기계학습 (Machine Learning)

L09

- Multi-Layer Perceptron

한밭대학교

정보통신공학과

최 해 철

국립 한밭대학교

VISUAL MEDIA LAB.
HANBAT NATIONAL UNIVERSITY

ToC

- Multi-Layer Perceptron (MLP)
- Training & Testing MLP
- Remarks on MLP

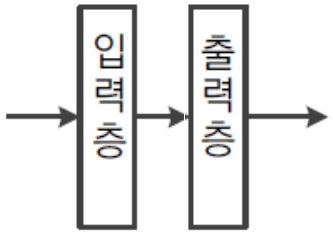
References

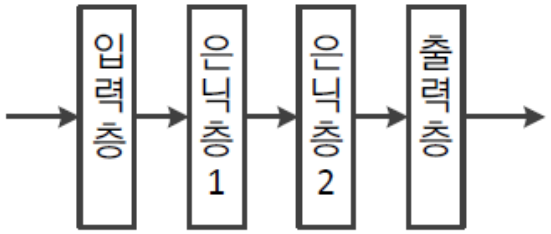
- *기계 학습 "3장 다층 퍼셉트론"* by 오일석, *패턴 인식* by 오일석

1. Multi-Layer Perceptron

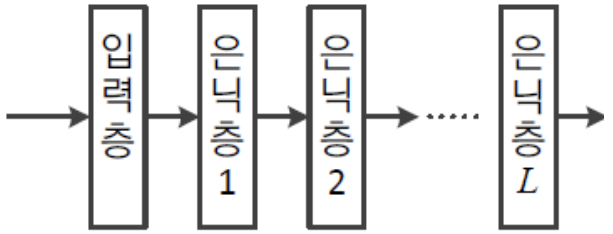
Neural Networks – Types of NN

◆ 단층 신경망, 얕은 신경망, 깊은 신경망



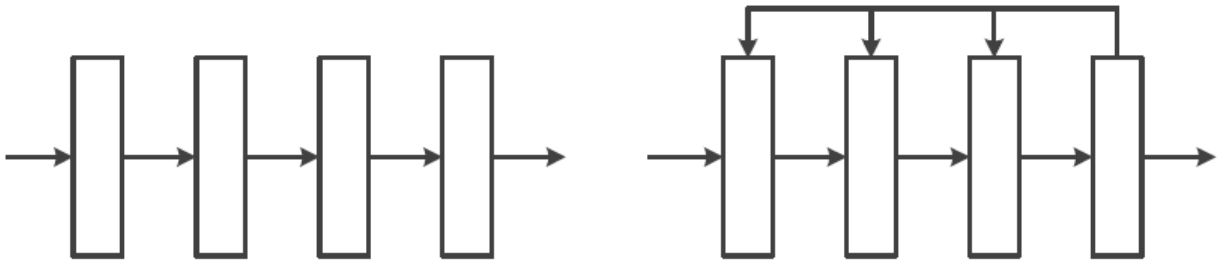


3-layer Perceptron



깊은 신경망 (Deep Neural Network) : $L \geq 4$

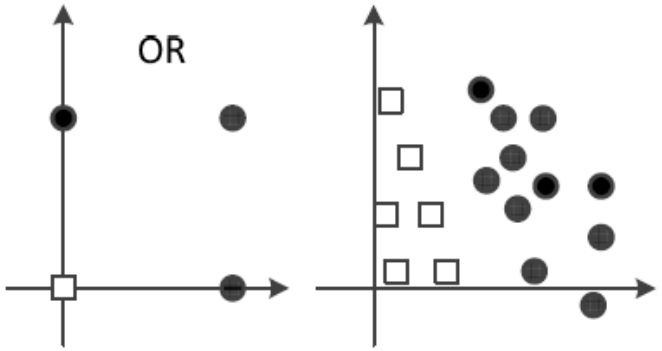
◆ 전방 신경망 vs. 순환 신경망



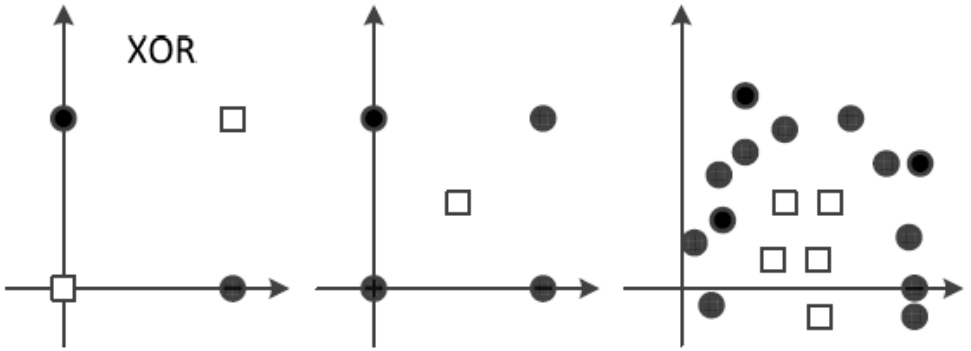
MLP – Intro.

◆ Perceptron의 한계 및 MLP로의 발전

- Perceptron의 한계: _____에서는 일정한 양의 오류 발생
  - 예) XOR 문제에서는 75%가 정확률 한계



(a) 선형분리 가능



(b) 선형분리 불가능

입력		출력
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

MLP – Intro. (cont'd)

◆ MLP의 핵심 아이디어

1. _____(hidden layer)의 도입
   - 은닉층을 통해 원래 특징 공간을 분류에 유리한 새로운 특징 공간으로 변환 가능
   - 이전 특징공간에서 선형분리 불가능한 문제를 변환된 특징공간에서 선형분리 가능하도록 할 수 있음
     (예: XOR 문제의 해결)

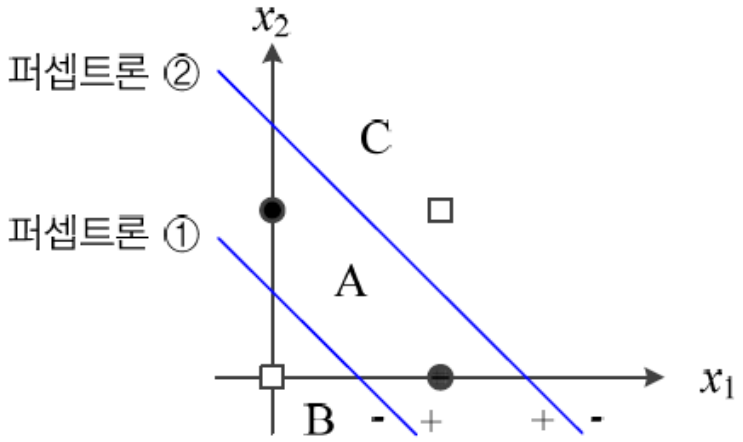
2. _____ **활성함수**(sigmoid activation function)의 도입
   - 계단 활성함수: 경성 의사결정 (hard decision)
     → 출력값이 곧 의사결정(분류) 결과
   - 시그모이드 활성함수: 연성 의사결정 (soft decision)
     → 출력값을 신뢰도(확률)로 간주하여 융통성 있는 의사결정 가능

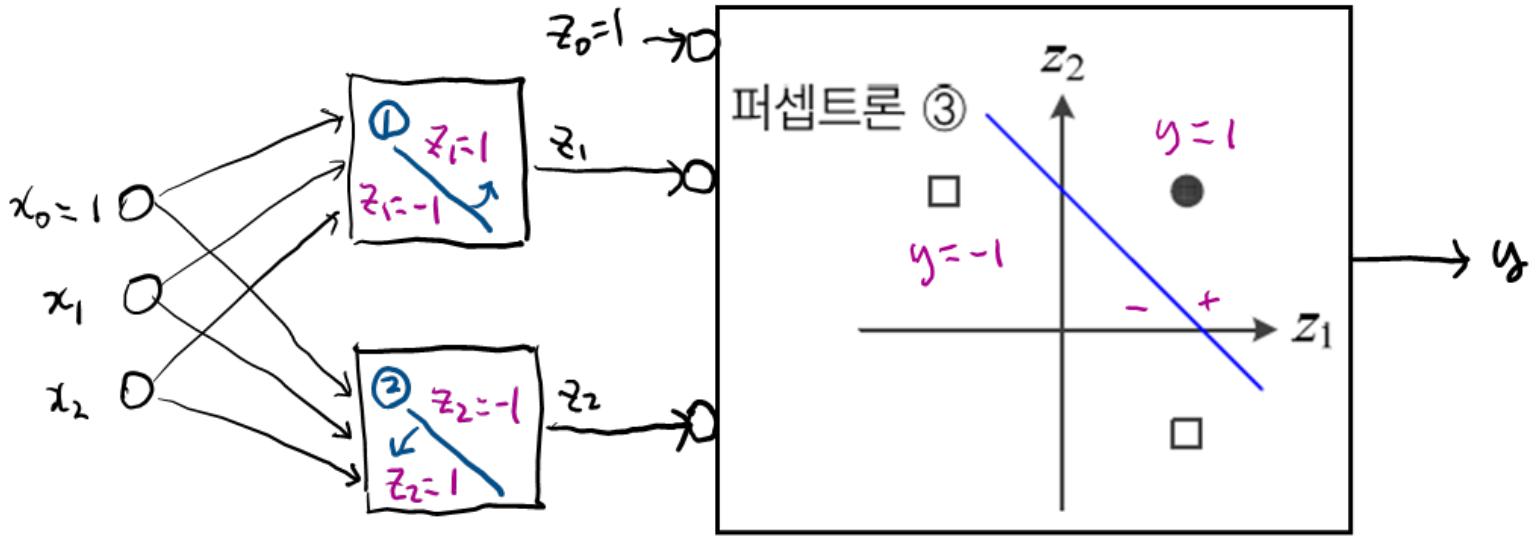
3. _____(back-propagation) 알고리즘 사용
   - 다층 신경망의 학습이 어려웠던 문제를 해결

XOR 문제의 해결

◆ 선형 결정 경계 (즉, 퍼셉트론) _____를 사용하면 XOR 문제 해결이 가능

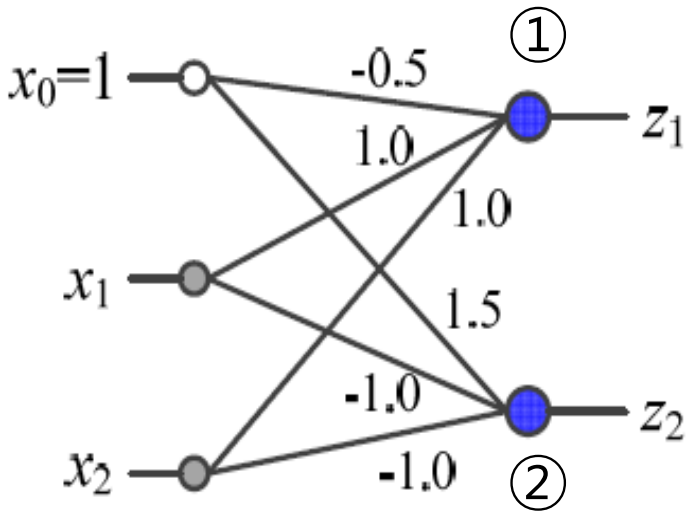
- 활성함수로 계단함수를 사용($\tau(s) = \pm 1$)





