[SWCON253] Machine Learning – Lec.10a

Neural Networks (MLP)

Fall 2025

김휘용

hykim.v@khu.ac.kr

경희대학교
KYUNG HEE UNIVERSITY

Contents

References

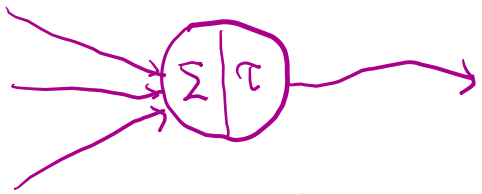
- *기계 학습* by 오일석, *패턴 인식* by 오일석

1. Neural Networks

- ✓ Human Neuron
- ✓ Brief History
- ✓ Human Brain vs. GPU
- ✓ Types of NN

Neural Networks – *Human Neuron*

- 사람의 뉴런
  - 두뇌의 가장 작은 정보처리 단위
  - 수상돌기는dendrite 신호 수신
  - 세포체는cell body 간단한 연산
  - 축삭(축색)은axon 처리 결과를 전송



인공 뉴런

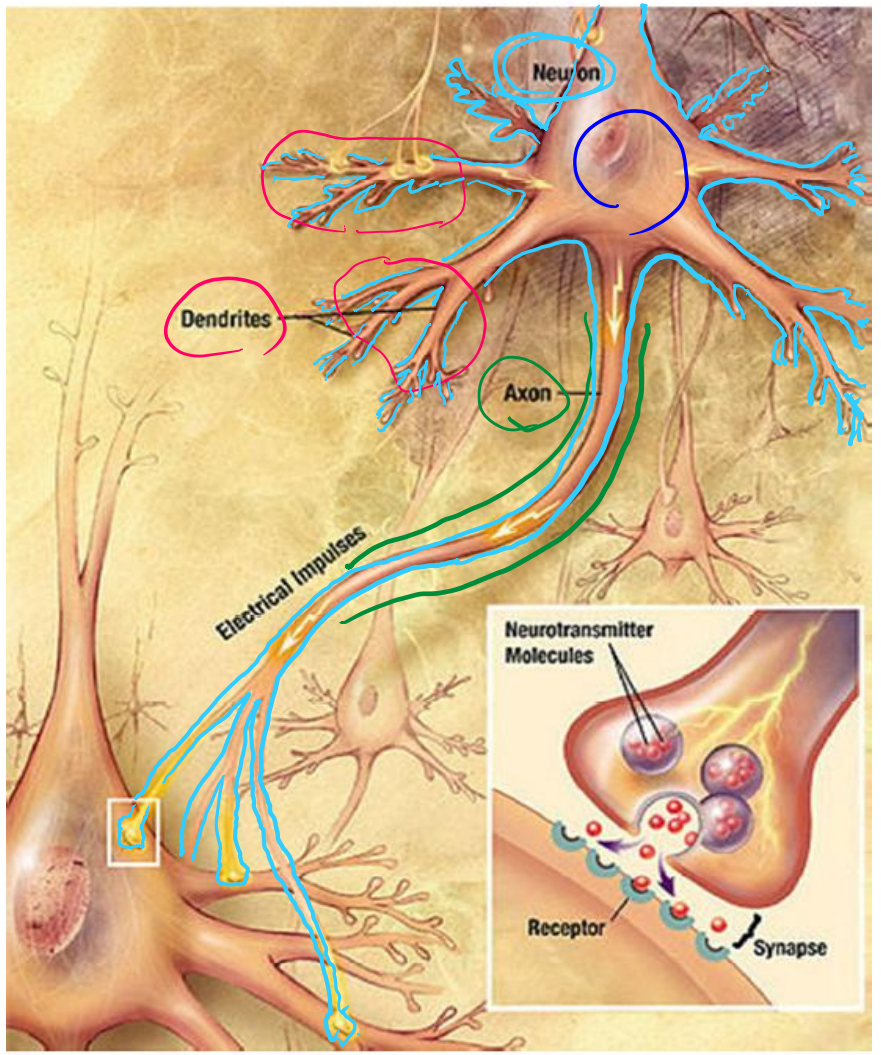


그림 3–1 사람의 뉴런의 구조와 동작

Neural Networks – *Brief History*

- (인공)신경망의 간략한 역사
  - 1943년 McCulloch과 Pitts가 최초의 신경망 제안
  - 1949년 Hebb는 최초로 학습 알고리즘 제안
  - 1958년 Rosenblatt은 **퍼셉트론** 제안
  - Widrow와 Hoff의 Adaline 과 Madaline
  - 1960년대의 과대 평가
  - 1969년 Minsky와 Papert의 저서『Perceptrons』는 퍼셉트론의 한계를 지적(퍼셉트론은 선형분류기에 불과하여 XOR 문제조차 해결 못함)하고, 다층 구조를 이용한 극복 방안 제시. 그러나 당시 기술로는 실현 불가능
  - 1974년 웨어보스는 박사 논문에서 오류 역전파 알고리즘 제안 ⇒ 신경망 연구 퇴조
  - 1986년 Rumelhart, Hinton, Williams는『Parallel Distributed Processing』에서 **다층 퍼셉트론**과 **오류 역전파** 학습 알고리즘 제안 ⇒ 신경망 연구 부활
  - 1990년대 **SVM**에 밀리는 형국
  - 2000년대 **딥러닝**이 실현되어 신경망이 기계 학습의 주류 기술로 자리매김
  - 보다 상세한 신경망 역사는 [**Kurenkov2015**] 참조

Neural Networks – *Human Brain vs. GPU*

Image Source: https://timedotcom.files.wordpress.com/2014/05/brain.jpg?w=1100&quality=85

