

Python

Machine Learning

Chap.5

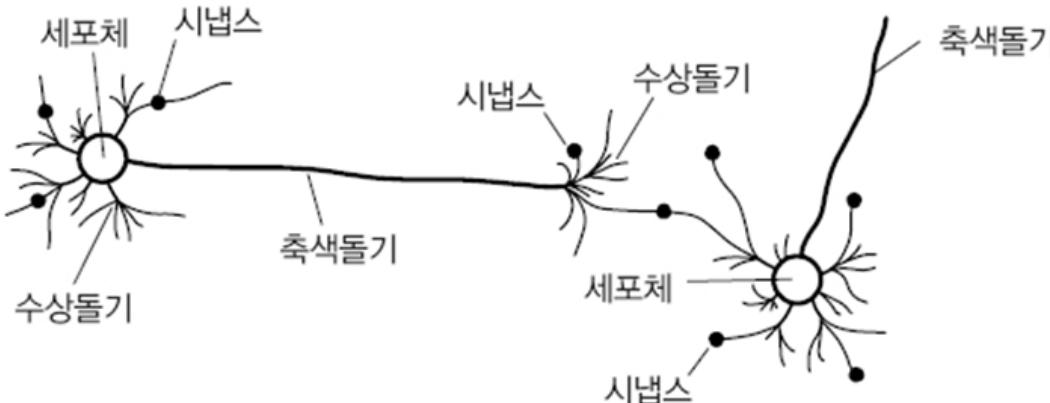


I What is Neuron?

❖ 인공 신경망

▪ 인공 신경망

- 인간 뇌를 기반으로 한 추론 모델.
- 인간 뇌의 추론 모델 - 뉴런(neuron) : [그림 6-1]



[그림 6-1] 생물학적인 신경망

- 뉴런은 기본적인 정보처리 단위.
- 인공 신경망의 주요 특징은 적응성을 가지고 있다는 것이다.

▪ 인간 뇌의 특징

- 100개의 뉴런과 각 뉴런을 연결하는 6조 개의 시냅스의 결합체
- 인간의 뇌는 현존하는 어떤 컴퓨터보다 빠르게 기능을 수행할 수 있음

■ Brain vs Artificial Neuron Network

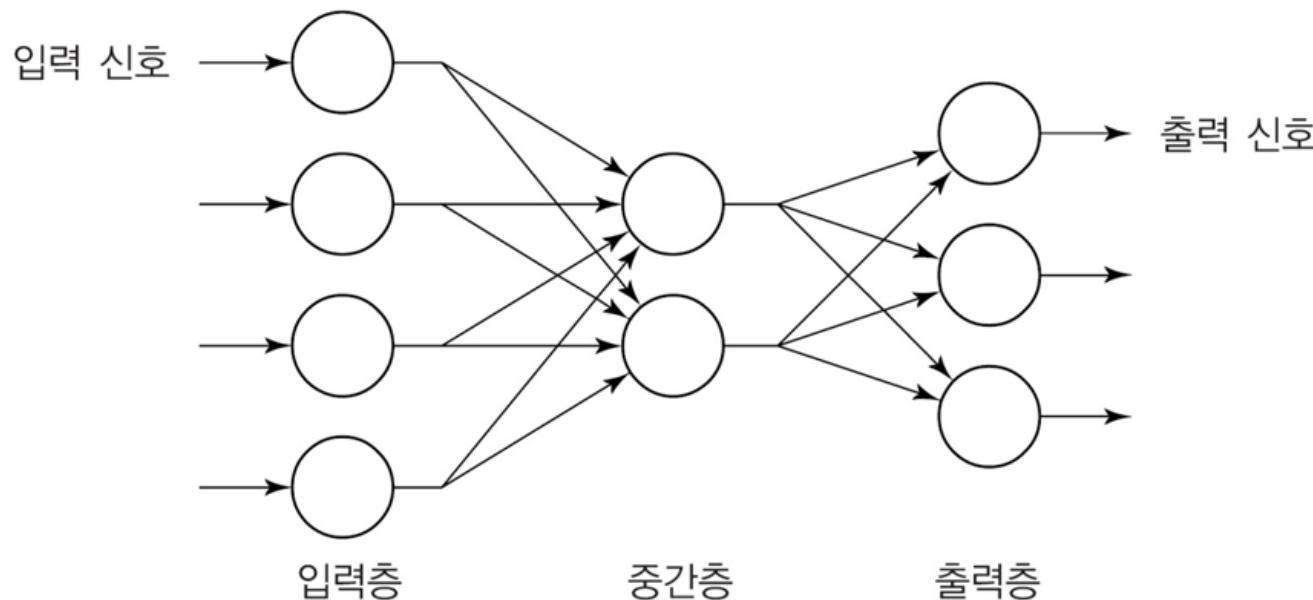
- 인간 뇌의 특징
 - 인간의 뇌는 매우 복잡하고, 비선형적이며, 병렬적인 정보 처리 시스템으로 생각할 수 있다.
 - 정보는 신경망 전체에 동시에 저장되고 처리된다.
 - 적응성에 따라 '잘못된 답'으로 이끄는 뉴런들 사이의 연결은 약화되고, '올바른 답'으로 이끄는 연결은 강화된다.
- 인공 신경망의 특징
 - 인간 뇌를 기반으로 모델링 함.
 - 인간 뇌의 적응성을 활용하여 '학습 능력'을 구현함.
 - 인공 신경망은 아직 인간의 뇌를 흉내내기에 미흡하다.
- 인공 신경망을 이용한 뇌 모델링
- 인공 신경망의 학습
- 인공 신경망의 가중치 조정