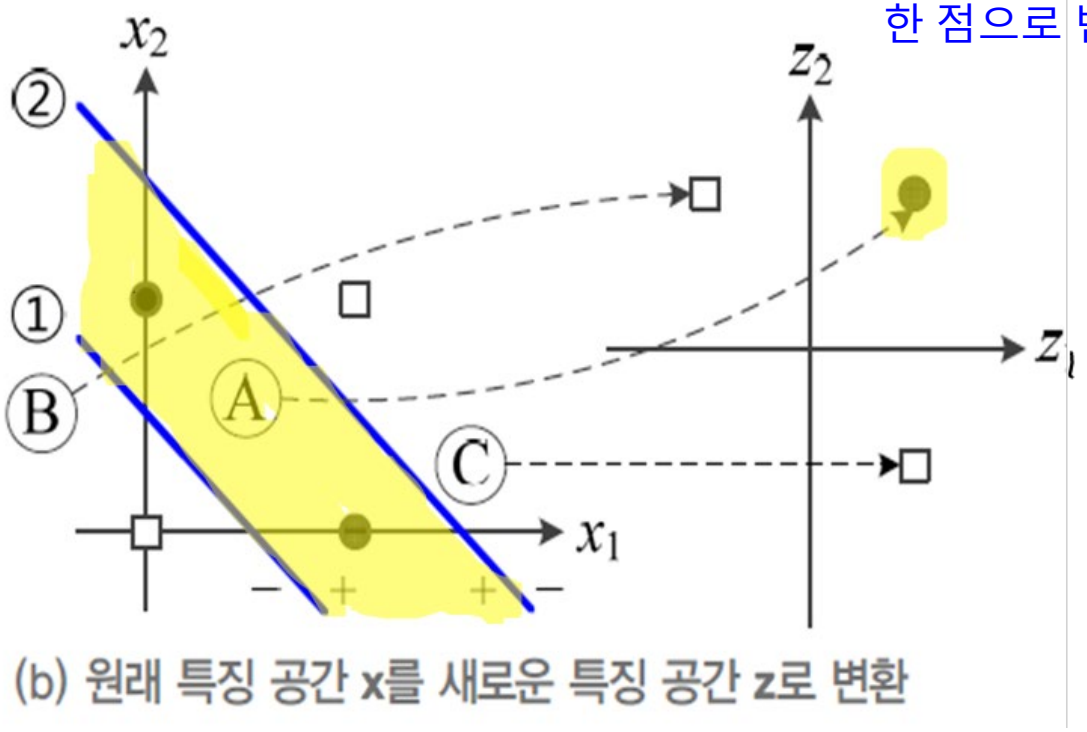
XOR 문제의 해결 (cont'd)

- **Step 1) 두 Perceptron을 _____로 결합한다.**
  - 원래 특징 공간 $\mathbf{x} = (x_1, x_2)^{\mathrm{T}}$를 _____ $\mathbf{z} = (z_1, z_2)^{\mathrm{T}}$로 **변환**



(a) 두 퍼셉트론을 병렬로 결합

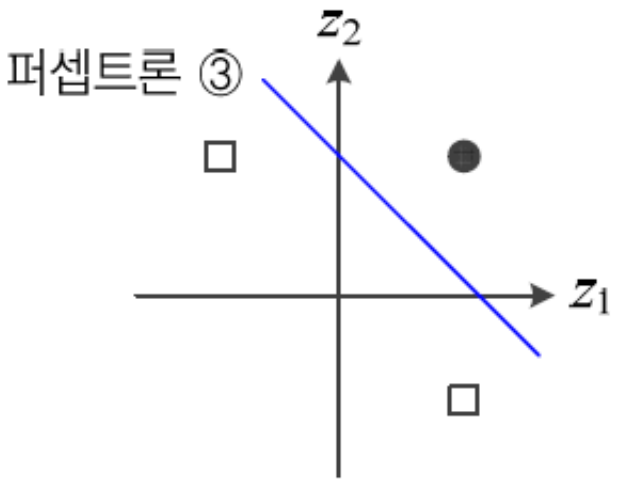
각 영역이 2차원의 한 점으로 변환



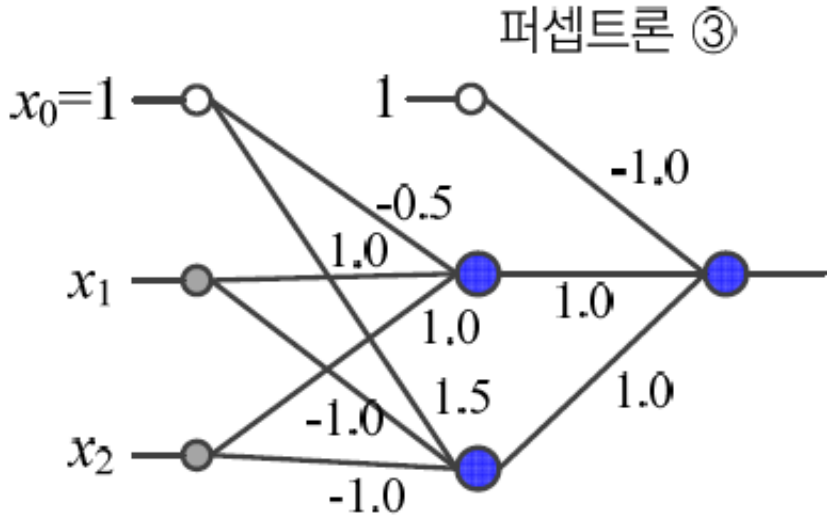
(b) 원래 특징 공간 x를 새로운 특징 공간 z로 변환

XOR 문제의 해결 (cont'd)

- **Step 2) Perceptron 1개를 _____로 결합한다. → *Multilayer Perceptron***
  - 새로운 특징 공간 **z**에서 _____를 수행



(a) 새로운 특징 공간에서 분할

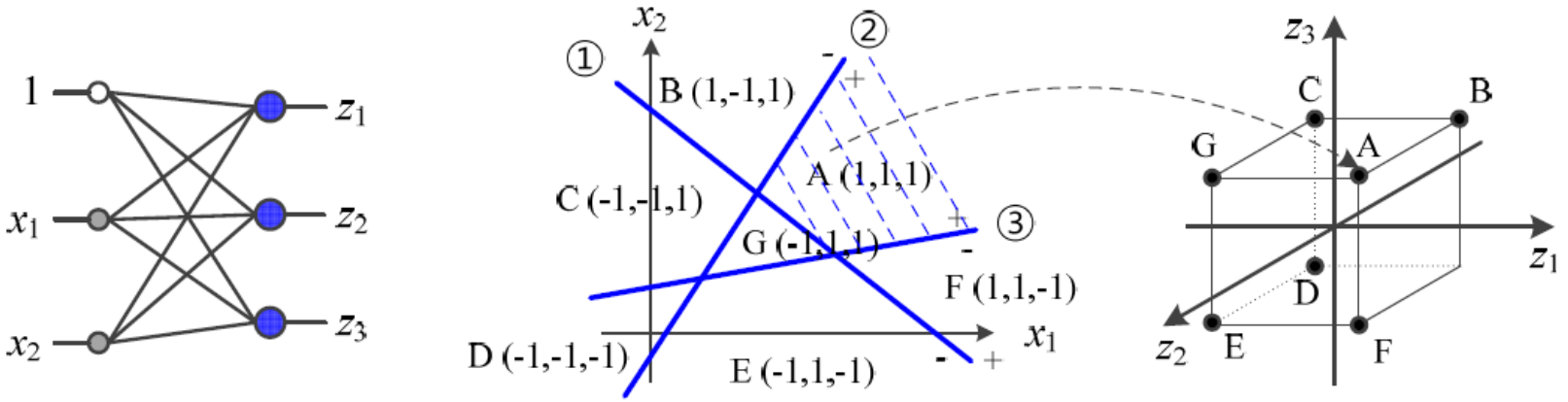


(b) 퍼셉트론 3개를 결합한 다층 퍼셉트론

특징 공간 변환

◆ 퍼셉트론의 개수와 특징 공간의 변환

- 3개 퍼셉트론을 병렬결합하면, 2차원 공간이 7개 영역으로 나뉨
- 활성함수로 계단함수를 사용($\tau(s) = \pm 1$)하므로 각영역이 3차원 상의 한 점으로 변환됨



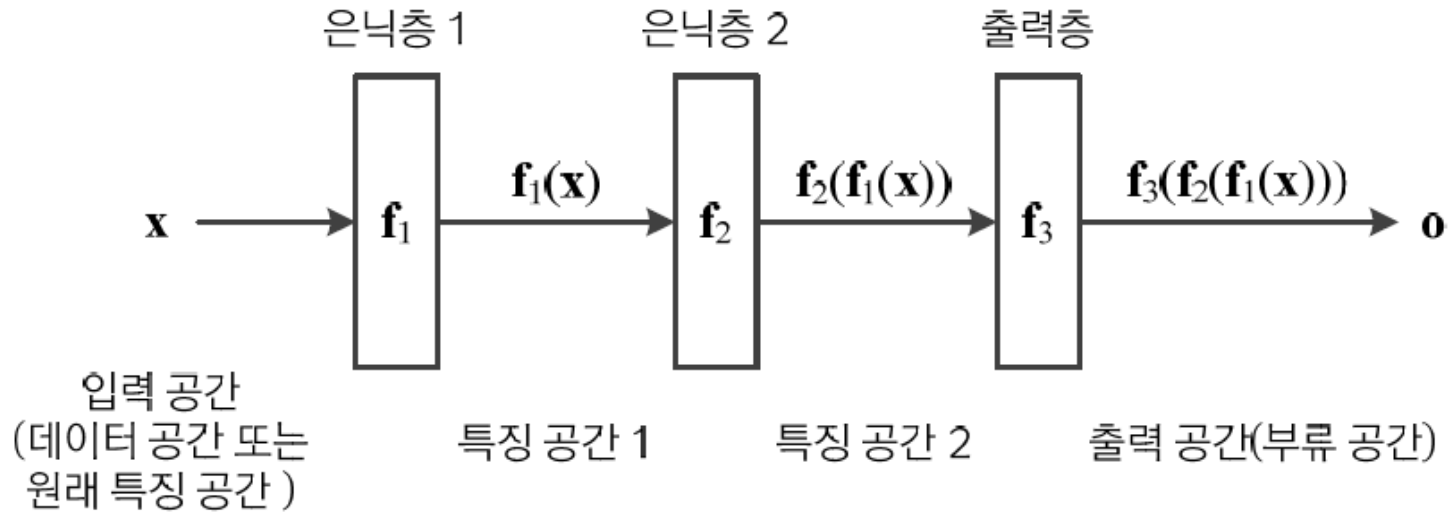
(a) 퍼셉트론 3개를 결합 (b) 7개 부분공간으로 나눔 (c) 3차원 공간의 점으로 매핑

- 이를 일반화 하면, p개 퍼셉트론을 결합하여 p차원 공간으로의 변환이 가능함을 알 수 있다.
  - Cf.) 부분공간의 개수 =

특징 공간 변환 (cont'd)

◆ 은닉층의 역할: _____

- 은닉층은 특징 벡터를 (분류에 더 유리한) 새로운 특징 공간으로 변환
- 현대 기계 학습에서는 **특징 학습**이라*feature learning* 부름 (딥러닝은 더 많은 층을 거쳐 특징 학습을 함)



활성함수 (Activation Function)

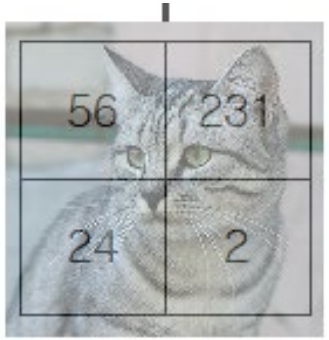
◆ ML_L07_Multiclass classification, Activation Function 참고