Brain with 1.6 x 10^{10} neurons

10^4 -10^5 connections per neuron

approx. 10^{15} connections in total

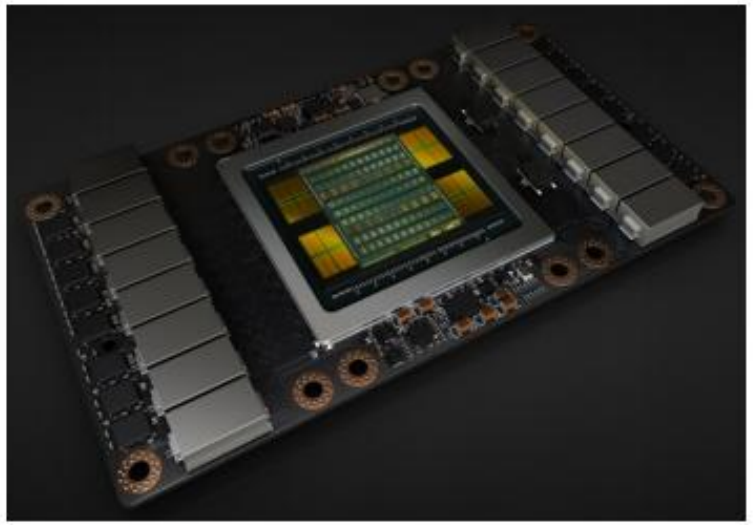


Image Source: https://fossbytes.com/wp-content/uploads/2017/05/nvidia-volta-v100-gpu.jpg

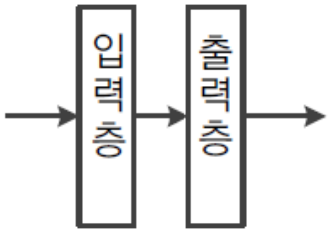
NVIDIA Volta with approx. 2.1 x 10^{10} transistors

approx. only 10 connections per transistor

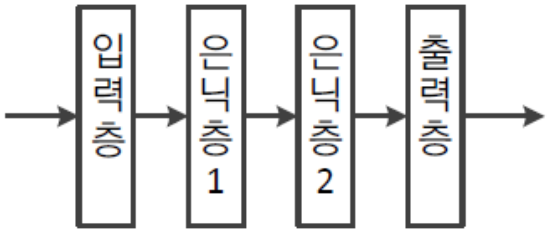
Neural Networks – *Types of NN*

단층 신경망, 다층 신경망

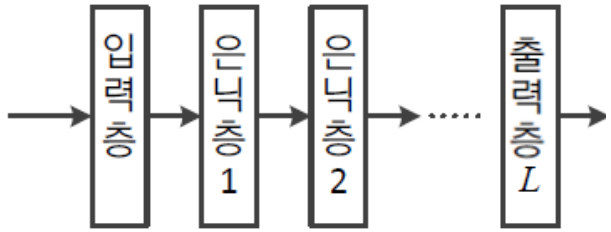
Single-layer NN



Multi-Layer NN (e.g., Multi-Layer Perceptron)

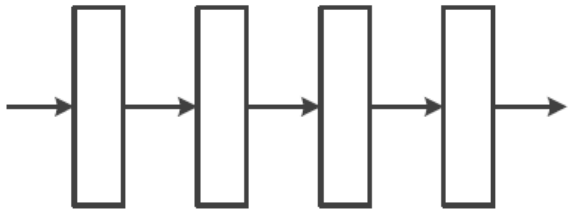


3-layer NN (Shallow)



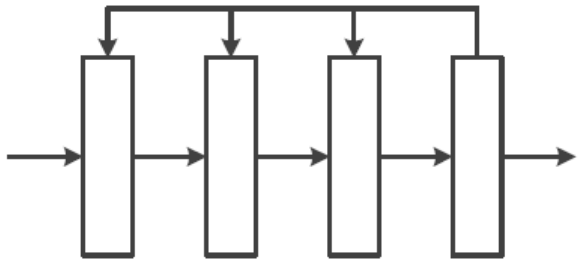
Deep NN ($L \geq 4$)

전방 신경망, 양방향 신경망, 잔차 신경망



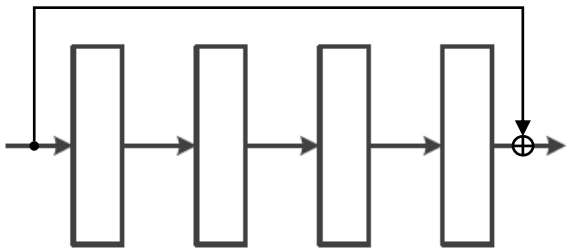
Feed-Forward NN

Feed-Back Path



Bidirectional NN

Skip Connection



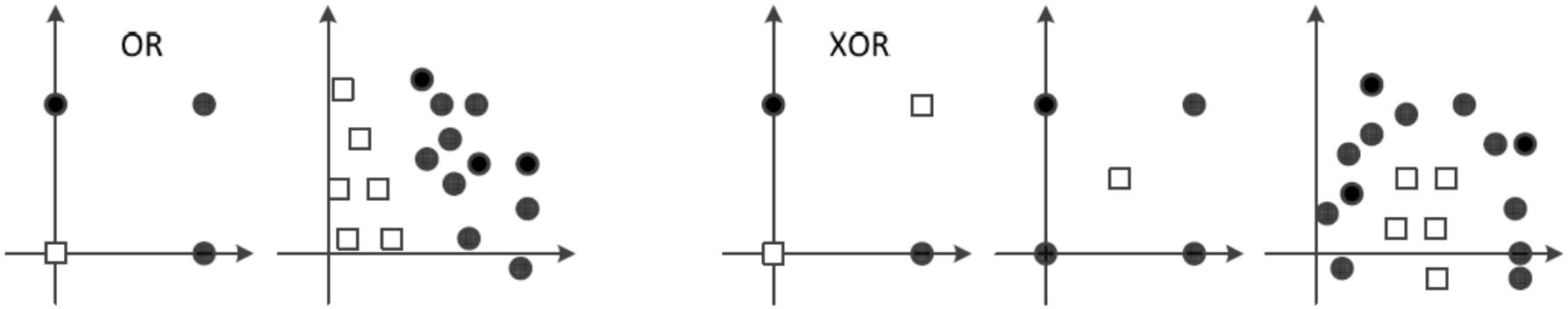
Residual NN

2. Multi-Layer Perceptron

- ✓ Perceptron의 한계 및 MLP로의 발전
- ✓ 은닉층을 이용한 특징 공간 변환
- ✓ 연성 활성함수의 도입
- ✓ MLP Architecture
- ✓ Forward Computation (전방 계산)

Perceptron의 한계 및 MLP로의 발전

- Perceptron의 한계: 선형 분리 불가능한 상황에서는 일정한 양의 오류 발생
  - 예) XOR 문제에서는 75%가 정확률 한계



(a) 선형분리 가능 (b) 선형분리 불가능

Perceptron의 한계 및 MLP로의 발전 (cont'd)