Artificial Neural Networks

❖ 인공 신경망 모델링

▪ 인간의 뇌 모델링

- 생물학적인 뇌의 뉴런과 비슷하게 모델링함.
- 인공 시경망은 뉴런이라는 아주 단순하지만 내부적으로 매우 복합하게 연결된 프로세스들로 이루어져 있음.
- 뉴런은 가중치 있는 링크들로 연결되어 있음.
- 각각의 뉴런은 연결을 통해 여려 입력 신호를 받지만 출력 신호는 오직 하나만 만듦.
- 인공 신경망 구조 [그림 6-2]



[그림 6-2] 전형적인 인공 신경망의 구조

Artificial Neural Networks

❖ 인공 신경망 모델링

▪ 인간의 뇌 모델링

- 생물학적인 신경망과 인공 신경망의 유사점

[표 6-1] 생물학적인 신경망과 인공 신경망 사이의 유사점

생물학적인 신경망	인공 신경망
세포체	뉴런
수상돌기	입력
축색돌기	출력
시냅스	가중치

▪ 인공 신경망의 학습

- 신경망은 가중치를 반복적으로 조정하여 학습함.
- 뉴런은 링크(link)로 연결되어 있고, 각 링크에는 그와 연관된 수치적인 가중치가 있다.
- 가중치는 장기 기억을 위한 기본적인 수단으로, 각 뉴런 입력 강도, 즉 중요도를 표현.

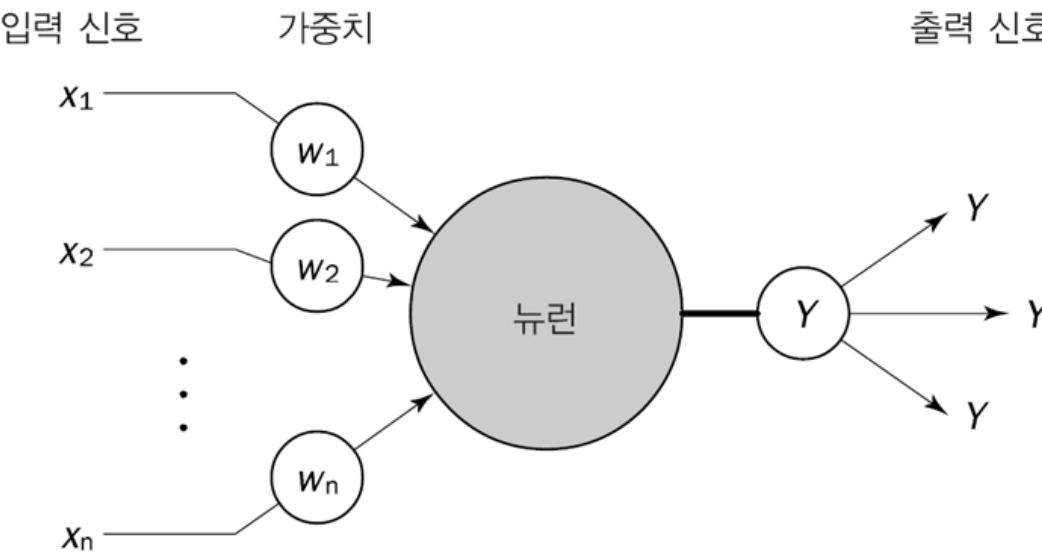
▪ 인공 신경망의 가중치 조정

- 신경망의 가중치를 초기화하고 훈련 예제들의 집합에서 해당 가중치를 갱신.
- 신경망의 구조를 먼저 선택하고, 어떤 학습 알고리즘을 사용할 것인지 결정한 후, 신경망을 훈련시킨다.

Artificial Neural Networks

▪ 뉴런의 특징

- 입력 링크에서 여러 신호를 받아서 새로운 활성화 수준을 계산하고, 출력 링크로 출력 신호를 내보낸다.
- 입력 신호는 미가공 데이터 또는 다른 뉴런의 출력이 될 수 있다.
- 출력 신호는 문제의 최종적인 해(solution)거나 다른 뉴런에 입력될 수 있다.
- 뉴런의 예 : [그림 6-3]



[그림 6-3] 뉴런의 도식

Artificial Neural Networks

❖ 뉴런의 계산

▪ 뉴런의 출력 결정

- 렌 맥클록(Warren McCulloch)과 월터 피츠(Walter Pitts)가 제안(1943년).
- 뉴런은 다음과 같은 전이 함수, 즉 활성화 함수(activation function)를 사용
- 활성화 함수를 이용한 출력 결정 순서
 1. 뉴런은 입력 신호의 가중치 합을 계산하여 임계값 θ 와 비교한다.
 2. 가중치 합이 임계값보다 작으면 뉴런의 출력은 '-1'이다.
 3. 중치 합이 임계값과 같거나 크면 뉴런은 활성화되고, 뉴런의 출력은 '+1'이 된다

▪ 활성화 함수 : 식(6.1)

$$X = \sum_{i=1}^n x_i w_i \quad (6.1)$$

$$Y = \begin{cases} +1 & \text{if } X \geq \theta \\ -1 & \text{if } X < \theta \end{cases}$$

x는 뉴런으로 들어가는 입력의 순 가중합

x_i 는 입력 i의 값, w_i 는 입력 i의 가중치, n은 뉴런의 입력 개수, Y는 뉴런의 출력

Artificial Neural Networks

❖ 뉴런의 계산

▪ 뉴런의 활성화 함수

- 계단(step)과 부호(sign) 활성화 함수는 하드 리밋 함수(hard limit function)라고도 하며, 분류와 패턴인식 작업에서 결정을 내리는 뉴런에 주로 쓰인다.
- 시그모이드 함수(sigmoid function)는 양과 음의 무한대 사이에 있는 입력값을 0~1 사이에 있는 적당한 값으로 바꾼다. 시그모이드 함수를 사용하는 뉴런은 역전파 신경망에 쓰인다.
- 선형 활성화 함수(linear activation function)는 뉴런의 입력에 가중치가 적용된 것과 같은 값을 출력으로 내놓는다. 선형 함수를 사용하는 뉴런은 선형 근사에 주로 쓰인다.