함수 이름	함수	1차 도함수	범위
계단	$\tau(s) = \begin{cases} 1 & s \geq 0 \\ -1 & s < 0 \end{cases}$	$\tau'(s) = \begin{cases} 0 & s \neq 0 \\ \text{불가} & s = 0 \end{cases}$	-1과 1
로지스틱 시그모이드	$\tau(s) = \frac{1}{1 + e^{-as}}$	$\tau'(s) = a\tau(s)(1 - \tau(s))$	(0,1)
하이퍼볼릭 탄젠트	$\tau(s) = \frac{2}{1 + e^{-as}} - 1$	$\tau'(s) = \frac{a}{2}(1 - \tau(s)^2)$	(-1,1)
소프트플러스	$\tau(s) = \log_e(1 + e^s)$	$\tau'(s) = \frac{1}{1 + e^{-s}}$	(0, ∞)
렉티파이어(ReLU)	$\tau(s) = \max(0, s)$	$\tau'(s) = \begin{cases} 0 & s < 0 \\ 1 & s > 0 \\ \text{불가} & s = 0 \end{cases}$	[0, ∞)

Model Architecture

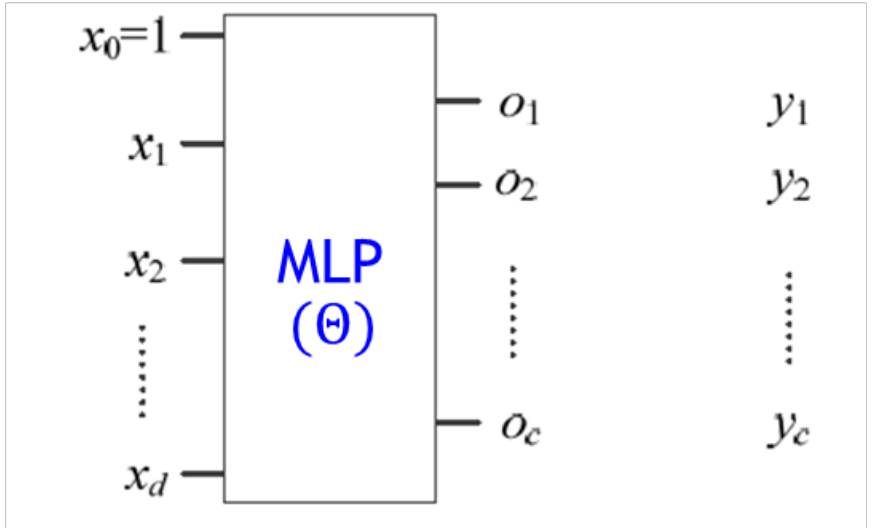
◆ 입력층과 출력층

- 입력층과 출력층 노드(neuron)의 개수는 주어진 문제에 의해 정해짐
  - 입력 특징의 개수: ___ → 입력층 노드 개수: ___
  - 정답 벡터의 차원: ___ → 출력층 노드 개수: ___



Input image

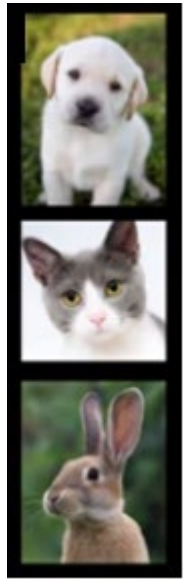
입력: **x** 출력: **o** 정답: **y**



dog

cat

rabbit

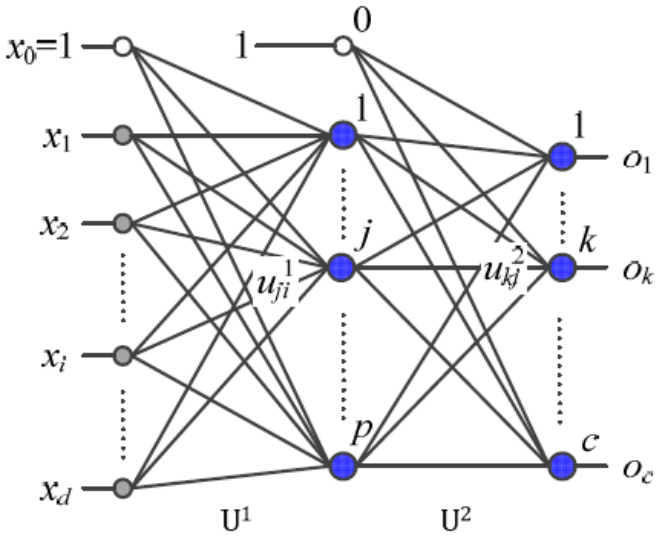


Model Architecture (cont'd)

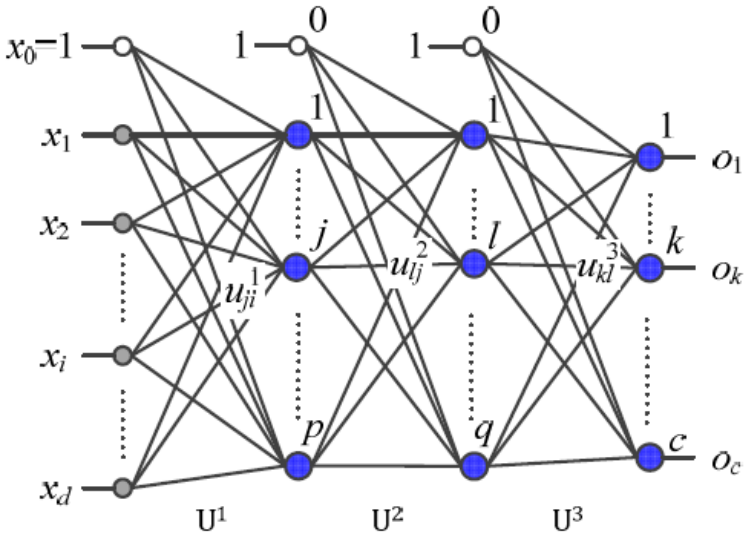
◆ 은닉층

● 은닉층의 개수 및 각 은닉층의 노드 개수는 _____ (사용자가 설정)

- 은닉층 1개 → **2층 Perceptron** : _____
- 은닉층 2개 → **3층 Perceptron** : _____



(a) 2층 퍼셉트론



(b) 3층 퍼셉트론

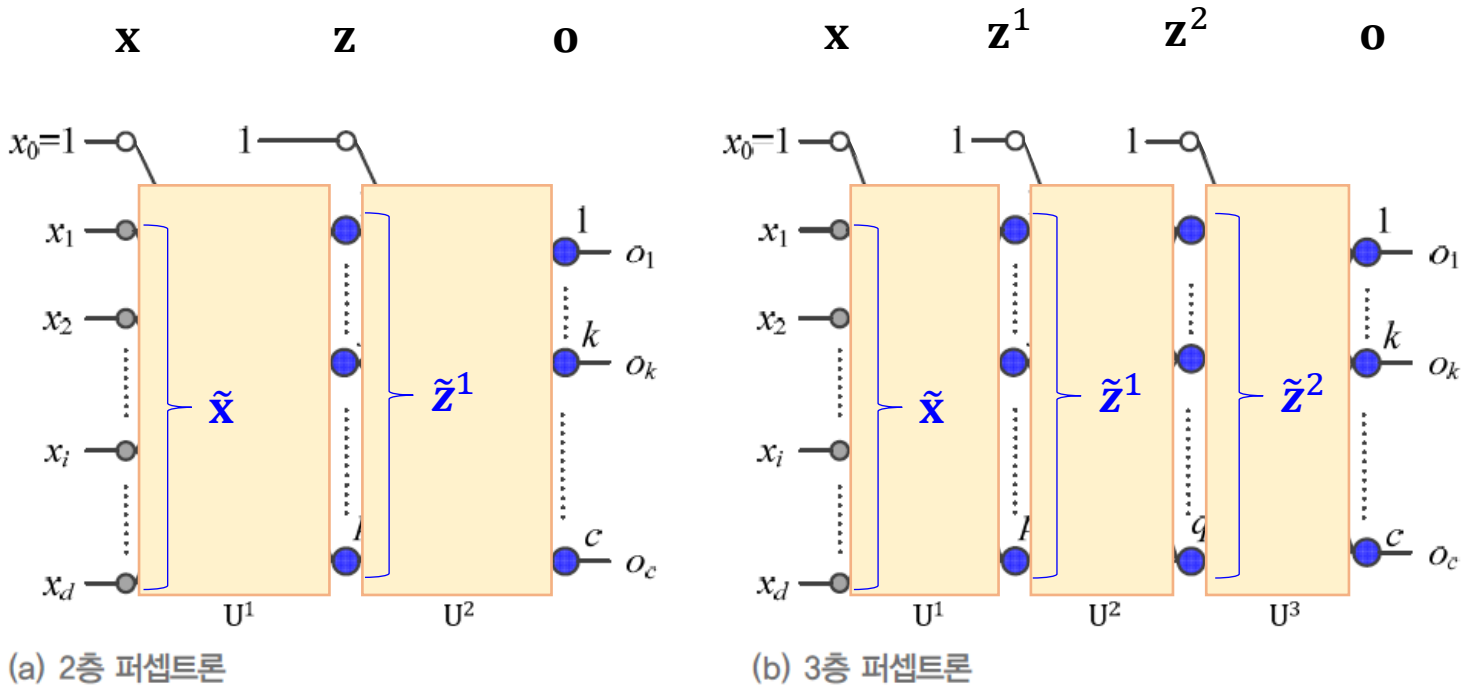
$$\mathbf{U}^1 = \begin{pmatrix} u_{10}^1 & u_{11}^1 & \cdots & u_{1d}^1 \\ u_{20}^1 & u_{21}^1 & \cdots & u_{2d}^1 \\ \vdots & \vdots & \ddots & \vdots \\ u_{p0}^1 & u_{p1}^1 & \cdots & u_{pd}^1 \end{pmatrix}$$

$$\mathbf{U}^2 = \begin{pmatrix} u_{10}^2 & u_{11}^2 & \cdots & u_{1p}^2 \\ u_{20}^2 & u_{21}^2 & \cdots & u_{2p}^2 \\ \vdots & \vdots & \ddots & \vdots \\ u_{c0}^2 & u_{c1}^2 & \cdots & u_{cp}^2 \end{pmatrix}$$

Model Architecture (cont'd)

◆ 각 계층의 출력

- **입력층:** $\tilde{\mathbf{x}} = (x_1 \cdots x_d)^T$, $\mathbf{x} = (x_0\ x_1 \cdots x_d)^T = (1\ x_1 \cdots x_d)^T$
- **은닉층:** $\tilde{\mathbf{z}} = (z_1 \cdots z_p)^T$, $\mathbf{z} = (z_0\ z_1 \cdots z_p)^T = (1\ z_1 \cdots z_p)^T$ → _____
- **출력층:** $\mathbf{o} = \tilde{\mathbf{o}} = (o_1 \cdots o_c)^T$