- MLP의 **핵심 아이디어**

1. **은닉층**(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. **시그모이드 활성함수**(sigmoid activation function)의 도입

★ 계단 활성함수: 경성 의사결정 (hard decision)
→ 출력값이 곧 의사결정(분류) 결과

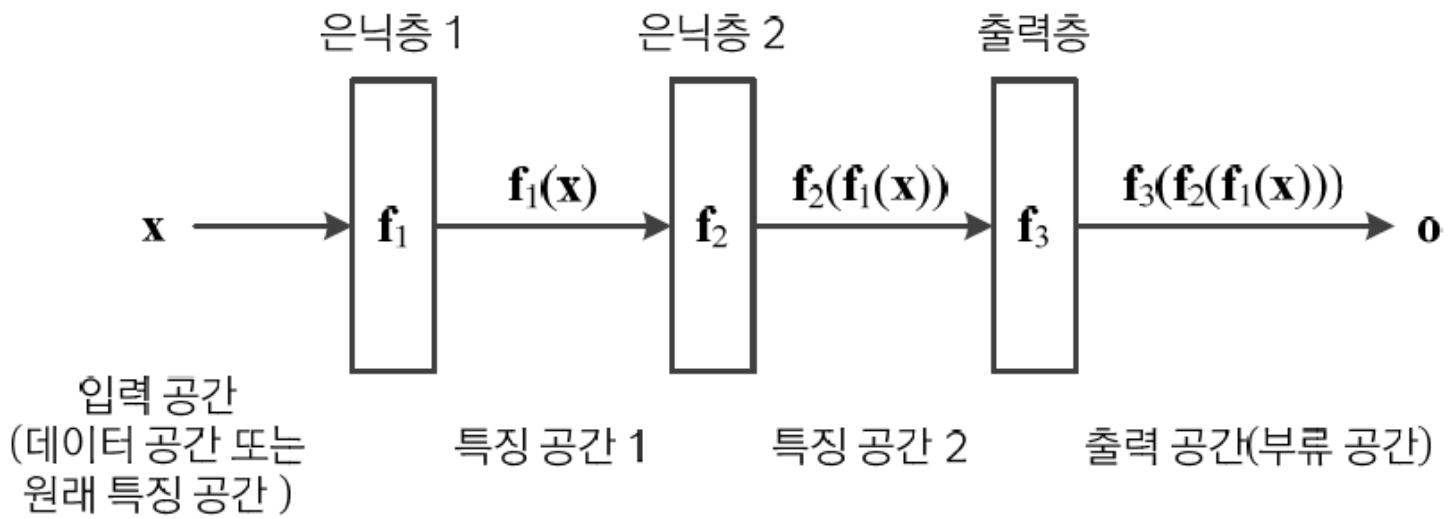
★ 시그모이드 활성함수: **연성 의사결정** (soft decision)
→ 출력값을 신뢰도(확률)로 간주하여 융통성 있는 의사결정 가능 (Logistic Regression과 동일)

3. **오류 역전파**(**back-propagation**) 알고리즘 사용

★ 다층 신경망의 학습이 어려웠던 문제를 해결

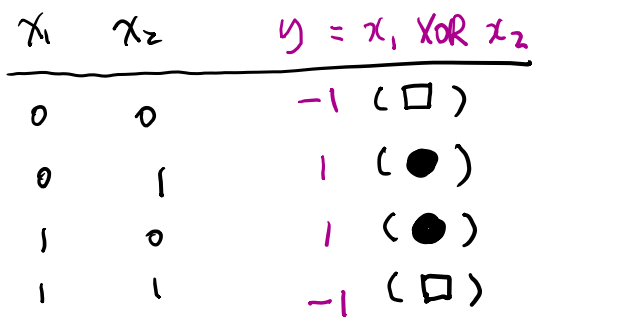
1. 은닉층을 이용한 특징 공간 변환

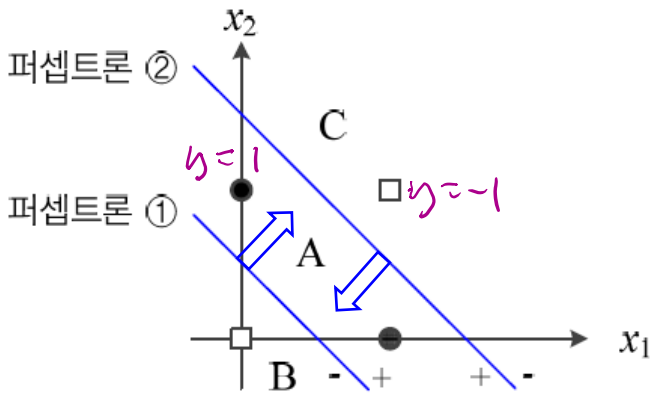
- 은닉층의 역할: 특징 추출 (특징 학습)
  - 은닉층은 특징 벡터를 (원하는 작업(예: 분류)에 더 유리한) 새로운 특징 공간으로 변환
  - 현대 기계 학습에서는 **특징 학습**이라*feature learning* 부름 (딥러닝은 더 많은 층을 거쳐 특징 학습을 함)