▪ 단일 뉴런의 학습

- 프랭크 로젠틀랫이 간단한 인공 신경망을 훈련시키기 위한 알고리즘인 퍼셉트론을 소개했다(1958년).
- 퍼셉트론은 신경망의 가장 간단한 형태로 조정 가능한 시냅스 가중치와 하드 리미터(hard limiter)를 포함한 단일 뉴런으로 구성된다.

Perceptron

I Perceptron model

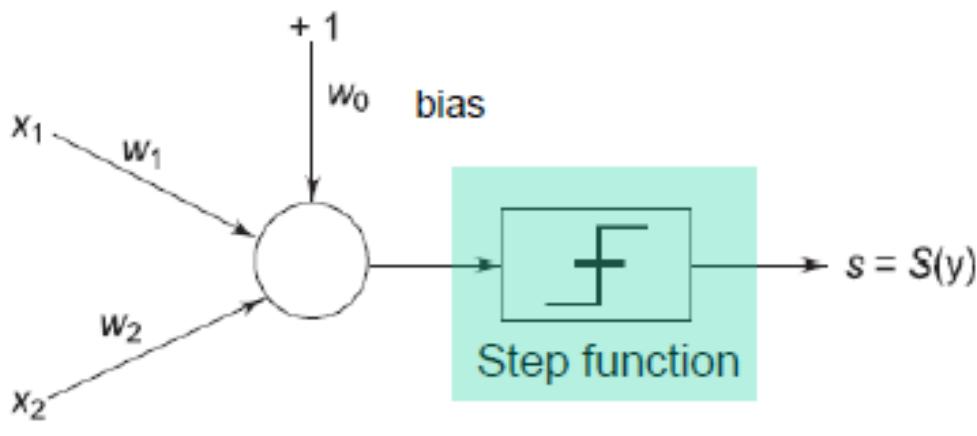
- 생물학적 신경세포의 작동 방식을 수학적으로 모델링한 **인공 뉴런**
- 활성함수activation function가 적용되기 전의 출력값

$$s = \sum_{i=1}^n w_i x_i + w_0 = \sum_{i=0}^n w_i x_i = \mathbf{w} \cdot \mathbf{x}$$

가중치 weight 벡터
 $x_0 = 1$ 입력값 input 벡터

- 적용하는 활성함수에 따라 출력값을 다양하게 활용 가능

I Perceptron model



- Perceptron Input vector $X = [1, x_1, x_2]$
- Weight vector $W = [w_0, w_1, w_2]$
- Perceptronal activation $y = \text{step}(X^T W) = \text{step}(w_1 x_1 + w_2 x_2 + w_0)$
- Perceptron discriminant function:
 $y(X) = 0$ if $y > 0$, $s = 1$, and if $y < 0$, $s = 0$.

I Perceptron model

- 퍼셉트론의 특징

- 로젠블랫 퍼셉트론의 동작 원리는 맥클록과 피츠의 뉴런 모델에 기반.
- 퍼셉트론은 선형 결합기와 하드 리미터로 구성.
- 입력의 가중합을 하드 리미터에 입력하고, 입력이 양이면 '+1', 음이면 '-1'을 출력.
- 기본적인 퍼셉트론의 경우, 초평면(hyperplane)으로 n차원 공간을 두 개의 결정 영역으로 나눈다.
- 초평면을 선형 분리 함수로 정의한다.
- 선형 분리 함수 : 식(6.3)

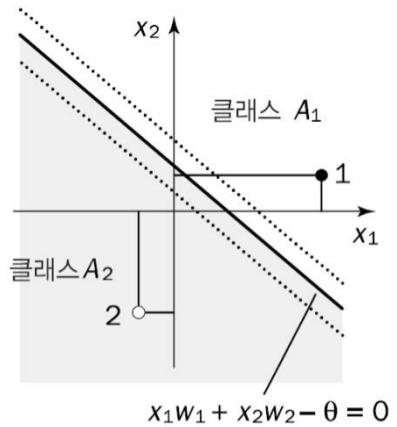
$$\sum_{i=1}^n x_i w_i - \theta = 0 \quad (6.3)$$

- 입력이 x_1 과 x_2 로 두 개인 경우, 결정 경계는 [그림 6-6]의 (a)에 보이듯 굵은 직선 형태로 나타낸다.
- 경계선 오른편에 있는 점(1)은 클래스 A_1 에 속하고, 경계선 왼편에 있는 점(2)는 클래스 A_2 에 속한다.
- 임계값 θ 는 결정 경계를 옮기는 데 쓰인다.

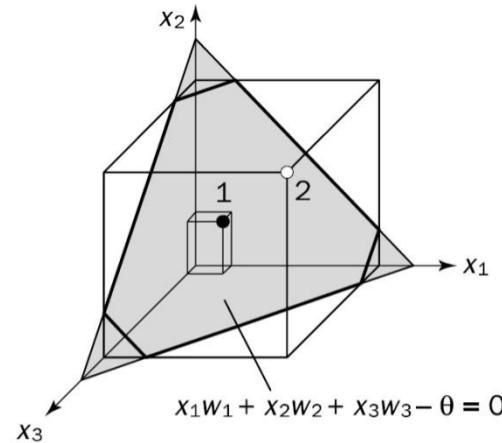
I Perceptron model

▪ 퍼셉트론의 특징

- 입력이 세 개일 때는 도식화 할 수 있다. [그림 6-6]의 (b)
- [그림 6-6]은 입력이 세 개인 퍼셉트론에 대한 3차원 공간을 보여준다.