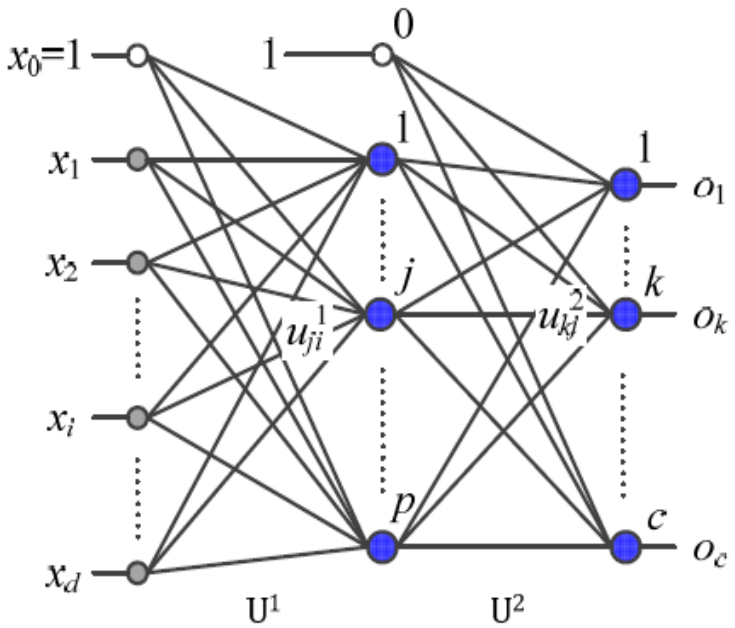


(a) 2층 퍼셉트론 (b) 3층 퍼셉트론

Model Architecture (cont'd)

◆ 각 계층의 Parameters (가중치)

- u_{ji}^{l} : l-1번째 층의 ***i*번째 노드**로부터 l번째 층의 ***j*번째 노드**와 연결되는 가중치
- $\mathbf{u}_{j}^{l}$: l-1번째 층으로부터 l번째 층의 ***j*번째 노드**에 연결되는 모든 가중치 (_____)
- $\mathbf{U}^{l}$: l-1번째 층으로부터 ***l*번째 층**에 연결되는 모든 가중치 (_____)



$$\mathbf{U}^{1} = \begin{pmatrix} u_{10}^{1} & u_{11}^{1} & \cdots & u_{1d}^{1} \\ u_{20}^{1} & u_{21}^{1} & \cdots & u_{2d}^{1} \\ \vdots & \vdots & \ddots & \vdots \\ u_{p0}^{1} & u_{p1}^{1} & \cdots & u_{pd}^{1} \end{pmatrix}$$

$$\mathbf{U}^{2} = \begin{pmatrix} u_{10}^{2} & u_{11}^{2} & \cdots & u_{1p}^{2} \\ u_{20}^{2} & u_{21}^{2} & \cdots & u_{2p}^{2} \\ \vdots & \vdots & \ddots & \vdots \\ u_{c0}^{2} & u_{c1}^{2} & \cdots & u_{cp}^{2} \end{pmatrix}$$

Forward Computation (전방계산)

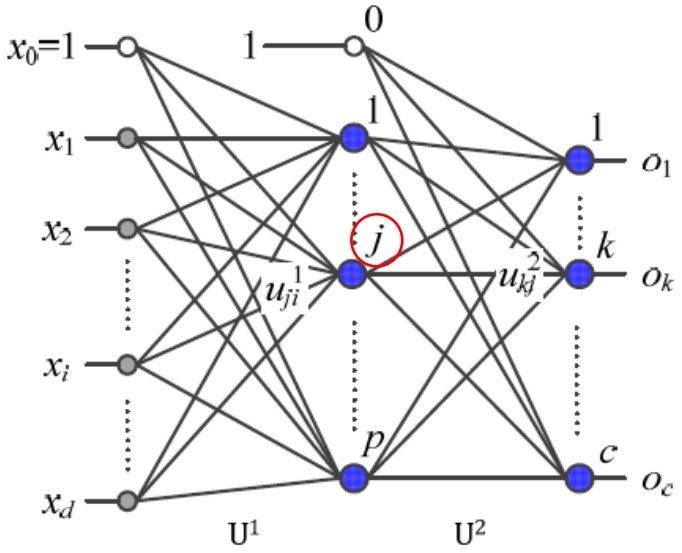
◆ 은닉층 노드의 연산

● $\mathbf{u}_j^1$은 j번째 은닉 노드에 연결된 가중치 벡터 ($\mathbf{U}^1$의 j번째 행)

j번째 은닉 노드의 연산:

$$z_j = \tau(zsum_j),\ j = 1,2,\cdots,p$$

이때 $zsum_j = \mathbf{u}_j^1\mathbf{x}$이고 $\mathbf{u}_j^1 = (u_{j0}^1, u_{j1}^1, \cdots, u_{jd}^1)$, $\mathbf{x} = (1, x_1, x_2, \cdots, x_d)^T$



● **행렬-벡터 형태** : _____, _____

$$\tilde{\mathbf{x}} = (x_1\ \cdots\ x_d)^T,\ \mathbf{x} = (x_0\ x_1\ \cdots\ x_d)^T$$

$$\tilde{\mathbf{z}} = (z_1\ \cdots\ z_p)^T,\ \mathbf{z} = (z_0\ z_1\ \cdots\ z_p)^T$$

$$\mathbf{o} = \tilde{\mathbf{o}} = (o_1\ \cdots\ o_c)^T$$

$$\mathbf{U}^1 = \begin{pmatrix} u_{10}^1 & u_{11}^1 & \cdots & u_{1d}^1 \\ u_{20}^1 & u_{21}^1 & \cdots & u_{2d}^1 \\ \vdots & \vdots & \ddots & \vdots \\ u_{p0}^1 & u_{p1}^1 & \cdots & u_{pd}^1 \end{pmatrix}$$

$$\mathbf{U}^2 = \begin{pmatrix} u_{10}^2 & u_{11}^2 & \cdots & u_{1p}^2 \\ u_{20}^2 & u_{21}^2 & \cdots & u_{2p}^2 \\ \vdots & \vdots & \ddots & \vdots \\ u_{c0}^2 & u_{c1}^2 & \cdots & u_{cp}^2 \end{pmatrix}$$

Forward Computation (전방계산) (cont'd)

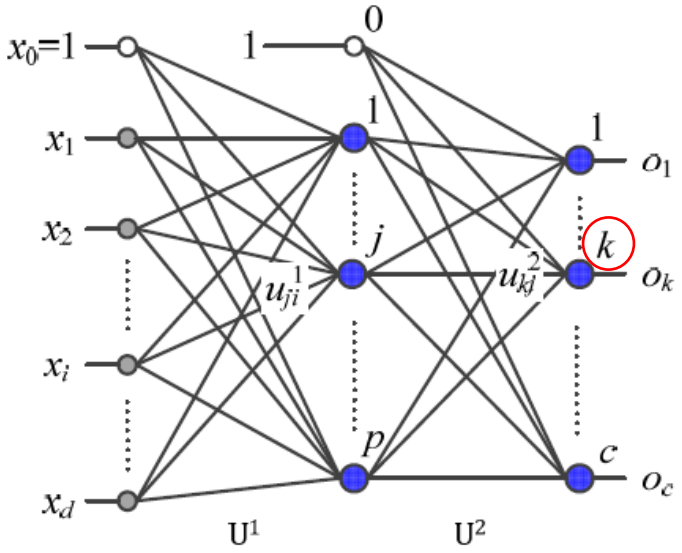
◆ 출력층 노드의 연산

● $\mathbf{u}_k^2$는 k번째 출력 노드에 연결된 가중치 벡터 ($\mathbf{U^2}$의 k번째 행)

k번째 출력 노드의 연산:

$$o_k = \tau(osum_k),\ k = 1,2,\cdots,c$$

이때 $osum_k = \mathbf{u}_k^2\mathbf{z}$이고 $\mathbf{u}_k^2 = (u_{k0}^2, u_{k1}^2, \cdots, u_{kp}^2),\ \mathbf{z} = (1, z_1, z_2, \cdots, z_p)^{\mathrm{T}}$



● **행렬-벡터 형태** : _____, _____

$$\tilde{\mathbf{x}} = (x_1\ \cdots\ x_d)^T,\ \mathbf{x} = (x_0\ x_1\ \cdots\ x_d)^T$$

$$\tilde{\mathbf{z}} = (z_1\ \cdots\ z_p)^T,\ \mathbf{z} = (z_0\ z_1\ \cdots\ z_p)^T$$

$$\mathbf{o} = \tilde{\mathbf{o}} = (o_1\ \cdots\ o_c)^T$$

$$\mathbf{U}^1 = \begin{pmatrix} u_{10}^1 & u_{11}^1 & \cdots & u_{1d}^1 \\ u_{20}^1 & u_{21}^1 & \cdots & u_{2d}^1 \\ \vdots & \vdots & \ddots & \vdots \\ u_{p0}^1 & u_{p1}^1 & \cdots & u_{pd}^1 \end{pmatrix}$$

$$\mathbf{U}^2 = \begin{pmatrix} u_{10}^2 & u_{11}^2 & \cdots & u_{1p}^2 \\ u_{20}^2 & u_{21}^2 & \cdots & u_{2p}^2 \\ \vdots & \vdots & \ddots & \vdots \\ u_{c0}^2 & u_{c1}^2 & \cdots & u_{cp}^2 \end{pmatrix}$$

2. Training MLP

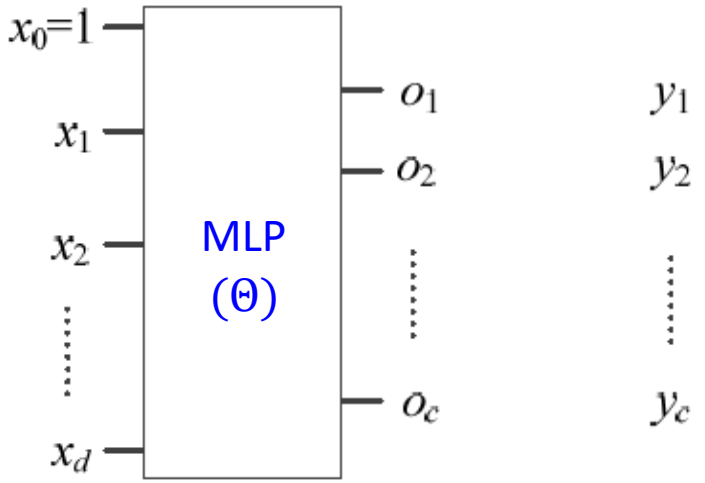
Training Set & Cost Function

◆ Training Set (of size n) c.f. $\mathbf{x}^{(1)}$, $\mathbf{y}^{(1)}$

- Feature vector 집합 $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\}$
- Target vector 집합 $\mathbb{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n\}$

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_n^T \end{pmatrix}, \quad \mathbf{Y} = \begin{pmatrix} \mathbf{y}_1^T \\ \mathbf{y}_2^T \\ \vdots \\ \mathbf{y}_n^T \end{pmatrix}$$

입력: $\mathbf{x}$ 출력: $\mathbf{o}$ 정답: $\mathbf{y}$

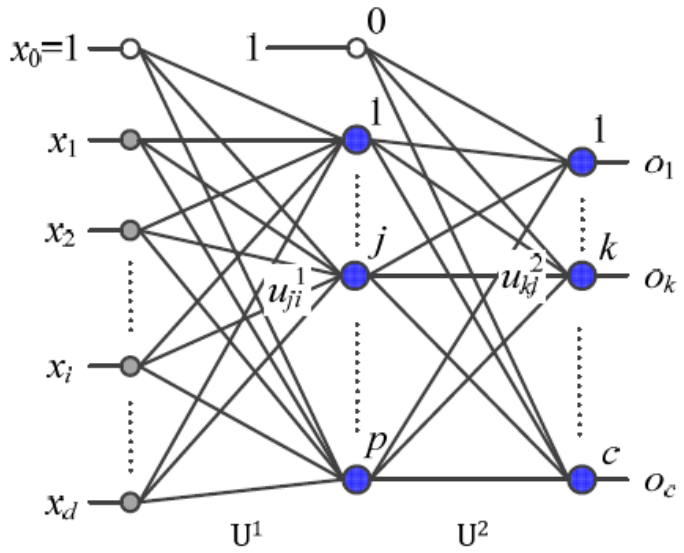


◆ MSE Cost :

$$J(\Theta) = \begin{cases} \text{온라인 모드:} & \frac{1}{2}\|\mathbf{y} - \mathbf{o}\|_2^2 \\ \text{배치 모드:} & \frac{1}{2n}\sum_{i=1}^{n}\|\mathbf{y}_i - \mathbf{o}_i\|_2^2 \end{cases}$$

Training Set & Cost Function

◆ 어떻게 훈련하여 $\Theta = \{\mathbf{U}^1, \mathbf{U}^2\}$을 최적화 시킬까?