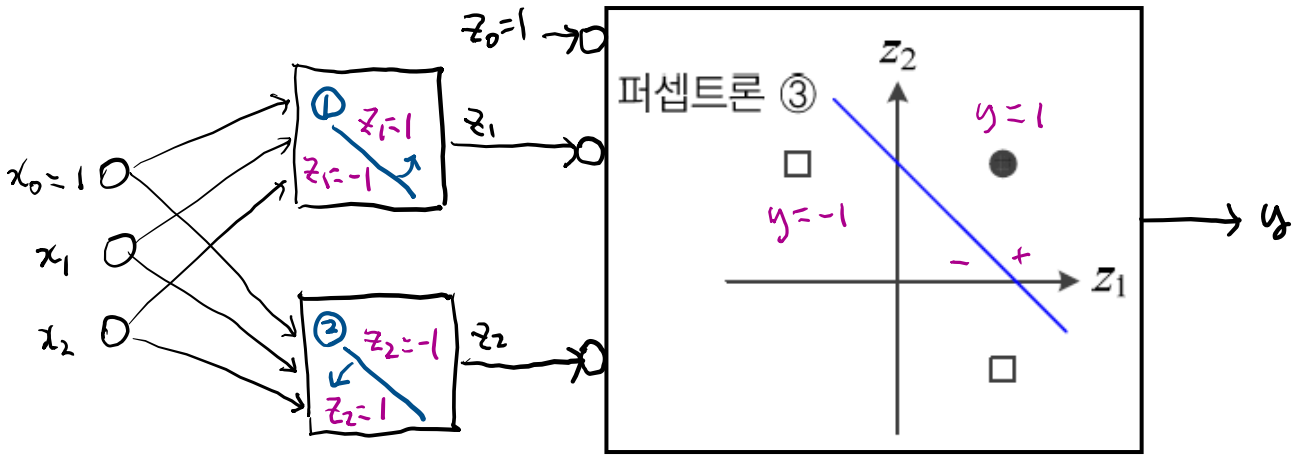


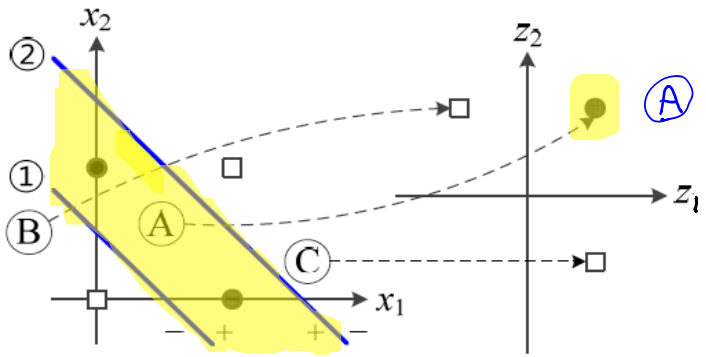
1. 은닉층을 이용한 특징 공간 변환 – 예시

- **선형 결정 경계 (즉, 퍼셉트론) 여러 개를 사용하면 XOR 문제 해결이 가능**
  1. 두 개의 퍼셉트론을 이용하여 원래 특징 공간 $\mathbf{x} = (x_1, x_2)^{\mathrm{T}}$를 새로운 **특징 공간** $\mathbf{z} = (z_1, z_2)^{\mathrm{T}}$로 **변환**
  2. 하나의 퍼셉트론을 이용하여 새로운 특징 공간 $\mathbf{z}$에서 **선형 분리**를 수행





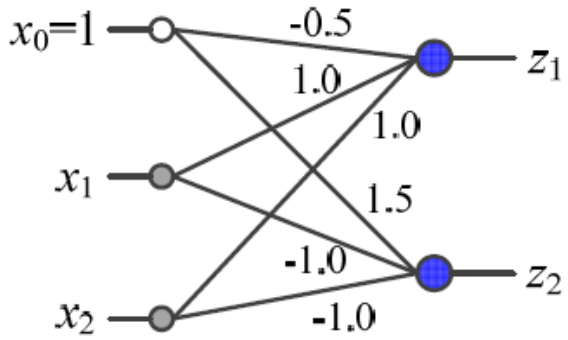




(b) 원래 특징 공간 x를 새로운 특징 공간 z로 변환

1. 은닉층을 이용한 특징 공간 변환 – 예시 (cont'd)

Step 1) Perceptron 2개를 병렬로 결합하여
$\mathbf{x} = (x_1, x_2)^{\mathrm{T}}$를 $\mathbf{z} = (z_1, z_2)^{\mathrm{T}}$로 **변환**



(a) 두 퍼셉트론을 병렬로 결합

$x \rightarrow \Sigma \xrightarrow{z_s} \tau \rightarrow z$

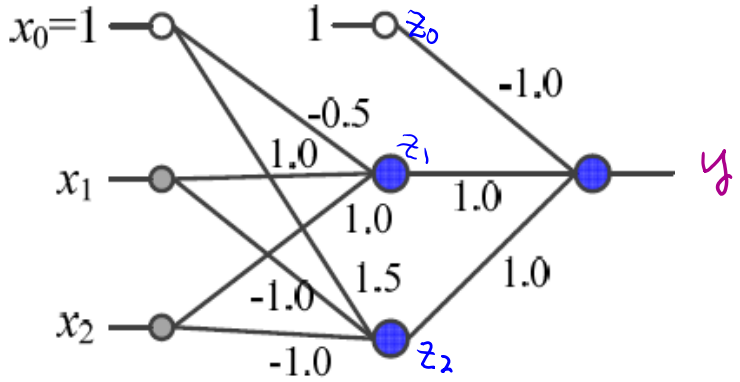
① : $x_2 > -x_1 + 0.5$

$\Rightarrow$ $z_{s1} = -0.5 + x_1 + x_2 > 0$

② : $x_2 < -x_1 + 1.5$

$\Rightarrow$ $z_{s2} = 1.5 - x_1 - x_2 > 0$

Step 2) Perceptron 1개를 순차로 결합하여
새로운 특징 공간 $\mathbf{z}$에서 **선형 분리**를 수행



(b) 퍼셉트론 3개를 결합한 다층 퍼셉트론

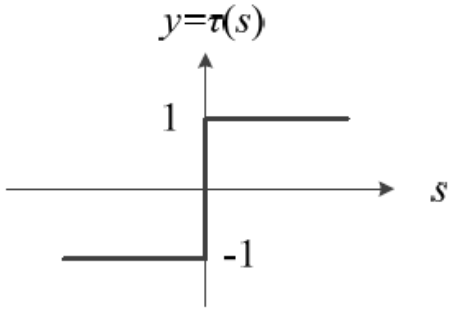
$\begin{cases} z_1 > 0 \text{ 이고 } z_2 > 0 \text{ 이면,} & \bullet \\ \text{otherwise,} & \square \end{cases}$

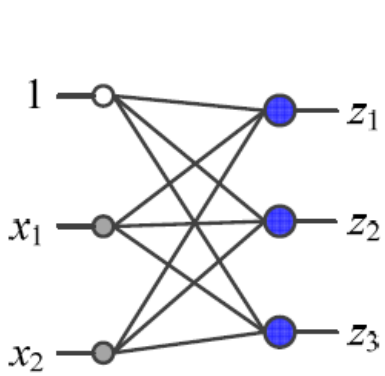
$\Rightarrow \begin{cases} y_s = -1 + z_1 + z_2 > 0 \text{ 이면,} & \bullet \ (y=1) \\ \text{otherwise,} & \square \ (y=-1) \end{cases}$