(a) 입력이 두 개인 퍼셉트론



(b) 입력이 세 개인 퍼셉트론

[그림 6-6] 퍼셉트론에서의 선형 분리성

▪ 분할 식

입력이 두 개인 퍼셉트론

$$x_1w_1 + x_2w_2 - \theta = 0$$

입력이 세 개인 퍼셉트론

$$x_1w_1 + x_2w_2 + x_3w_3 - \theta = 0$$

I Perceptron model

- 매개변수: 데이터와 관계 없이 **모델을 구성**하는 변수.
 - 퍼셉트론에서는 가중치들이 퍼셉트론의 매개변수에 해당함

$$s = \sum_{i=1}^n w_i x_i + w_0 = \sum_{i=0}^n w_i x_i = \mathbf{w} \cdot \mathbf{x}$$

가중치: 매개변수
($x_0 = 1$) 입력값: 매개변수 아님

- 가설집합: 학습을 통해 구현할 수 있는 **모델의 범위**
 - 매개변수가 표현할 수 있는 범위가 넓을수록 커짐
 - 가중치 w 가 정수만 가질 수 있는 경우 < 가중치 w 가 실수를 가질 수 있는 경우
 - 매개변수가 많을수록 커짐
 - 가중치 w 가 3개인 퍼셉트론 < 가중치 w 가 200개인 퍼셉트론

I Perceptron model

데이터

$$\mathcal{D}_N = \{(\mathbf{x}^{(d)}, y^{(d)})\}_{d=1}^N$$

출력값

$$f_{perceptron}^{(d)} = f_{perceptron}(\mathbf{x}^{(d)}; \mathbf{w}) = f_{lin}\left(\sum_i w_i x_i^{(d)}\right) = \sum_i w_i x_i^{(d)}$$

Loss

$$\mathcal{L}_N = \sum_{d=1}^N \mathcal{L}_d, \quad \mathcal{L}_d = \frac{1}{2} \left(y^{(d)} - f_{perceptron}^{(d)} \right)^2$$

I Perceptron model

- 오차를 줄이는 방향으로 가중치를 교정

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = -\eta \frac{\partial L_d}{\partial w_i}$$

η : 학습률 learning rate

$$f_{perceptron}^{(d)} = \sum_i w_i x_i^{(d)}$$

- 교정하는 방향과 크기를 알기 위해 편미분 수행

$$\frac{\partial L_d}{\partial w_i} = \frac{\partial L_d}{\partial f_{perc}^{(d)}} \cdot \frac{\partial f_{perc}^{(d)}}{\partial w_i} = \frac{\partial}{\partial f_{perc}^{(d)}} \frac{1}{2} (y^{(d)} - f_{perc}^{(d)})^2 \frac{\partial f_{perc}^{(d)}}{\partial w_i}$$

$$= \frac{1}{2} (-2) (y^{(d)} - f_{perc}^{(d)}) x_i^{(d)} = -(y^{(d)} - f_{perc}^{(d)}) x_i^{(d)}$$

$$\Delta w_i = -\eta \frac{\partial L_d}{\partial w_i} = \eta (y^{(d)} - f_{perc}^{(d)}) x_i^{(d)}$$

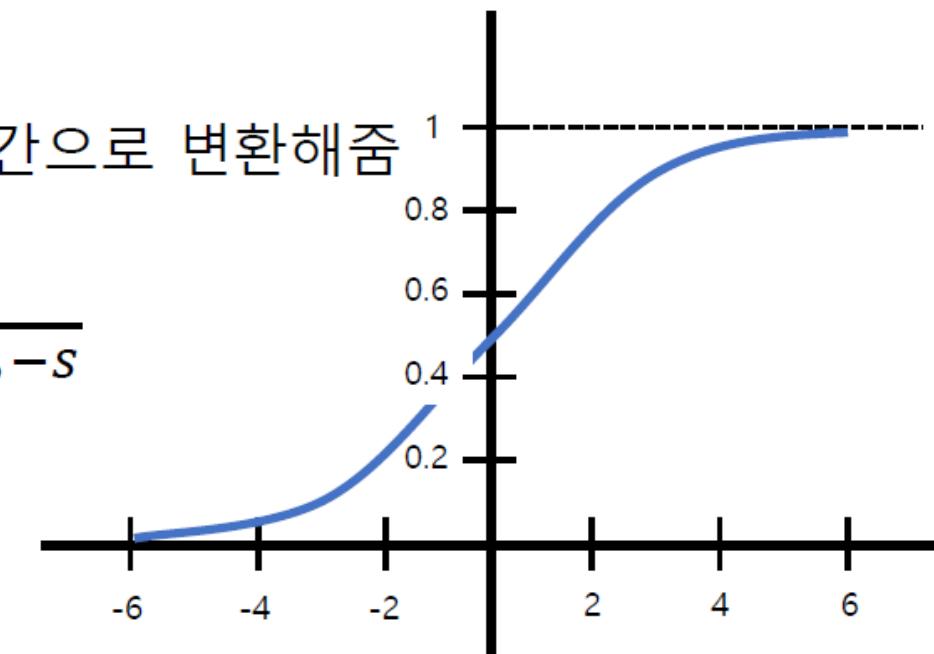
I Perceptron model

- 시그모이드 Sigmoid 활성화 함수

- 선형 활성화 함수는 선형 분리가 가능한 패턴분류 문제만 해결 가능
- 복잡한 문제를 풀기 위해서는 비선형 결정 경계를 생성 가능해야 함
- 활성화 함수의 미분이 가능하려면 연속함수여야 함
- 시그모이드 함수의 성질
 - 미분 가능한 비선형 함수
 - $[-\infty, +\infty]$ 의 입력 구간에 대한 출력을 $[0, 1]$ 의 구간으로 변환해줌