(a) 2층 퍼셉트론

$$\mathbf{U}^1 = \begin{pmatrix} u_{10}^1 & u_{11}^1 & \cdots & u_{1d}^1 \\ u_{20}^1 & u_{21}^1 & \cdots & u_{2d}^1 \\ \vdots & \vdots & \ddots & \vdots \\ u_{p0}^1 & u_{p1}^1 & \cdots & u_{pd}^1 \end{pmatrix} \qquad \mathbf{U}^2 = \begin{pmatrix} u_{10}^2 & u_{11}^2 & \cdots & u_{1p}^2 \\ u_{20}^2 & u_{21}^2 & \cdots & u_{2p}^2 \\ \vdots & \vdots & \ddots & \vdots \\ u_{c0}^2 & u_{c1}^2 & \cdots & u_{cp}^2 \end{pmatrix}$$

Parameter Update by SGD (online mode)

알고리즘 3-3 다층 퍼셉트론을 위한 스토케스틱 경사 하강법

입력: 훈련집합 $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\}$, $\mathbb{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n\}$, 학습률 ρ

출력: 가중치 행렬 $\mathbf{U}^1$과 $\mathbf{U}^2$

```
1  U¹과 U²를 초기화한다.
2  repeat
3     𝕏의 순서를 섞는다.
4     for (𝕏의 샘플 각각에 대해)
5           식 (3.15)로 전방 계산을 하여 o를 구한다.
6           ∂J/∂U¹와 ∂J/∂U²를 계산한다.
7           식 (3.21)로 U¹과 U²를 갱신한다.
8  until (멈춤 조건)
```

$$\tilde{\mathbf{z}}_{sum} = \mathbf{U}^1\mathbf{x},\ \tilde{\mathbf{z}} = \tau(\tilde{\mathbf{z}}_{sum})$$

$$\tilde{\mathbf{o}}_{sum} = \mathbf{U}^2\mathbf{z},\ \tilde{\mathbf{o}} = \tau(\tilde{o}_{sum})$$

Gradient 계산 – 출력층

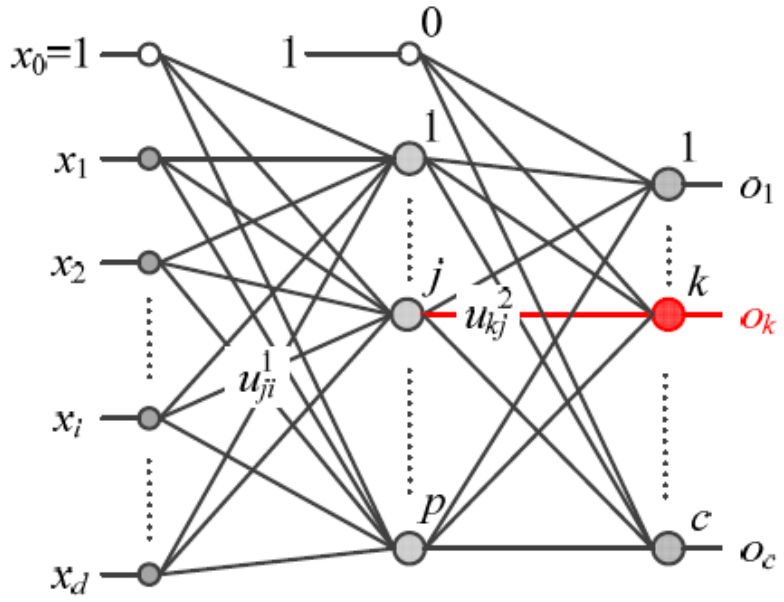
◆ $\dfrac{\partial J}{\partial \mathbf{U}^2}$: $\dfrac{\partial J}{\partial u_{kj}^2}$ $1 \le k \le c$, $0 \le j \le p$

$$\frac{\partial J}{\partial u_{kj}^2} = \delta_k z_j$$

where, $\delta_k = (o_k - y_k)\tau'(o_{sum_k})$

$o_{sum_k} = z_0 u_{k0}^2 + z_1 u_{k1}^2 + \cdots + z_j u_{kj}^2 + \cdots z_p u_{kp}^2$

$o_k = \tau(o_{sum_k})$



(a) u_{kj}^2가 미치는 영향

$$\mathbf{U}^2 = \begin{pmatrix} u_{10}^2 & u_{11}^2 & \cdots & u_{1p}^2 \\ u_{20}^2 & u_{21}^2 & \cdots & u_{2p}^2 \\ \vdots & \vdots & \ddots & \vdots \\ u_{c0}^2 & u_{c1}^2 & \cdots & u_{cp}^2 \end{pmatrix}$$

shape: $(p+1) \times c$

$$J(\Theta) = \frac{1}{2}\|\mathbf{y} - \mathbf{o}(\Theta)\|_2^2$$

Gradient 계산 – 출력층

$$\frac{\partial J}{\partial u_{kj}^2} = \frac{\partial}{\partial u_{kj}^2}\left(\sum_{q=1}^{c}\frac{1}{2}(o_q - y_q)^2\right)$$

$q \neq k$ 인 것은 모두 상수이므로 미분에서 제외

$$= \frac{\partial}{\partial o_k}\left(\frac{1}{2}(o_k - y_k)^2\right)\frac{\partial o_k}{\partial u_{kj}^2}$$

$$= (o_k - y_k)\frac{\partial o_k}{\partial u_{kj}^2}$$

$$= (o_k - y_k)\underset{①}{\frac{\partial \tau(o_k)}{\partial o_{sum_k}}}\underset{②}{\frac{\partial o_{sum_k}}{\partial u_{kj}^2}}$$

$$= \underbrace{(o_k - y_k)\tau'(o_{sum_k})z_j}_{\triangleq \delta_k}$$

$$= \delta_k z_j$$

② $o_{sum_k} = z_0 u_{k0}^2 + z_1 u_{k1}^2 + \cdots + z_j u_{kj}^2 + \cdots z_p u_{kp}^2$

① $o_k = \tau(o_{sum_k})$

함수 이름	함수	1차 도함수
계단	$\tau(s) = \begin{cases} 1 & s \geq 0 \\ -1 & s < 0 \end{cases}$	$\tau'(s) = \begin{cases} 0 & s \neq 0 \\ \text{불가} & s = 0 \end{cases}$
로지스틱 시그모이드	$\tau(s) = \frac{1}{1+e^{-as}}$	$\tau'(s) = a\tau(s)(1-\tau(s))$
하이퍼볼릭 탄젠트	$\tau(s) = \frac{2}{1+e^{-as}} - 1$	$\tau'(s) = \frac{a}{2}(1-\tau(s)^2)$

Gradient 계산 – 은닉층

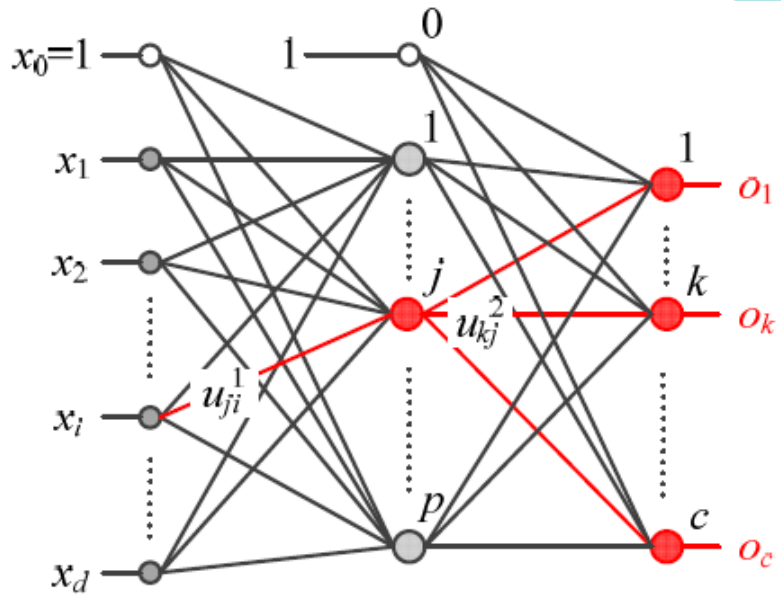
◆ $\frac{\partial J}{\partial \mathbf{U}^1}$: $\frac{\partial J}{\partial u_{ji}^1}$ $\quad 1 \le j \le p$, $0 \le i \le d$

$$\frac{\partial J}{\partial u_{ji}^1} = \eta_j x_i$$

where, $\eta_j = \tau'\left(z_{sum_j}\right)\sum_{q=1}^{c}\delta_q\, u_{qj}^2$

$$z_{sum_j} = x_0 u_{j0}^1 + x_1 u_{j1}^1 + \cdots + x_i u_{ji}^1 + \cdots x_d u_{pd}^1$$

$$z_j = \tau\left(z_{sum_j}\right)$$



(b) u_{ji}^1이 미치는 영향

$$\mathbf{U}^1 = \begin{pmatrix} u_{10}^1 & u_{11}^1 & \cdots & u_{1d}^1 \\ u_{20}^1 & u_{21}^1 & \cdots & u_{2d}^1 \\ \vdots & \vdots & \ddots & \vdots \\ u_{p0}^1 & u_{p1}^1 & \cdots & u_{pd}^1 \end{pmatrix}$$

shape: $p \times (d+1)$

$$J(\Theta) = \frac{1}{2}\|\mathbf{y} - \mathbf{o}(\Theta)\|_2^2$$

Gradient 계산 – 은닉층

$$\frac{\partial J}{\partial u_{ji}^1} = \frac{\partial}{\partial u_{ji}^1}\left(\sum_{q=1}^{c}\frac{1}{2}(o_q - y_q)^2\right)$$

$$= \frac{\partial}{\partial o_q}\left(\sum_{q=1}^{c}\frac{1}{2}(o_q - y_q)^2\right)\frac{\partial o_q}{\partial u_{ji}^1}$$