1. 은닉층을 이용한 특징 공간 변환 (cont'd)

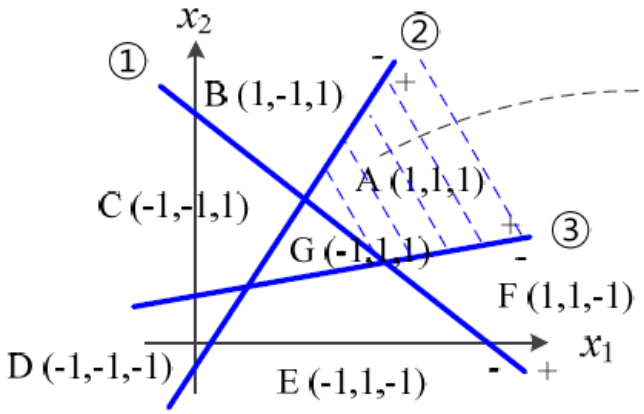
(참고) 퍼셉트론의 개수와 특징 공간의 변환

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

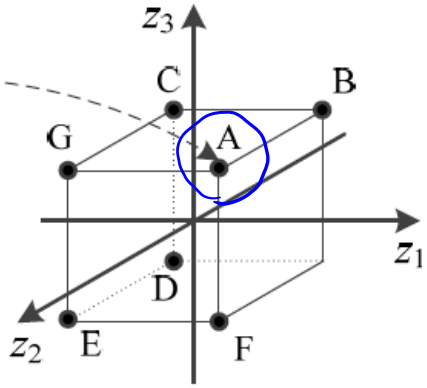




(a) 퍼셉트론 3개를 결합



(b) 7개 부분공간으로 나눔



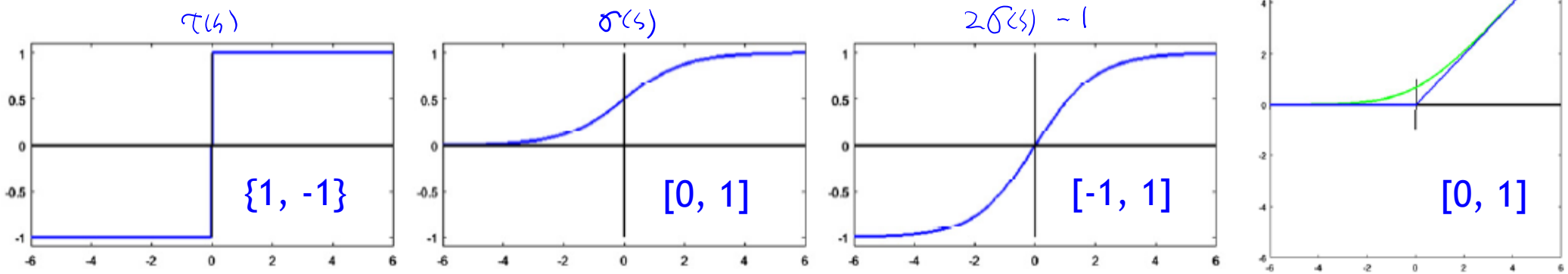
(c) 3차원 공간의 점으로 매핑

- 이를 일반화 하면, p개 퍼셉트론을 결합하여 p차원 공간으로의 변환이 가능함을 알 수 있다.

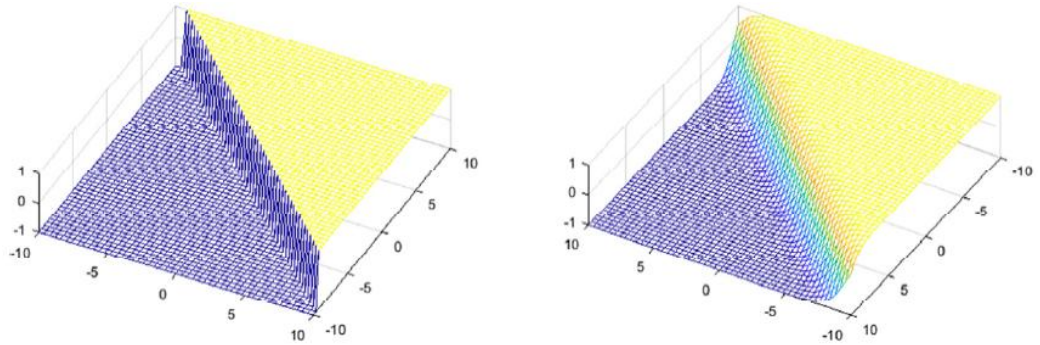
★ Cf.) 부분공간의 개수 = $1 + \sum_{i=1}^{p} i$

2. 연성 활성함수의 도입

- **연성 활성함수(*Soft Activation Function*)에 의한 부드러운 공간 분할**
  - 계단함수는 경성 의사결정(영역을 점으로 변환)
  - **연성 활성함수를 통해 부드러운 의사결정(영역을 영역으로 변환) 가능**



(a) 계단 함수 (b) 로지스틱 시그모이드 (c) 하이퍼볼릭 탄젠트 시그모이드 (d) softplus와 rectifier (ReLU)



(a) 계단함수의 딱딱한 공간 분할 (b) 로지스틱 시그모이드의 부드러운 공간 분할

2. 연성 활성함수의 도입 (cont'd)

◆ 신경망이 사용하는 다양한 활성함수

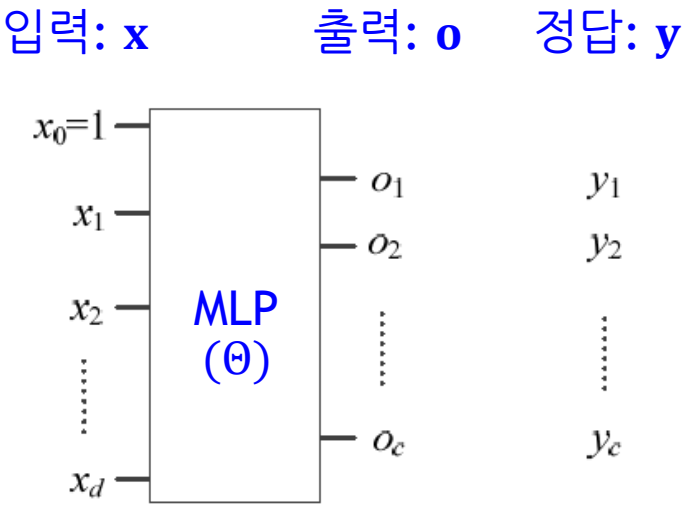
- Perceptron은 계단함수, MLP는 로지스틱 시그모이드와 하이퍼볼릭 탄젠트, DNN은 ReLU를 주로 사용
- 로지스틱 시그모이드와 하이퍼볼릭 탄젠트는 a가 커질수록 계단함수에 가까워짐
- 모두 1차 도함수 계산이 빠름 (특히 ReLU는 비교 연산 한 번)