$$f_{sigmoid}(s) = \sigma(s) = \frac{1}{1 + e^{-s}}$$

$$\frac{d\sigma(s)}{ds} = \sigma(s)(1 - \sigma(s))$$



I Perceptron model

$$\frac{d\sigma(s)}{ds} = \sigma(s)(1 - \sigma(s))$$

- 시그모이드 Sigmoid 활성화 함수

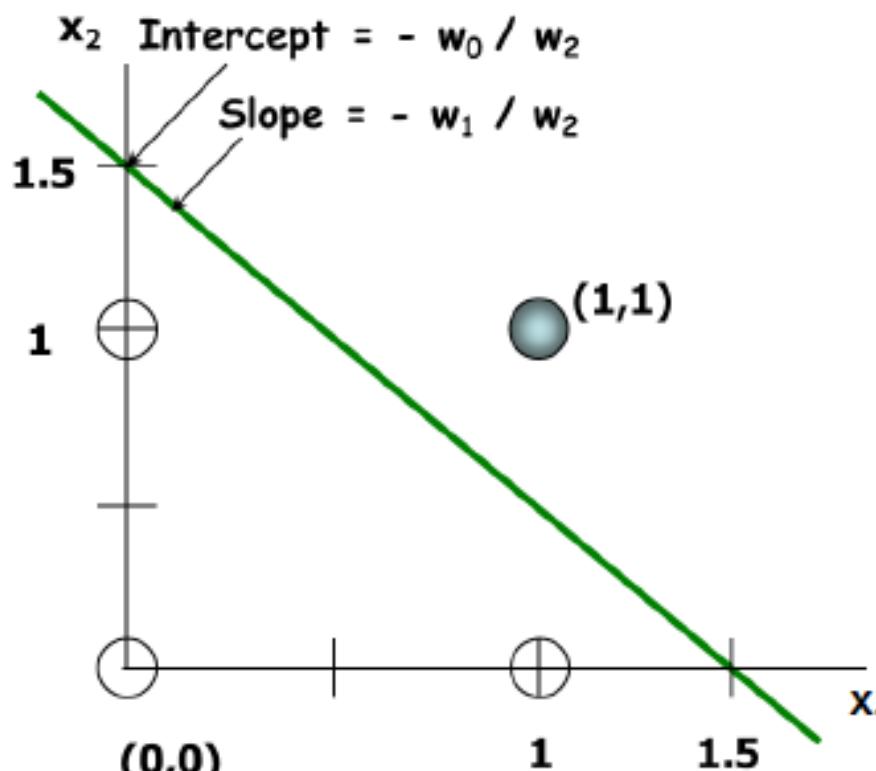
$$\bullet s^{(d)} = \sum_i w_i x_i^{(d)}, f_{perc}^{(d)} = f_{sig}\left(\sum_i w_i x_i^{(d)}\right) = \frac{1}{1+e^{-s^{(d)}}}$$

$$\begin{aligned}\bullet \frac{\partial L_d}{\partial w_i} &= \frac{\partial L_d}{\partial f_{perc}^{(d)}} \cdot \frac{\partial f_{perc}^{(d)}}{\partial w_i} = \frac{\partial}{\partial f_{perc}^{(d)}} \frac{1}{2} (y^{(d)} - f_{perc}^{(d)})^2 \frac{\partial f_{perc}^{(d)}}{\partial s^{(d)}} \cdot \frac{\partial s^{(d)}}{\partial w_i} \\ &= \frac{1}{2} (-2)(y^{(d)} - f_{perc}^{(d)}) f_{perc}^{(d)} (1 - f_{perc}^{(d)}) x_i^{(d)} \\ &= -(y^{(d)} - f_{perc}^{(d)}) f_{perc}^{(d)} (1 - f_{perc}^{(d)}) x_i^{(d)}\end{aligned}$$

$$\Delta w_i = -\eta \frac{\partial L_d}{\partial w_i} = \eta (y^{(d)} - f_{perc}^{(d)}) [f_{perc}^{(d)} (1 - f_{perc}^{(d)}) x_i^{(d)}]$$

Example 1

- slope $m = -w_1/w_2 = -1$, intercept $c = -w_0/w_2 = 1.5$.
- Choosing $w_1 = w_2 = 1$ and $w_0 = -1.5$ yields a perceptron classifier that appropriately partitions the pattern space for AND classification.
- The value of the threshold is -1.5 .