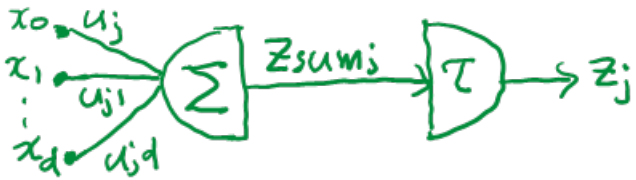
$$= \sum_{q=1}^{c}(o_q - y_q)\frac{\partial o_q}{\partial z_j}\frac{\partial z_j}{\partial u_{ji}^1}$$

$$= \sum_{q=1}^{c}(o_q - y_q)\,①\,\frac{\partial o_q}{\partial o_{sum_q}}\frac{\partial o_{sum_q}}{\partial z_j}\frac{\partial z_j}{\partial u_{ji}^1}$$

$$= \sum_{q=1}^{c}(o_q - y_q)\,\tau'\left(o_{sum_q}\right)\frac{\partial o_{sum_q}}{\partial z_j}\frac{\partial z_j}{\partial u_{ji}^1}$$

$$o_{sum_q} = z_0 u_{q0}^2 + z_1 u_{q1}^2 + \cdots + z_j u_{qj}^2 + \cdots z_p u_{qp}^2$$

① $o_q = \tau\left(o_{sum_q}\right)$



$$z_{sum_j} = x_0 u_{j0}^1 + x_1 u_{j1}^1 + \cdots + x_i u_{ji}^1 + \cdots x_d u_{pd}^1$$

$$z_j = \tau\left(z_{sum_j}\right)$$

Gradient 계산 – 은닉층

$$\frac{\partial J}{\partial u_{ji}^1} = \sum_{q=1}^{c} (o_q - y_q)\,\tau'\left(o_{sum_q}\right) \frac{\partial o_{sum_q}}{\partial z_j} \frac{\partial z_j}{\partial u_{ji}^1} \quad (②)$$

$$= \sum_{q=1}^{c} (o_q - y_q)\,\tau'\left(o_{sum_q}\right) u_{qj}^2 \frac{\partial z_j}{\partial u_{ji}^1}$$

$$= \sum_{q=1}^{c} (o_q - y_q)\,\tau'\left(o_{sum_q}\right) u_{qj}^2 \frac{\partial z_j}{\partial z_{sum_j}} \frac{\partial z_{sum_j}}{\partial u_{ji}^1} \quad (③, ④)$$

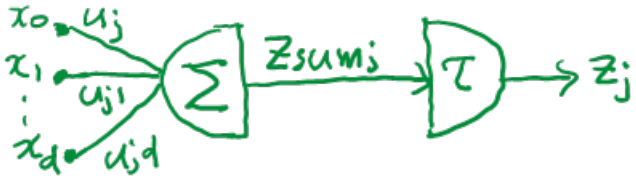
$$= \sum_{q=1}^{c} \underbrace{(o_q - y_q)\,\tau'\left(o_{sum_q}\right)}_{\triangleq \delta_q} u_{qj}^2 \tau'\left(z_{sum_j}\right) x_j$$

$$= \underbrace{\tau'\left(z_{sum_j}\right) \sum_{q=1}^{c} \delta_q\, u_{qj}^2}_{\triangleq \eta_j}\, x_j$$

$$= \eta_j x_i$$

② $o_{sum_q} = z_0 u_{q0}^2 + z_1 u_{q1}^2 + \cdots + z_j u_{qj}^2 + \cdots z_p u_{qp}^2$

$o_q = \tau\left(o_{sum_q}\right)$



④ $z_{sum_j} = x_0 u_{j0}^1 + x_1 u_{j1}^1 + \cdots + x_i u_{ji}^1 + \cdots x_d u_{pd}^1$

③ $z_j = \tau\left(z_{sum_j}\right)$

Gradient 계산 – Summary

- $\frac{\partial J}{\partial \mathbf{U}^2}$:

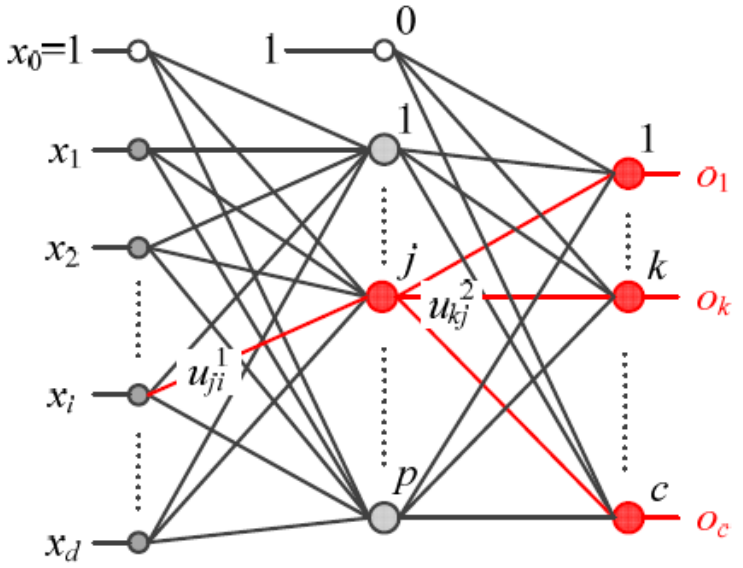
$$\delta_k = (o_k - y_k)\tau'(osum_k), \qquad 1 \le k \le c$$

$$\frac{\partial J}{\partial u_{kj}^2} = \Delta u_{kj}^2 = \delta_k z_j, \qquad 0 \le j \le p, 1 \le k \le c$$

- $\frac{\partial J}{\partial \mathbf{U}^1}$:

$$\eta_j = \tau'(zsum_j)\sum_{q=1}^{c}\delta_q u_{qj}^2, \qquad 1 \le j \le p$$

$$\frac{\partial J}{\partial u_{ji}^1} = \Delta u_{ji}^1 = \eta_j x_i, \qquad 0 \le i \le d, 1 \le j \le p$$



- _____ 알고리즘
  - 위 식들을 이용하여 출력층의 오류를 역방향(왼쪽)으로 전파하며 _____를 계산하는 알고리즘

Algorithm – SGD

- Feed forward

$$\tilde{\mathbf{z}} = \mathbf{U}^1\mathbf{x} \qquad \tilde{\mathbf{o}} = \mathbf{U}^2\mathbf{z}$$

- Back propagation

$$\delta_k = (o_k - y_k)\tau'(osum_k), \qquad 1 \le k \le c$$

$$\frac{\partial J}{\partial u_{kj}^2} = \Delta u_{kj}^2 = \delta_k z_j, \qquad 0 \le j \le p, 1 \le k \le c$$

$$\eta_j = \tau'(zsum_j)\sum_{q=1}^{c}\delta_q u_{qj}^2, \qquad 1 \le j \le p$$

$$\frac{\partial J}{\partial u_{ji}^1} = \Delta u_{ji}^1 = \eta_j x_i, \qquad 0 \le i \le d, 1 \le j \le p$$

- Parameter update

$$\mathbf{U}^1 = \mathbf{U}^1 - \rho\frac{\partial J}{\partial \mathbf{U}^1}$$

$$\mathbf{U}^2 = \mathbf{U}^2 - \rho\frac{\partial J}{\partial \mathbf{U}^2}$$

알고리즘 3-4 다층 퍼셉트론 학습을 위한 스토캐스틱 경사 하강법

입력: 훈련집합 $\mathbb{X}$와 $\mathbb{Y}$, 학습률 ρ
출력: 가중치 행렬 $\mathbf{U}^1$과 $\mathbf{U}^2$

1 $\mathbf{U}^1$과 $\mathbf{U}^2$를 초기화한다.
2 repeat
3 $\mathbb{X}$의 순서를 섞는다.
4 for ($\mathbb{X}$의 샘플 각각에 대해)
5 현재 처리하는 샘플을 $\mathbf{x} = (x_0, x_1, x_2, \cdots, x_d)^T$, $\mathbf{y} = (y_1, y_2, \cdots, y_c)^T$라 표기한다.
6 x_0과 z_0을 1로 설정한다. // 바이어스
// 전방 계산
7 for (j=1 to ρ) $zsum_j = \mathbf{u}_j^1\mathbf{x}$, $z_j = \tau(zsum_j)$ // 식 (3.13)
8 for (k=1 to c) $osum_k = \mathbf{u}_k^2\mathbf{z}$, $o_k = \tau(osum_k)$ // 식 (3.14)
// 오류 역전파
9 for (k=1 to c) $\delta_k = (o_k - y_k)\tau'(osum_k)$ // 식 (3.22)
10 for (k=1 to c) for (j=0 to ρ) $\Delta u_{kj}^2 = \delta_k z_j$ // 식 (3.23)
11 for (j=1 to ρ) $\eta_j = \tau'(zsum_j)\sum_{q=1}^{c}\delta_q u_{qj}^2$ // 식 (3.24)
12 for (j=1 to ρ) for (i=0 to d) $\Delta u_{ji}^1 = \eta_j x_i$ // 식 (3.25)
// 가중치 갱신
13 for (k=1 to c) for (j=0 to ρ) $u_{kj}^2 = u_{kj}^2 - \rho\Delta u_{kj}^2$ // 식 (3.21)
14 for (j=1 to ρ) for (i=0 to d) $u_{ji}^1 = u_{ji}^1 - \rho\Delta u_{ji}^1$ // 식 (3.21)
15 until (멈춤 조건)

Algorithm – SGD (cont'd)

- Matrix Version
  - GPU를 사용한 고속 연산에 적합

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

$$\mathbf{x}^{\mathrm{T}}\mathbf{y} = x_1y_1 + x_2y_2 + x_3y_3 = \sum_{i=1}^{3} x_iy_i$$

$$\mathbf{x} \odot \mathbf{y} = \begin{bmatrix} x_1 \times y_1 \\ x_2 \times y_2 \\ x_3 \times y_3 \end{bmatrix}$$

⊙ : Element-wise multiplication

알고리즘 3-5 다층 퍼셉트론 학습을 위한 스토캐스틱 경사 하강법(행렬 표기)

입력 : 훈련집합 $\mathbb{X}$와 $\mathbb{Y}$, 학습률 ρ
출력 : 가중치 행렬 $\mathbf{U}^1$과 $\mathbf{U}^2$

$\mathbf{U}^1$과 $\mathbf{U}^2$를 초기화한다.
repeat
  $\mathbb{X}$의 순서를 섞는다.
  for ($\mathbb{X}$의 샘플 각각에 대해)
    현재 처리하는 샘플을 $\mathbf{x} = (x_0, x_1, x_2, \cdots, x_d)^{\mathrm{T}}$, $\mathbf{y} = (y_1, y_2, \cdots, y_c)^{\mathrm{T}}$라 표기한다.
    x_0과 z_0을 1로 설정한다. // 바이어스
    // 전방 계산
    $\mathbf{zsum} = \mathbf{U}^1\mathbf{x}$, $\tilde{\mathbf{z}} = \boldsymbol{\tau}(\mathbf{zsum})$ // 식 (3.13), $\mathbf{zsum}_{p*1}$, ${\mathbf{U}^1}_{p*(d+1)}$, $\mathbf{x}_{(d+1)*1}$, $\tilde{\mathbf{z}}_{p*1}$
    $\mathbf{osum} = \mathbf{U}^2\mathbf{z}$, $\mathbf{o} = \boldsymbol{\tau}(\mathbf{osum})$ // 식 (3.14), $\mathbf{osum}_{c*1}$, ${\mathbf{U}^2}_{c*(p+1)}$, $\mathbf{z}_{(p+1)*1}$, $\mathbf{o}_{c*1}$
    // 오류 역전파
    $\boldsymbol{\delta} = (\mathbf{o} - \mathbf{y}) \odot \boldsymbol{\tau}'(\mathbf{osum})$ // 식 (3.22), $\boldsymbol{\delta}_{c*1}$
    $\Delta\mathbf{U}^2 = \boldsymbol{\delta}\mathbf{z}^{\mathrm{T}}$ // 식 (3.23), ${\Delta\mathbf{U}^2}_{c*(p+1)}$
    $\boldsymbol{\eta} = (\boldsymbol{\delta}^{\mathrm{T}}\tilde{\mathbf{U}}^2)^{\mathrm{T}} \odot \boldsymbol{\tau}'(\mathbf{zsum})$ // 식 (3.24), ${\tilde{\mathbf{U}}^2}_{c*p}$, $\boldsymbol{\eta}_{p*1}$
    $\Delta\mathbf{U}^1 = \boldsymbol{\eta}\mathbf{x}^{\mathrm{T}}$ // 식 (3.25), ${\Delta\mathbf{U}^1}_{p*(d+1)}$
    // 가중치 갱신
    $\mathbf{U}^2 = \mathbf{U}^2 - \rho\Delta\mathbf{U}^2$ // 식 (3.21)
    $\mathbf{U}^1 = \mathbf{U}^1 - \rho\Delta\mathbf{U}^1$ // 식 (3.21)
until (멈춤 조건)