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, ∞)

3. MLP Architecture

◆ 입력층과 출력층

- 입력층과 출력층 노드(neuron)의 개수는 주어진 문제에 의해 정해짐
  - ★ 입력 특징의 개수: d → 입력층 노드 개수: d+1 ($\mathbf{x} = [x_0\ x_1 \ldots x_d]^{\mathrm{T}}$)
  - ★ 정답 벡터의 차원: c → 출력층 노드 개수: c ($\mathbf{y} = [y_1 \ldots y_c]^{\mathrm{T}}, \mathbf{o} = [o_1 \ldots o_c]^{\mathrm{T}}$)



◆ 정답 벡터의 표현

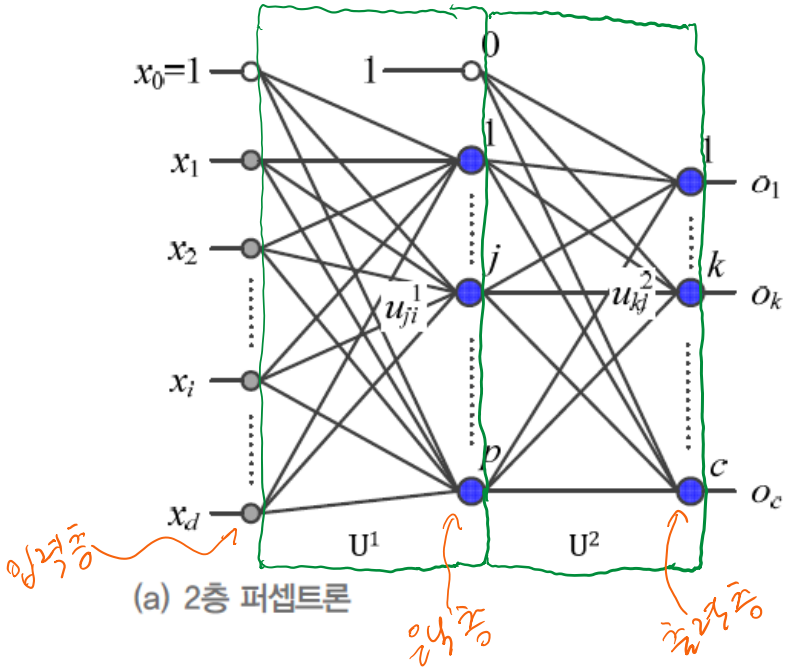
- Classification 문제일 경우, 정답 벡터는 ***One-Hot-Encoding***으로 표현됨
  - ★ k번째 class의 정답 벡터는 "k번째 자리는 1, 나머지 자리는 0"이 되도록 설정
  - ★ 예: $\mathbf{y} = [0, \cdots, 1, \cdots, 0]^{\mathrm{T}}$

Index	Job
1	Police
2	Doctor
3	Student
4	Teacher
5	Driver

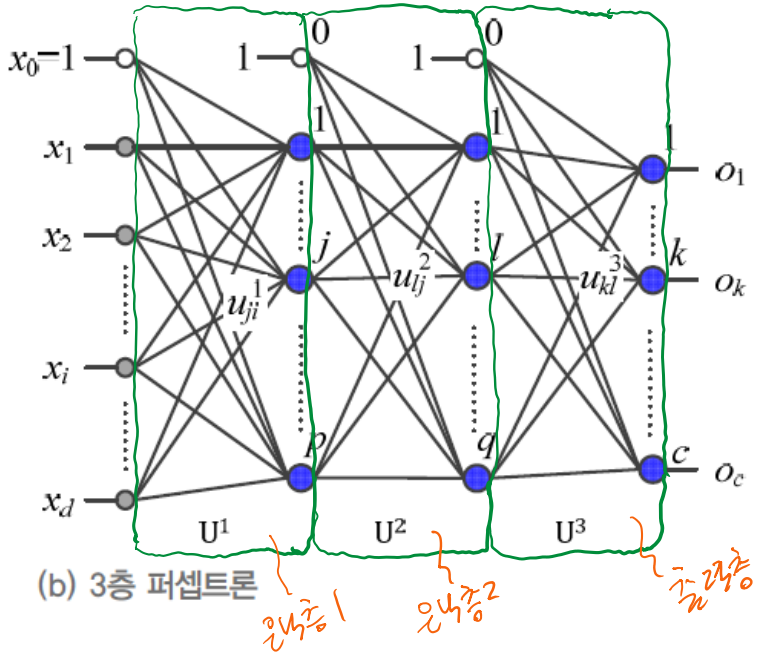
One hot encoded data
[1 0 0 0 0]
[0 1 0 0 0]
[0 0 1 0 0]
[0 0 0 1 0]
[0 0 0 0 1]

3. MLP Architecture (cont'd)

- 각 층의 가중치 (Parameters) 표현: $\mathbf{U}^l$ (*l*번째 층의 가중치 행렬)
  - $\mathbf{U}^l$은 l-1번째 층의 출력을 바탕으로 ***l*번째 층**의 출력을 계산하기 위한 모든 가중치를 모아 놓은 행렬
  - 이때 $\mathbf{U}^l$의 각 행은 l번째 층에 포함된 각 뉴런의 가중치 벡터 $\mathbf{u}_j^l$ (행벡터)가 됨. 예를 들어, $\mathbf{U}^l = \begin{bmatrix} \mathbf{u}_1^l \\ \vdots \\ \mathbf{u}_p^l \end{bmatrix}$
  - 은닉층의 개수 및 각 은닉층의 노드(뉴런) 개수는 hyper-parameter (즉, 모델 설계자가 설정)
    - ★ 예1: 은닉층 1개 → **2층 Perceptron** : $\Theta = \{\mathbf{U}^1, \mathbf{U}^2\}$
    - ★ 예2: 은닉층 2개 → **3층 Perceptron** : $\Theta = \{\mathbf{U}^1, \mathbf{U}^2, \mathbf{U}^3\}$



