5*(1+\tau_2(-0.04479))(1-\tau_2(-0.04479))$$
$$= -0.48856$$

$$\delta_2 = (1.0-0.15441)\tau_2'(0.31131) = 0.84559*0.5*(1+\tau_2(0.31131))(1-\tau_2(0.31131))$$
$$= 0.41271$$

➢ Line 9:

$$\Delta v_{01} = 0.2*(-0.48856)*1.0 = -0.09771$$

$$\Delta v_{02} = 0.2*0.41271*1.0 = 0.08254$$

$$\Delta v_{11} = 0.2*(-0.48856)*0.30044 = -0.02936$$

$$\Delta v_{12} = 0.2*0.41271*0.30044 = 0.02480$$

$$\Delta v_{21} = 0.2*(-0.48856)*(-0.21175) = 0.02069$$

$$\Delta v_{22} = 0.2*0.41271*(-0.21175) = -0.01748$$

➢ Line 10:

$$\eta_1 = \tau_2'(0.62000)*((-0.48856)*(-0.2)+0.41271*0.3) = 0.10076$$

$$\eta_2 = \tau_2'(-0.43000)*((-0.48856)*(0.4)+0.41271*(-0.1)) = -0.11304$$

Learning in MLP

- **Back-propagation**
  - Line 11:

$$\Delta u_{01} = 0.2*0.10076*1.0 = 0.02015$$

$$\Delta u_{02} = 0.2*(-0.11304)*1.0 = -0.02261$$

$$\Delta u_{11} = 0.2*0.10076*0.7 = 0.01411$$

$$\Delta u_{12} = 0.2*(-0.11304)*0.7 = -0.01583$$

$$\Delta u_{21} = 0.2*0.10076*0.2 = 0.00403$$

$$\Delta u_{22} = 0.2*(-0.11304)*0.2 = -0.00452$$

  - Line 12:

$$v_{01} = 0.1 - 0.09771 = 0.00229$$

$$v_{02} = 0.2 + 0.08254 = 0.28254$$

$$v_{11} = -0.2 - 0.02936 = -0.22936$$

$$v_{12} = 0.3 + 0.02480 = 0.32480$$

$$v_{21} = 0.4 + 0.02069 = 0.42069$$

$$v_{22} = -0.1 - 0.01748 = -0.11748$$

Line 13:

$$u_{01} = 0.3 + 0.02015 = 0.32015$$

$$u_{02} = -0.1 - 0.02261 = -0.12261$$

$$u_{11} = 0.4 + 0.01411 = 0.41411$$

$$u_{12} = -0.5 - 0.01583 = -0.51583$$

$$u_{21} = 0.2 + 0.00403 = 0.20403$$

$$u_{22} = 0.1 - 0.00452 = 0.09548$$

Learning in MLP

- **Learning Effect of One Iteration**
  - Line 6 & 7:

$$z\_sum_1 = 1.0*0.32015+0.7*0.41411+0.2*0.20403 = 0.65083$$

$$z\_sum_2 = 1.0*(-0.12261)+0.7*(-0.51583)+0.2*0.09548 = -0.46460$$

$$z_1 = 0.31440$$

$$z_2 = -0.22821$$

$$o\_sum_1 = 1.0*0.00229+0.31440*(-0.22936)+(-0.22821)*0.42069 = -0.16582$$

$$o\_sum_2 = 1.0*0.28254+0.31440*(0.32480)+(-0.22821)*(-0.11748) = 0.41147$$

$$o_1 = -0.08272$$

$$o_2 = 0.20288$$

  - $o = (-0.08272, 0.20288)^T$, $t = (-1,1)^T$
  - ***Error***: *E=0.73840 (vs. 0.83537)*

More Considerations

- **Typical complaints**
  - # of layers
  - # of hidden units per layer
  - The gradient descent learning rate
  - The initialization
  - the stopping iteration or weight regularization

significant

- **Random Initialization**
  - *Small random weights (say, uniform between -0.1 and 0.1)*
  - *By training a collection of networks, each with a different random initialization, we can often obtain better solutions*

More Considerations

❖ Initialization Tip

Initial Value

초기값 Settings

- Random 하게 주되 특정 구역안에서 Random 하게 주는것이 좋다.

tanh 를 Activation 으로 사용하는 경우

$$\text{Interval} = [-\sqrt{\frac{6}{fan_{in}+fan_{out}}}, \sqrt{\frac{6}{fan_{in}+fan_{out}}}]$$

fan_{in} = the number of units in the *(i-1)*th layer.

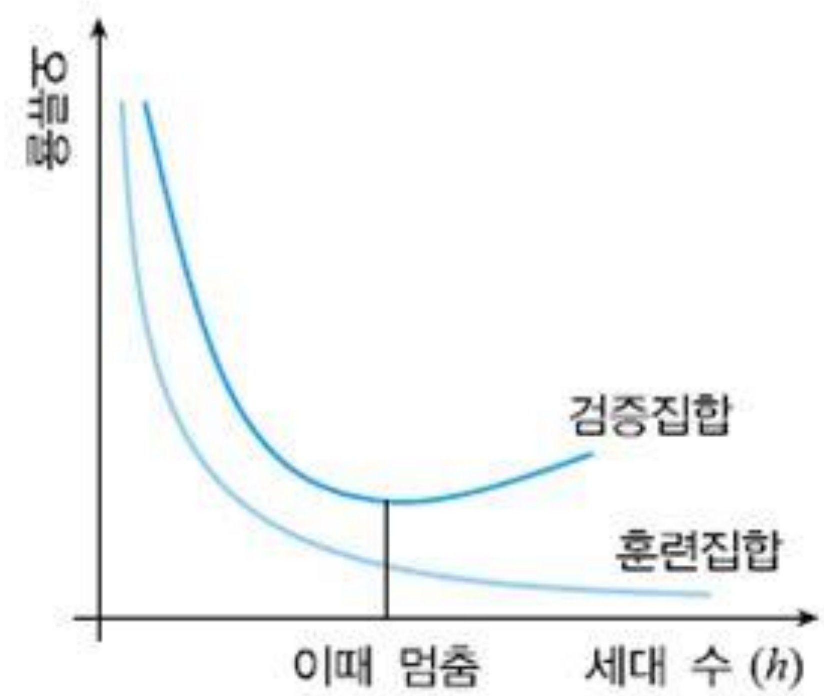
fan_{out} = the number of units in the *i*th layer

sigmoid 를 Activation 으로 사용하는 경우

$$\text{Interval} = [-4\sqrt{\frac{6}{fan_{in}+fan_{out}}}, 4\sqrt{\frac{6}{fan_{in}+fan_{out}}}]$$

More Considerations

- **When is the proper number of iteration for early stopping?**



Python Code and Practice

- **You should install Python 2.7 and Numpy**

- **Download from: http://nlpmlir.blogspot.kr/2016/02/multilayer-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