x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

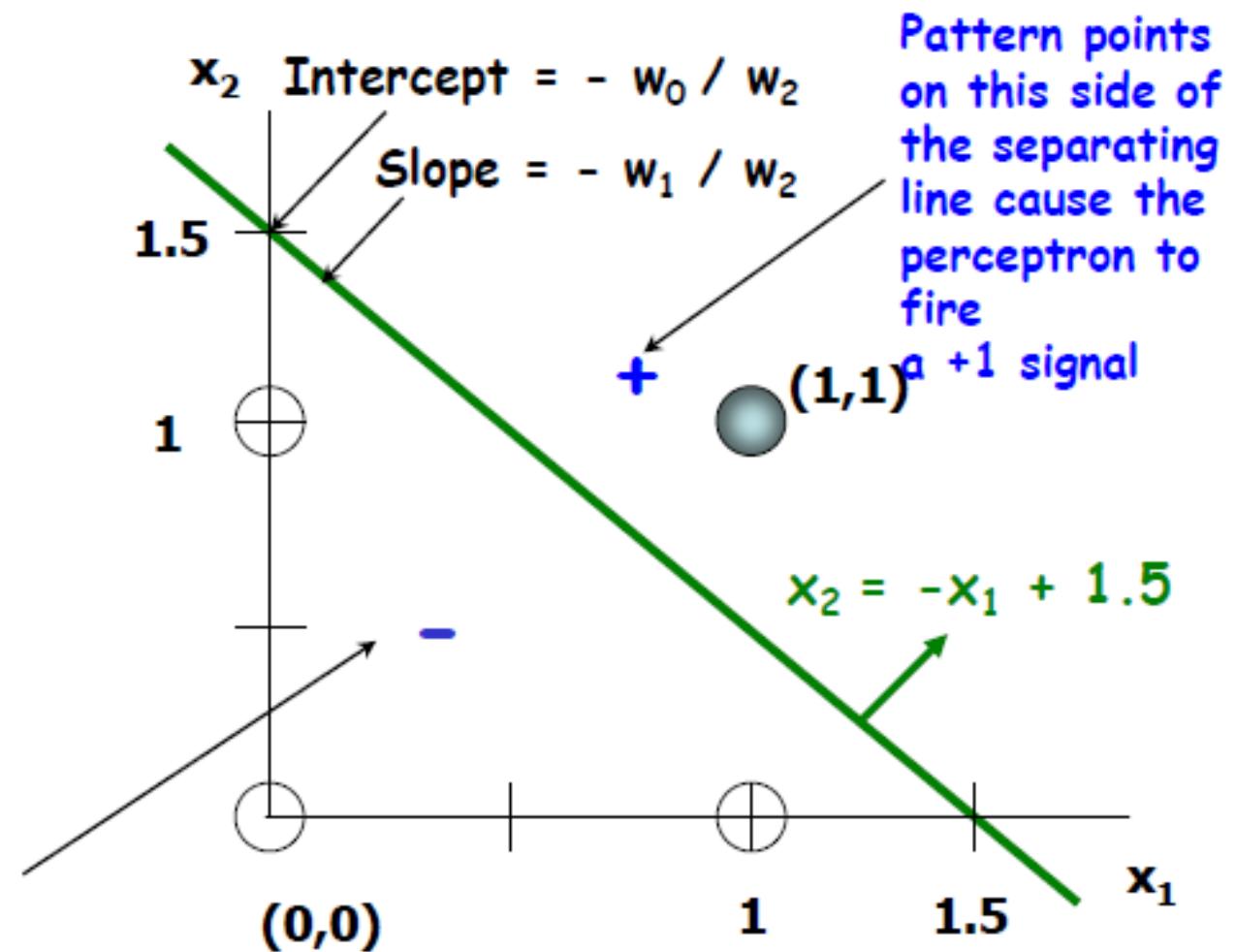
$$x_2 = -\frac{w_1}{w_2}x_1 - \frac{w_0}{w_2}$$

Example 1

- Discriminant function thus becomes

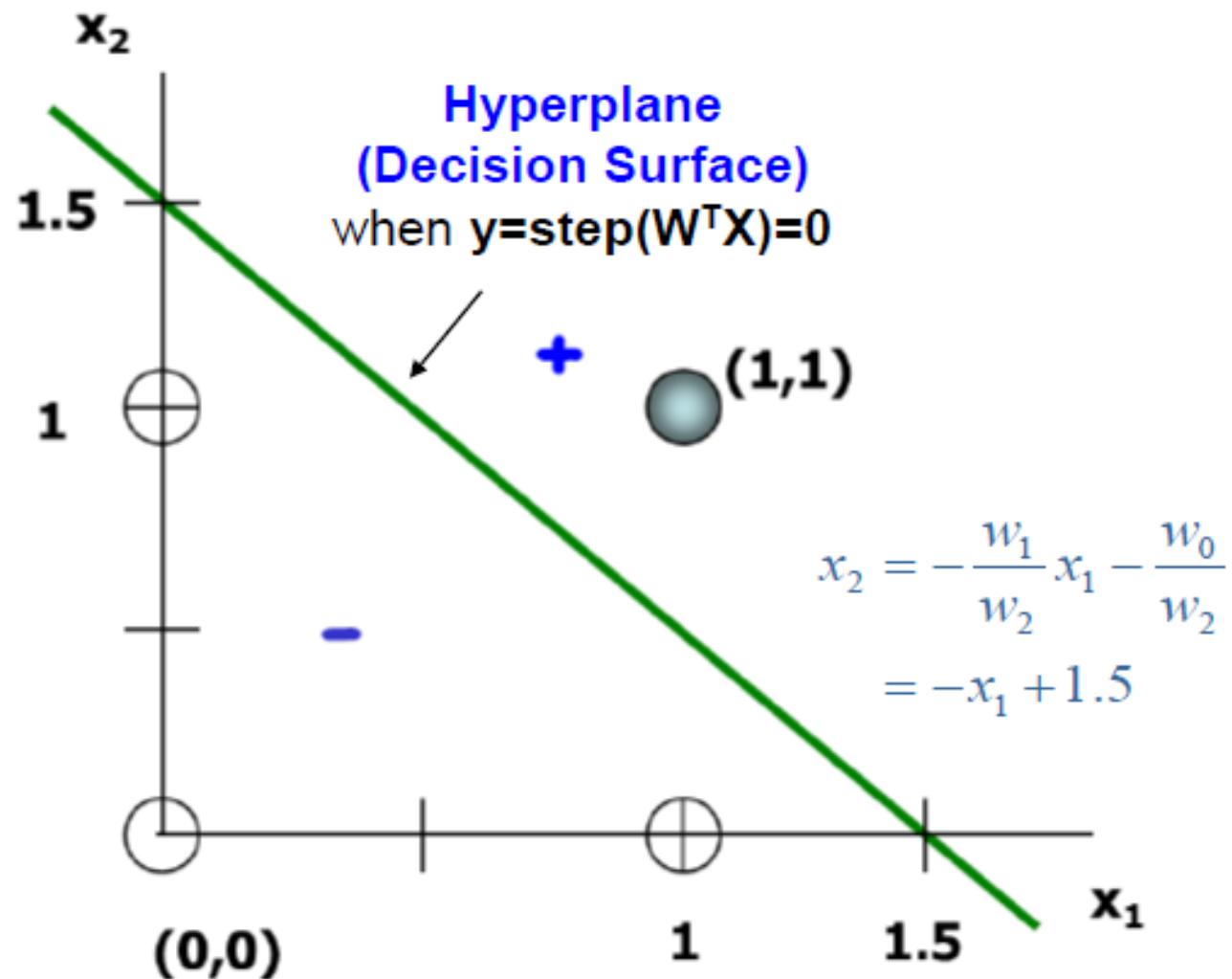
$$x_2 = -\frac{w_1}{w_2}x_1 - \frac{w_0}{w_2}$$
$$= -x_1 + 1.5$$