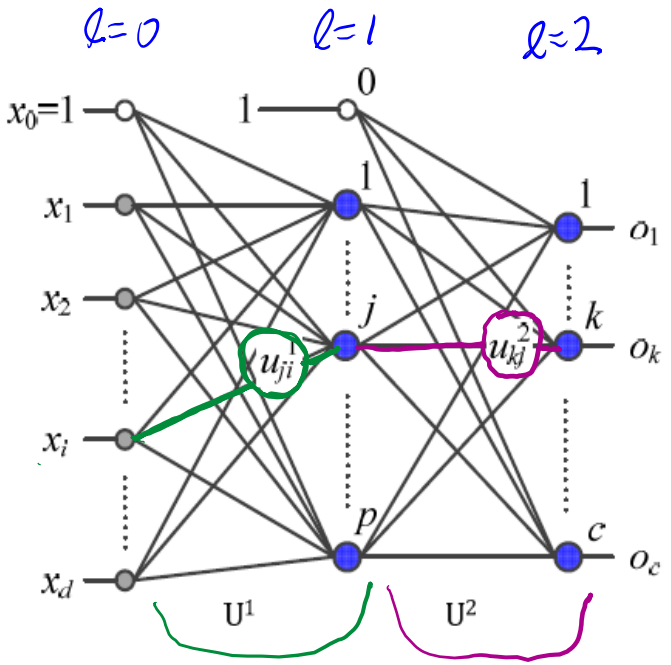
(a) 2층 퍼셉트론

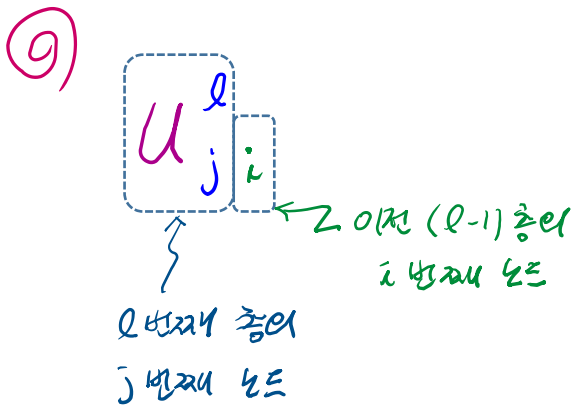


(b) 3층 퍼셉트론

3. MLP Architecture (cont'd)

- 각 층의 가중치 (Parameters) 표현 (cont'd): $\mathbf{U}^l > \mathbf{u}_j^l > u_{ji}^l$
  - $\mathbf{U}^l$: l-1번째 층으로부터 ***l*번째 층**에 연결되는 모든 가중치 (행렬)
  - $\mathbf{u}_j^l$: l-1번째 층으로부터 l번째 층의 ***j*번째 노드**에 연결되는 모든 가중치 (행벡터)
  - u_{ji}^l : l-1번째 층의 ***i*번째 노드**로부터 l번째 층의 ***j*번째 노드**와 연결되는 가중치 (값)





$$\mathbf{U}^1 = \begin{pmatrix} u_{10}^1 & u_{11}^1 & \cdots & u_{1d}^1 \\ u_{20}^1 & u_{21}^1 & \cdots & u_{2d}^1 \\ \vdots & \vdots & \ddots & \vdots \\ u_{p0}^1 & u_{p1}^1 & \cdots & u_{pd}^1 \end{pmatrix} = \begin{pmatrix} \underline{u}_1^1 \\ \underline{u}_2^1 \\ \vdots \\ \underline{u}_p^1 \end{pmatrix}$$

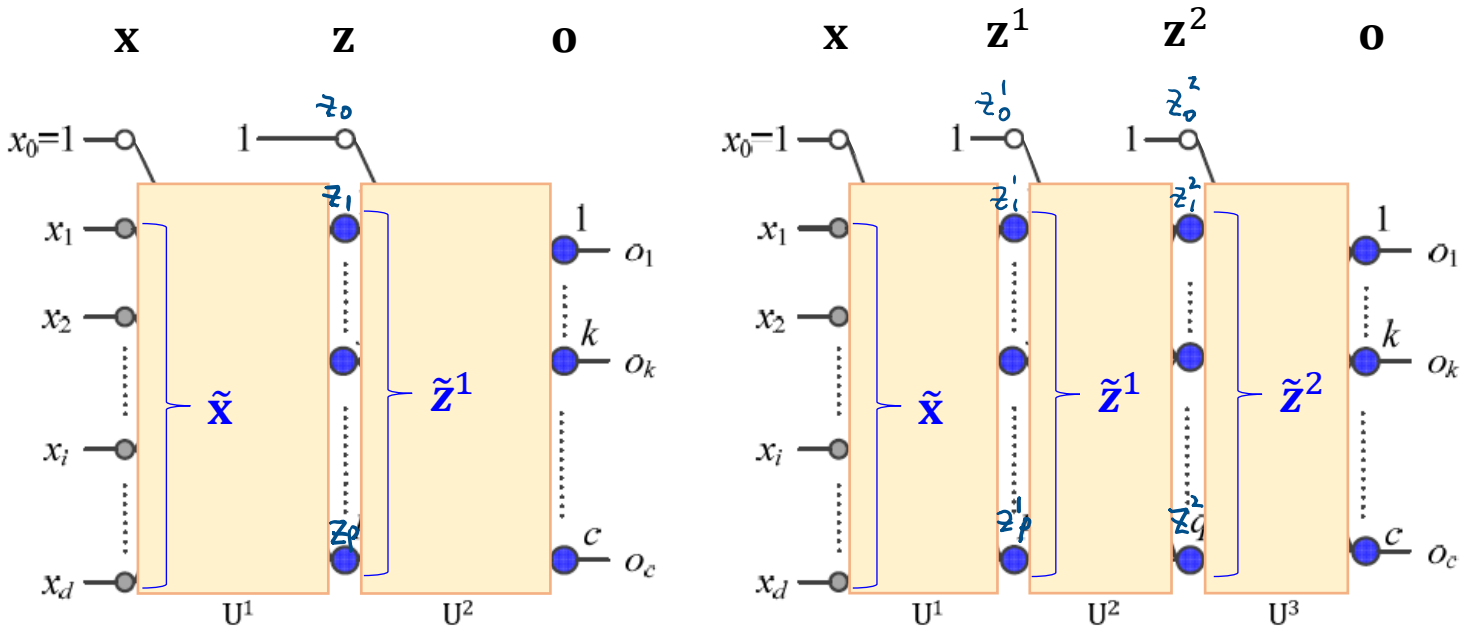
row vector

$$\mathbf{U}^2 = \begin{pmatrix} u_{10}^2 & u_{11}^2 & \cdots & u_{1p}^2 \\ u_{20}^2 & u_{21}^2 & \cdots & u_{2p}^2 \\ \vdots & \vdots & \ddots & \vdots \\ u_{c0}^2 & u_{c1}^2 & \cdots & u_{cp}^2 \end{pmatrix} = \begin{pmatrix} \underline{u}_1^2 \\ \underline{u}_2^2 \\ \vdots \\ \underline{u}_c^2 \end{pmatrix}$$