Algorithm – Minibatch SGD

- *Minibatch* Mode
  - 한번에 t개의 샘플을 처리
    - sample = training example
  - t를 _____라고 함
    - t = 수십~수백
    - Cf.) $t = 1$: SGD
    - Cf.) $t = n$: (Full) Batch GD
  - Gradient의 잡음을 줄여 수렴이 빨라짐
  - GPU를 사용한 병렬처리에도 유리
  - 현대 기계 학습은 미니배치를 널리 사용

알고리즘 3-6 다층 퍼셉트론 학습을 위한 '미니배치' 스토캐스틱 경사 하강법

입력: 훈련집합 $\mathbb{X}$와 $\mathbb{Y}$, 학습률 ρ, 미니배치 크기 t
출력: 가중치 행렬 $\mathbf{U}^1$과 $\mathbf{U}^2$

1 $\mathbf{U}^1$과 $\mathbf{U}^2$를 초기화한다.
2 repeat
3 $\mathbb{X}$와 $\mathbb{Y}$에서 t개의 샘플을 무작위로 뽑아 미니배치 $\mathbb{X}'$와 $\mathbb{Y}'$를 만든다.
4 $\Delta\mathbf{U}^2 = \mathbf{0}$, $\Delta\mathbf{U}^1 = \mathbf{0}$
5 for ($\mathbb{X}'$의 샘플 각각에 대해)
6 현재 처리하는 샘플을 $\mathbf{x} = (x_0, x_1, x_2, \cdots, x_d)^T$, $\mathbf{y} = (y_1, y_2, \cdots, y_c)^T$라 표기한다.
7 x_0와 z_0를 1로 설정한다. // 바이어스
  // 전방 계산
8 $\mathbf{zsum} = \mathbf{U}^1\mathbf{x}$, $\tilde{\mathbf{z}} = \boldsymbol{\tau}(\mathbf{zsum})$ // 식 (3.13), $\mathbf{zsum}_{p*1}$, ${\mathbf{U}^1}_{p*(d+1)}$, $\mathbf{x}_{(d+1)*1}$, $\tilde{\mathbf{z}}_{p*1}$
9 $\mathbf{osum} = \mathbf{U}^2\mathbf{z}$, $\mathbf{o} = \boldsymbol{\tau}(\mathbf{osum})$ // 식 (3.14), $\mathbf{osum}_{c*1}$, ${\mathbf{U}^2}_{c*(p+1)}$, $\mathbf{z}_{(p+1)*1}$, $\mathbf{o}_{c*1}$
  // 오류 역전파
10 $\boldsymbol{\delta} = (\mathbf{o} - \mathbf{y}) \odot \boldsymbol{\tau}'(\mathbf{osum})$ // 식 (3.22), $\boldsymbol{\delta}_{c*1}$
11 $\Delta\mathbf{U}^2 = \Delta\mathbf{U}^2 + (\ \boldsymbol{\delta}\mathbf{z}^T)$ // 식 (3.23)을 누적, ${\Delta\mathbf{U}^2}_{c*(p+1)}$
12 $\boldsymbol{\eta} = (\boldsymbol{\delta}^T\tilde{\mathbf{U}}^2)^T \odot \boldsymbol{\tau}'(\mathbf{zsum})$ // 식 (3.24), ${\tilde{\mathbf{U}}^2}_{c*p}$, $\boldsymbol{\eta}_{p*1}$
13 $\Delta\mathbf{U}^1 = \Delta\mathbf{U}^1 + (\ \boldsymbol{\eta}\mathbf{x}^T)$ // 식 (3.25)를 누적, ${\Delta\mathbf{U}^1}_{p*(d+1)}$
  // 가중치 갱신
14 $\mathbf{U}^2 = \mathbf{U}^2 - \rho\left(\frac{1}{t}\right)\Delta\mathbf{U}^2$ // 식 (3.21) - 평균 그레이디언트로 갱신
15 $\mathbf{U}^1 = \mathbf{U}^1 - \rho\left(\frac{1}{t}\right)\Delta\mathbf{U}^1$ // 식 (3.21) - 평균 그레이디언트로 갱신
16 until (멈춤 조건)

Testing (Prediction)

◆ 학습을 마친 후 성능 검증 (혹은 현장 설치하여 사용)

- Feed-forward (전방계산)만 하면 됨

알고리즘 3-7 다층 퍼셉트론을 이용한 인식

입력: 테스트 샘플 x // 신경망의 가중치 $\mathbf{U}^1$과 $\mathbf{U}^2$는 이미 설정되었다고 가정함.
출력: 부류 y

1. $\mathbf{x} = (x_0, x_1, x_2, \cdots, x_d)^{\mathrm{T}}$로 확장하고, x_0과 z_0을 1로 설정한다.
2. $\mathbf{zsum} = \mathbf{U}^1\mathbf{x}$
3. $\tilde{\mathbf{z}} = \boldsymbol{\tau}(\mathbf{zsum})$ // 식 (3.13)
4. $\mathbf{osum} = \mathbf{U}^2\mathbf{z}$
5. $\mathbf{o} = \boldsymbol{\tau}(\mathbf{osum})$ // 식 (3.14)
6. $\mathbf{o}$에서 가장 큰 값을 가지는 노드에 해당하는 부류 번호를 y에 대입한다.

$$y = \underset{k}{\operatorname{argmax}}\, o_k$$

Example – Binary Classification with 2-Layer Perceptron

◆ Architecture

- input (**x**) dimension: d=2
- output (**o**) dimension: m=2
- # layers: L=2
- parameters: u and v

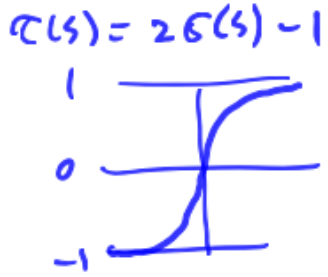
◆ Activation function

- *bipolar sigmoid* (*hyperbolic tangent*)
  - let a=1

$$\tau(s) = \frac{2}{1+e^{-as}} - 1$$

$$\tau'(s) = \frac{a}{2}(1-\tau(s)^2)$$

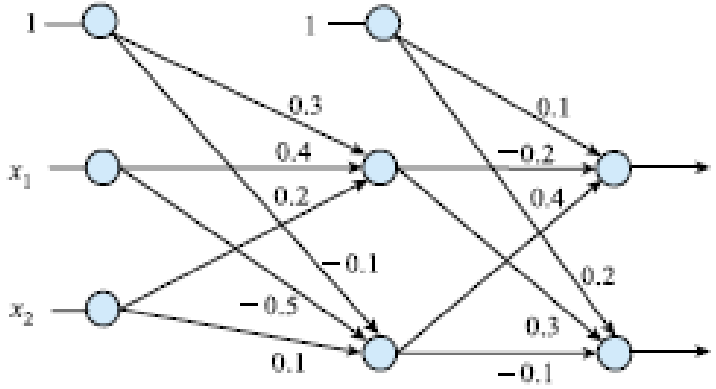
Logistic sigmoid



알고리즘 3-5 예제 (슬라이드 p.30)

그림 4.13은 $d=2$, $p=2$, 그리고 $m=2$인 아키텍처를 가진 다층 퍼셉트론이다. 가중치는 그림에서처럼 초기화되어 있다고 하자. 활성 함수로 $\alpha=1$인 양극 시그모이드를 사용하고 학습률은 $\rho=0.2$라 한다. 아래 샘플을 가지고 알고리즘 [4.5]의 학습 과정을 살펴보자.

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



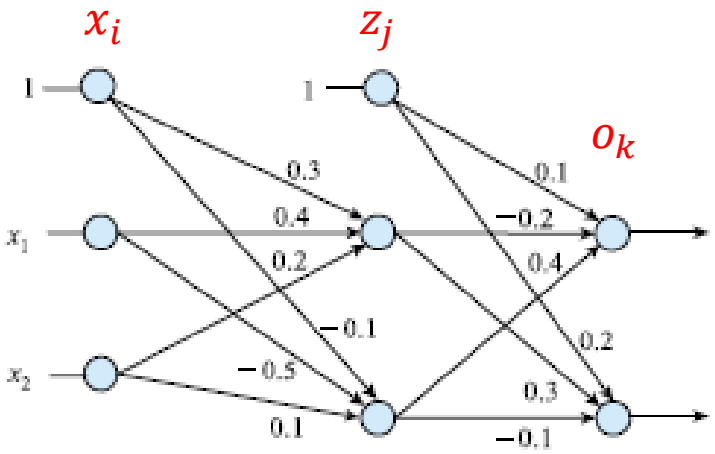
Example – Binary Classification (cont'd)

◆ Feed Forward (전방 계산)

for (j=1 to p) $zsum_j = \mathbf{u}_j^1\mathbf{x}$, $z_j = \tau(zsum_j)$

for (k=1 to c) $osum_k = \mathbf{u}_k^2\mathbf{z}$, $o_k = \tau(osum_k)$

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



라인 7:

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

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

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

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

라인 8:

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

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

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

$o_2 = \tau_2(0.31131) = 0.15441$

이 다층 퍼셉트론은 입력 $\mathbf{x} = (0.7,0.2)^T$에 대해 $\mathbf{o} = (-0.02239,0.15441)^T$을 출력하였다. 기대하는 값 $\mathbf{t} = (-1,1)^T$과의 오류는 아래와 같이 계산할 수 있다.