Pattern points on this side cause the perceptron to fire a 0 signal



I Summary of Important Properties of Perceptrons

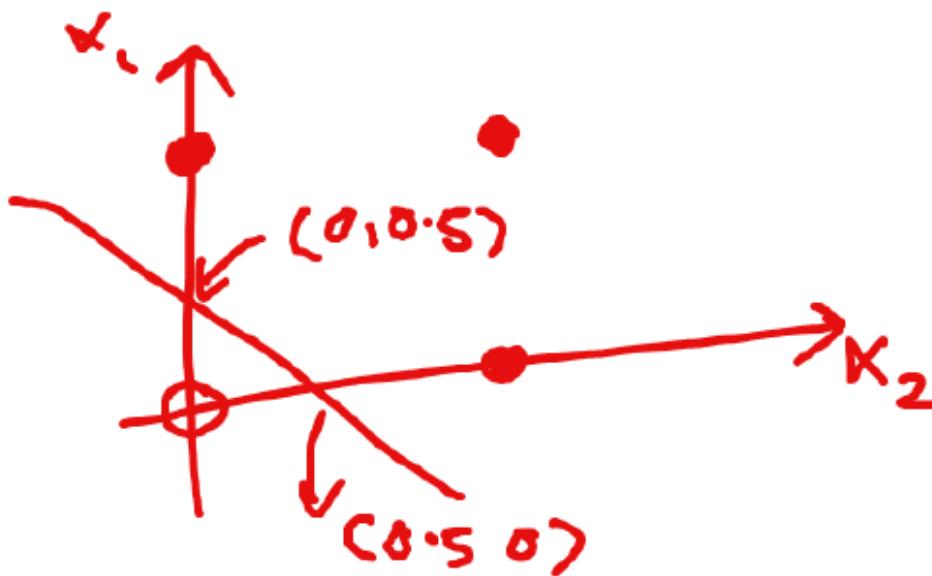
$$y(X) = \text{step}(W^T X) = \text{step}(x_2 + x_1 - 1.5)$$



I More examples

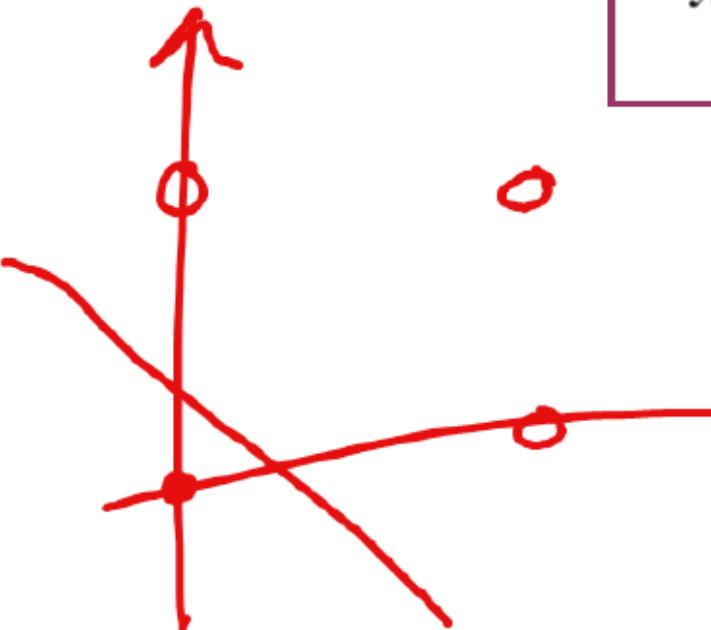
OR: $x_1 + x_2$

x_1	x_2	$x_1 + x_2$
0	0	0
0	1	1
1	0	1
1	1	1



NOR: $(x_1+x_2)'$

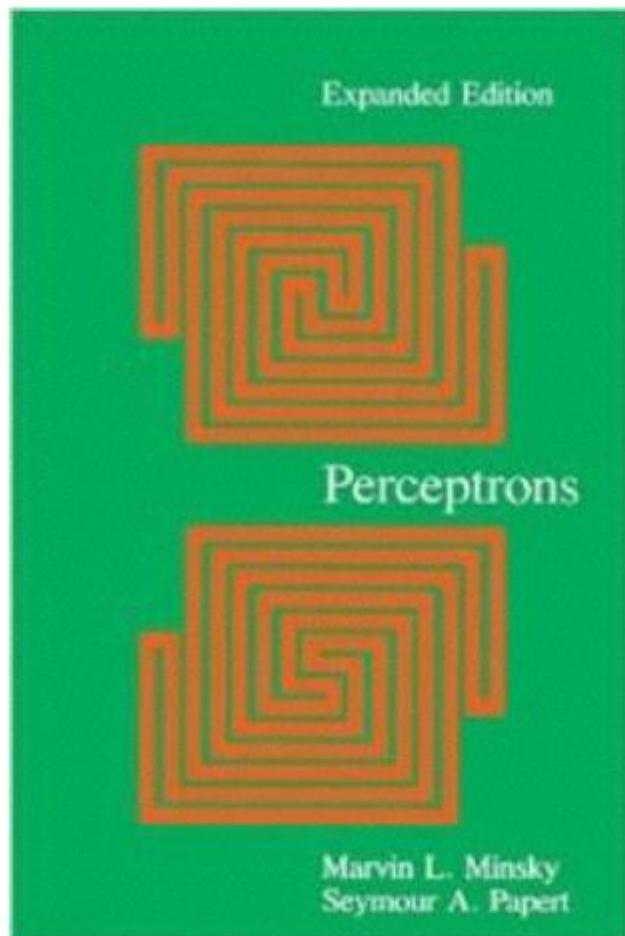
x_1	x_2	$(x_1+x_2)'$
0	0	1
0	1	0
1	0	0
1	1	0