3. MLP Architecture (cont'd)

◆ 각 층의 출력 표현

- **입력층:** $\tilde{\mathbf{x}} = (x_1 \ \cdots \ x_d)^T$, $\mathbf{x} = (x_0 \ x_1 \ \cdots \ x_d)^T$
- **은닉층:** $\tilde{\mathbf{z}} = (z_1 \ \cdots \ z_p)^T$, $\mathbf{z} = (z_0 \ z_1 \ \cdots \ z_p)^T$ → l번째 은닉층의 출력: $\tilde{\mathbf{z}}^l$, $l+1$번째 층의 입력: $\mathbf{z}^l$
- **출력층:** $\mathbf{o} = \tilde{\mathbf{o}} = (o_1 \ \cdots \ o_c)^T$



(a) 2층 퍼셉트론 (b) 3층 퍼셉트론

4. Forward Computation (전방계산)

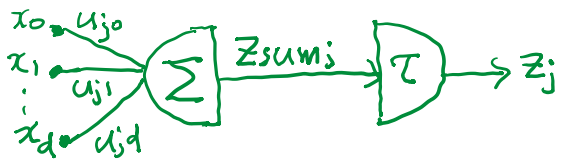
◆ 은닉층 노드의 연산

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



- **행렬-벡터 형태 :** $\tilde{\mathbf{z}}_{sum} = \mathbf{U}^1\mathbf{x},\quad \tilde{\mathbf{z}} = \tau(\tilde{\mathbf{z}}_{sum})$

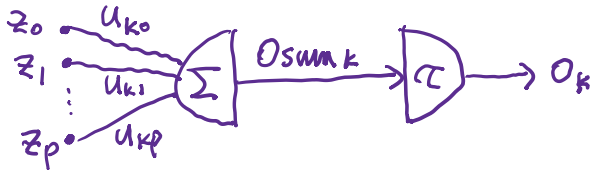
◆ 출력층 노드의 연산

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

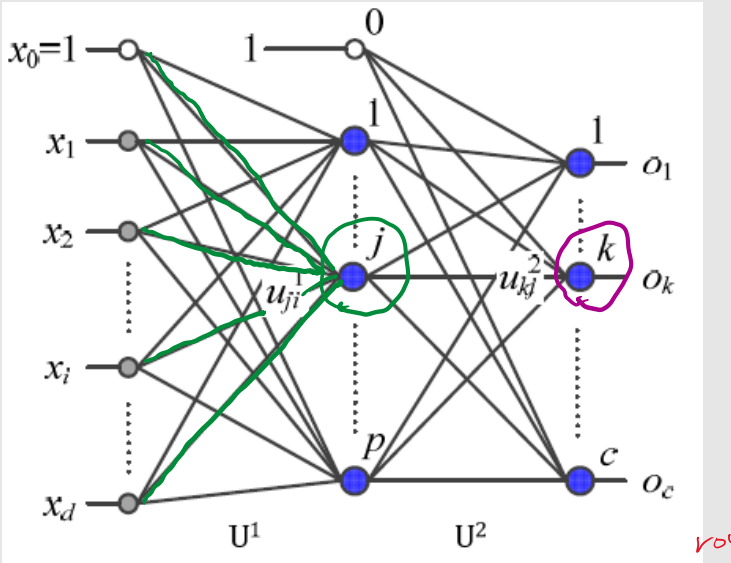
k번째 출력 노드의 연산:

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

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



$$\mathbf{U}^1 = \begin{pmatrix} u_{10}^1 & u_{11}^1 & \cdots & u_{1d}^1 \\ u_{20}^1 & u_{21}^1 & \cdots & u_{2d}^1 \\ \vdots & \vdots & \ddots & \vdots \\ u_{p0}^1 & u_{p1}^1 & \cdots & u_{pd}^1 \end{pmatrix} = \begin{pmatrix} \underline{u}_1^{1} \\ \underline{u}_2^{1} \\ \vdots \\ \underline{u}_p^{1} \end{pmatrix}$$

row vector

$$\mathbf{U}^2 = \begin{pmatrix} u_{10}^2 & u_{11}^2 & \cdots & u_{1p}^2 \\ u_{20}^2 & u_{21}^2 & \cdots & u_{2p}^2 \\ \vdots & \vdots & \ddots & \vdots \\ u_{c0}^2 & u_{c1}^2 & \cdots & u_{cp}^2 \end{pmatrix} = \begin{pmatrix} \underline{u}_1^{2} \\ \underline{u}_2^{2} \\ \vdots \\ \underline{u}_c^{2} \end{pmatrix}$$

Exercise: Build Your Toy MLP

- For given the following setting,
  - Input: 3 features (i.e., d=3)
  - Output: 3 class probabilities (i.e., c=3)
  - # of hidden layers: 2 (i.e., L=2)
    - ★ # of nodes in the first hidden layer: 3 (i.e., p=3)
    - ★ #of nodes in the second hidden layer: 2 (i.e., q=2)

(1) Draw the MLP architecture

(2) Mathematically specify the following data structures

- 각 층의 뉴런들의 출력 벡터: $\tilde{\mathbf{z}}^1, \tilde{\mathbf{z}}^2, \mathbf{o}$
- 각 층의 가중치 행렬: $\mathbf{U}^1, \mathbf{U}^2, \mathbf{U}^3$
- Point: 각각의 벡터와 행렬의 dimension을 올바르게 기재할 수 있는가?

Q & A