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

Example – Binary Classification (cont'd)

◆ Error Back Propagation (오류 역전파)

for (k=1 to c) $\delta_k = (o_k - y_k)\tau'(osum_k)$

for (k=1 to c) for (j=0 to p) $\Delta u_{kj}^2 = \delta_k z_j$

라인 9:

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

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

라인 10:

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

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

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

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

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

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

for (j=1 to p) $\eta_j = \tau'(zsum_j) \sum_{q=1}^{c} \delta_q u_{qj}^2$

for (j=1 to p) for (i=0 to d) $\Delta u_{ji}^1 = \eta_j x_i$

라인 11:

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

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

라인 12:

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

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

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

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

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

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

Example – Binary Classification (cont'd)

◆ Parameter Update (가중치 갱신)

for (k=1 to c) for (j=0 to p) $u_{kj}^2 = u_{kj}^2 - \rho \Delta u_{kj}^2$

for (j=1 to p) for (i=0 to d) $u_{ji}^1 = u_{ji}^1 - \rho \Delta u_{ji}^1$

라인 13:

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

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

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

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

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

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

라인 14:

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

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

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

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

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

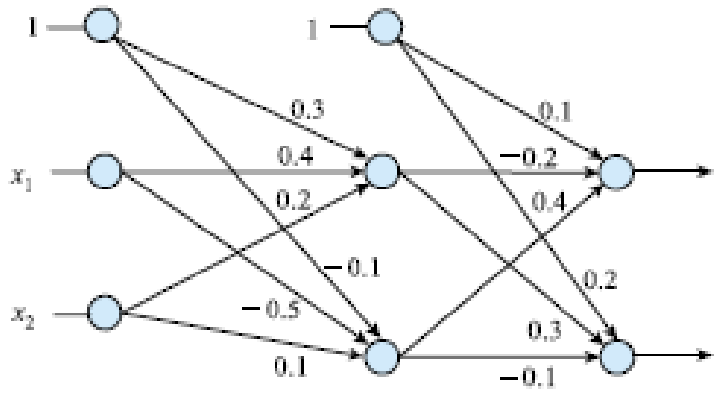
$u_{22} = 0.1 - 0.00452 = 0.09548$

Example – Binary Classification (cont'd)

◆ Evaluating the Updated Results

- Prediction vs. Target
- Error (MSE)

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



이 예제를 마치기 전에 학습한 효과를 확인해 보자. 이 작업은 새로 얻은 **u**와 **v**가 좋아졌는지를 확인하는 것이다. 라인 6과 라인 7로 전방 계산을 해보자.

라인 7과 8:

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

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

$z_1 = 0.31440$

$z_2 = -0.22821$

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

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

$o_1 = -0.08272$

$o_2 = 0.20288$

$\mathbf{o} = (-0.08272, 0.20288)^T$을 얻어 우리가 원하는 $\mathbf{t} = (-1,1)^T$에 가까워졌음을 알 수 있다. 오류도 $E = 0.73840$이 되어 이전보다 줄어들었음을 확인할 수 있다. ■■■

3. Remarks on MLP

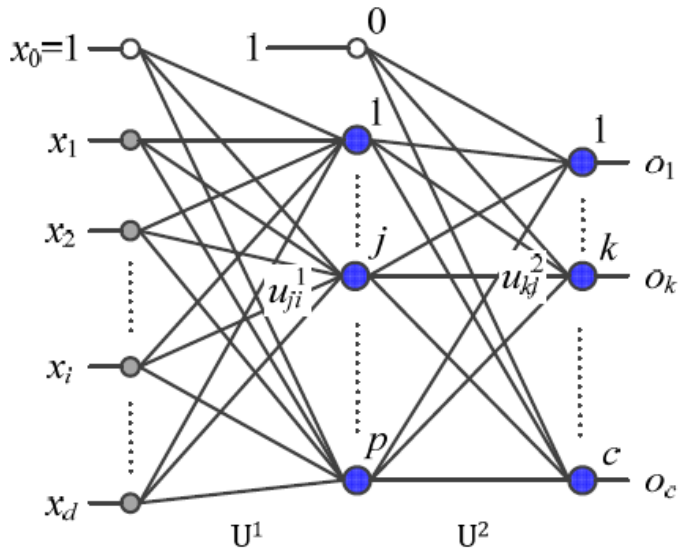
Universal Approximator

◆ 호닉의 주장 [Hornik1989]

- 은닉층을 하나만 가진 다층 퍼셉트론은 _____ *universal approximator*

  "… standard multilayer feedforward network architectures using arbitrary squashing functions can approximate virtually any function of interest to any desired degree of accuracy, provided sufficiently many hidden units are available. … 은닉 노드가 충분히 많다면, 포화함수(활성함수)로 무엇을 사용하든 표준 다층 퍼셉트론은 어떤 함수라도 원하는 정확도만큼 근사화할 수 있다."

- 실질적으로는 은닉 노드를 무수히 많게 할 수 없으므로, 복잡한 구조의 데이터에서는 성능 한계 발생



Practical Considerations

- 순수한 최적화 알고리즘만으로 높은 성능을 얻기 쉽지 않음
  - 데이터 희소성, 잡음, 미숙한 신경망 구조 등의 이유
- 성능 향상을 위한 Heuristics가 필요
  - 예)『*Neural Networks: Tricks of the Trade*』[Montavon2012]

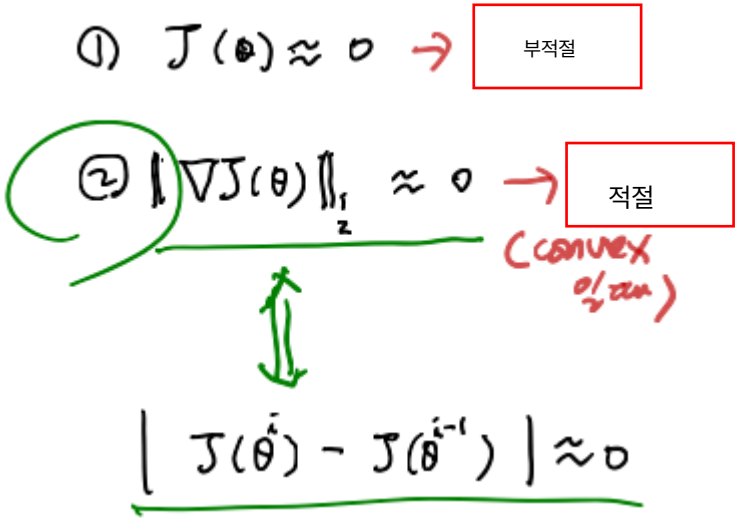
- 아키텍처: 은닉층과 은닉 노드의 개수를 정해야 한다. 은닉층과 은닉 노드를 늘리면 신경망의 용량은 커지는 대신, 추정할 매개변수가 많아지고 학습 과정에서 과잉적합할 가능성이 커진다. 1.6절에서 소개한 바와 같이 현대 기계 학습은 복잡한 모델을 사용하되, 적절한 규제 기법을 적용하는 경향이 있다.
- 초깃값 [알고리즘 3–4]의 라인 1에서 가중치를 초기화한다. 보통 난수를 생성하여 설정하는데, 값의 범위와 분포가 중요하다. 이 주제는 5.2.2절에서 다룬다.
- 학습률 처음부터 끝까지 같은 학습률을 사용하는 방식과 처음에는 큰 값으로 시작하고 점점 줄이는 적응적 방식이 있다. 5.2.4절에서 여러 가지 적응적 학습률 기법을 소개한다.
- 활성함수 초창기 다층 퍼셉트론은 주로 로지스틱 시그모이드나 tanh 함수를 사용했는데, 은닉층의 개수를 늘림에 따라 그레이디언트 소멸과 같은 몇 가지 문제가 발생한다. 따라서 깊은 신경망은 주로 ReLU 함수를 사용한다. 5.2.5절에서 여러 가지 ReLU 함수를 설명한다.

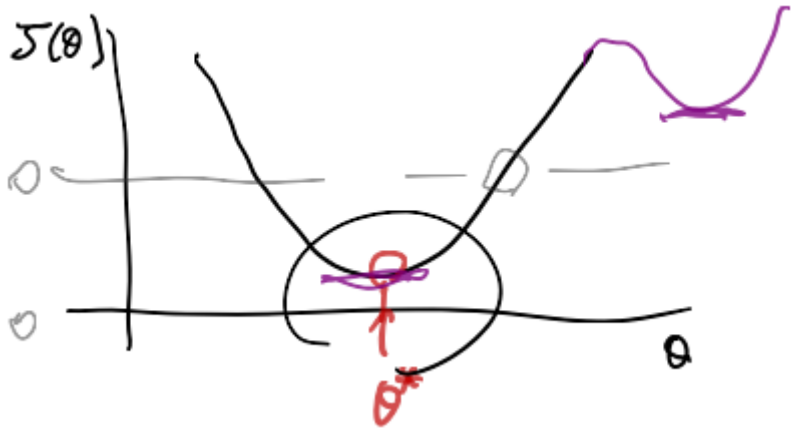
Practical Considerations (cont'd)

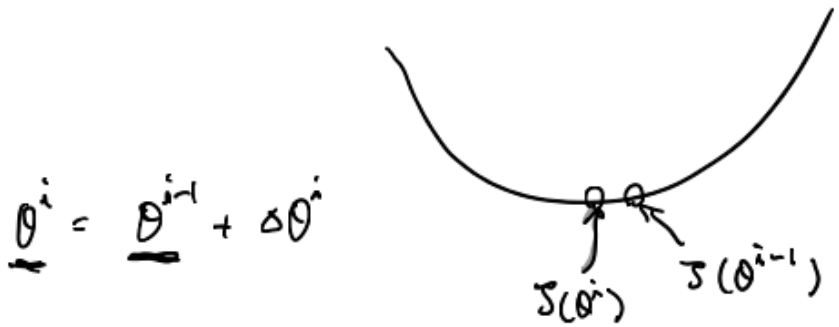
- Activation Functions & Vanishing Gradient
  - MLP에서 연성 활성함수가 도입됨
    - 영역을 영역으로 변환 가능
    - (Hard threshold와 달리) 미분값이 폭넓은 영역에 대해 존재 → **오류역전파 가능**
  - 그러나 **깊은 신경망**의 경우 sigmoid와 같은 활성함수는 그래디언트 소멸을 유발 → **ReLU**가 도입됨

Practical Considerations (cont'd)

- Stop Condition (멈춤조건)
  - Cost의 크기 vs. Cost Gradient의 크기







Practical Considerations (cont'd)

- Early Stopping (조기멈춤)
  - 훈련에 사용하지 않을 검증집합 _____ 을 별도로 구성
  - 검증집합의 성능이 최소일 때 멈춤
  - 모델 용량이 커서 과적합overfitting할 가능성이 높을 경우 유용

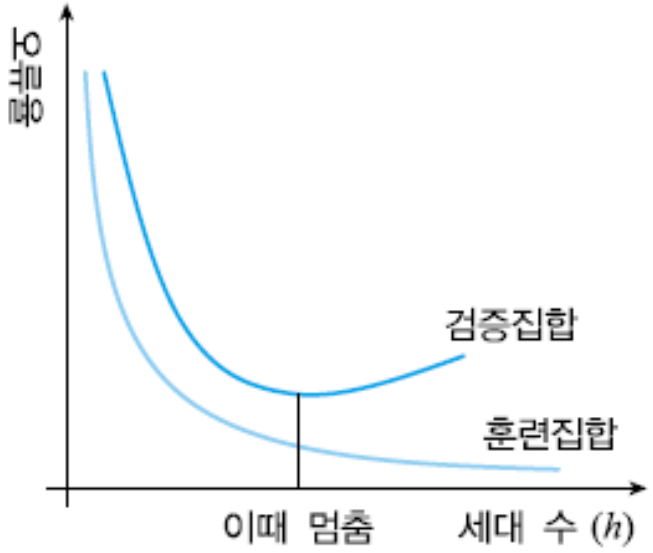


그림 4.15 일반화 기준에 따른 멈춤 조건

MLP to DNN (Deep Neural Networks)

- MLP의 실용적 성능
  - 1980~1990년대에 MLP는 실용 시스템 제작에 크게 기여
    - 인쇄/필기 문자 인식으로 우편물 자동 분류기, 전표 인식기, 자동차 번호판 인식기 등
    - 음성 인식, 게임, 주가 예측, 정보 검색, 의료 진단, 유전자 검색, 반도체 결함 검사 등

- MLP의 한계
  - 잡음이 섞인 상황에서 음성인식 성능 저하
  - 필기 주소 인식 능력 저하
  - 바둑에서의 한계

- Deep Learning (DNN)은 이들 한계를 극복함

감사합니다.