$$x_2 = -\frac{w_1}{w_2}x_1 - \frac{w_0}{w_2}$$

I Perceptrons

by Marvin Minsky, founder of the MIT AI Lab



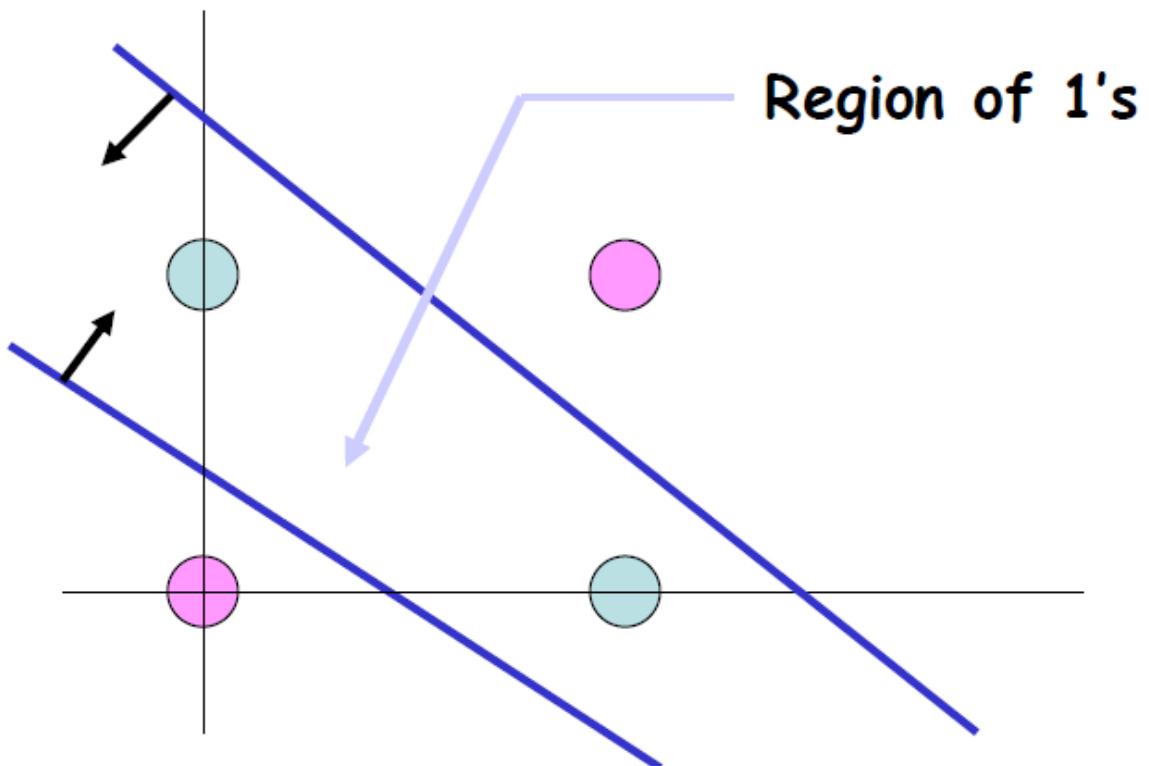
- We need to use MLP, multilayer perceptrons (multilayer neural nets)
- No one on earth had found a viable way to train MLPs good enough to learn such simple functions.

XOR Problem

- The geometry of the Boolean XOR function shows that **two straight lines** are required for proper class separation

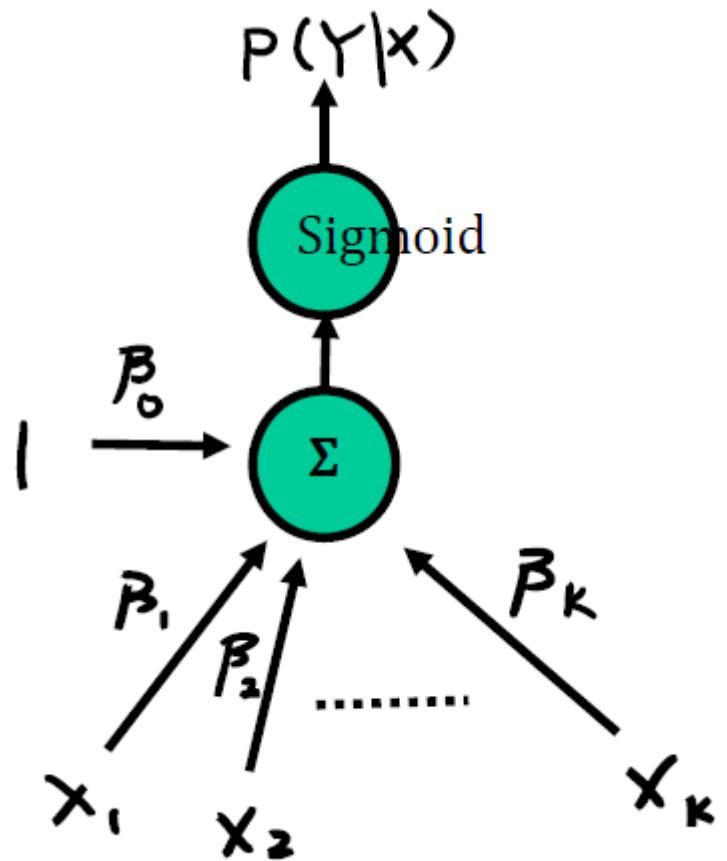
Truth Table

x_1	x_2	S
0	0	0
0	1	1
1	0	1
1	1	0



- No single perceptron exists that can solve the XOR classification problem. We need **more than one** perceptron.

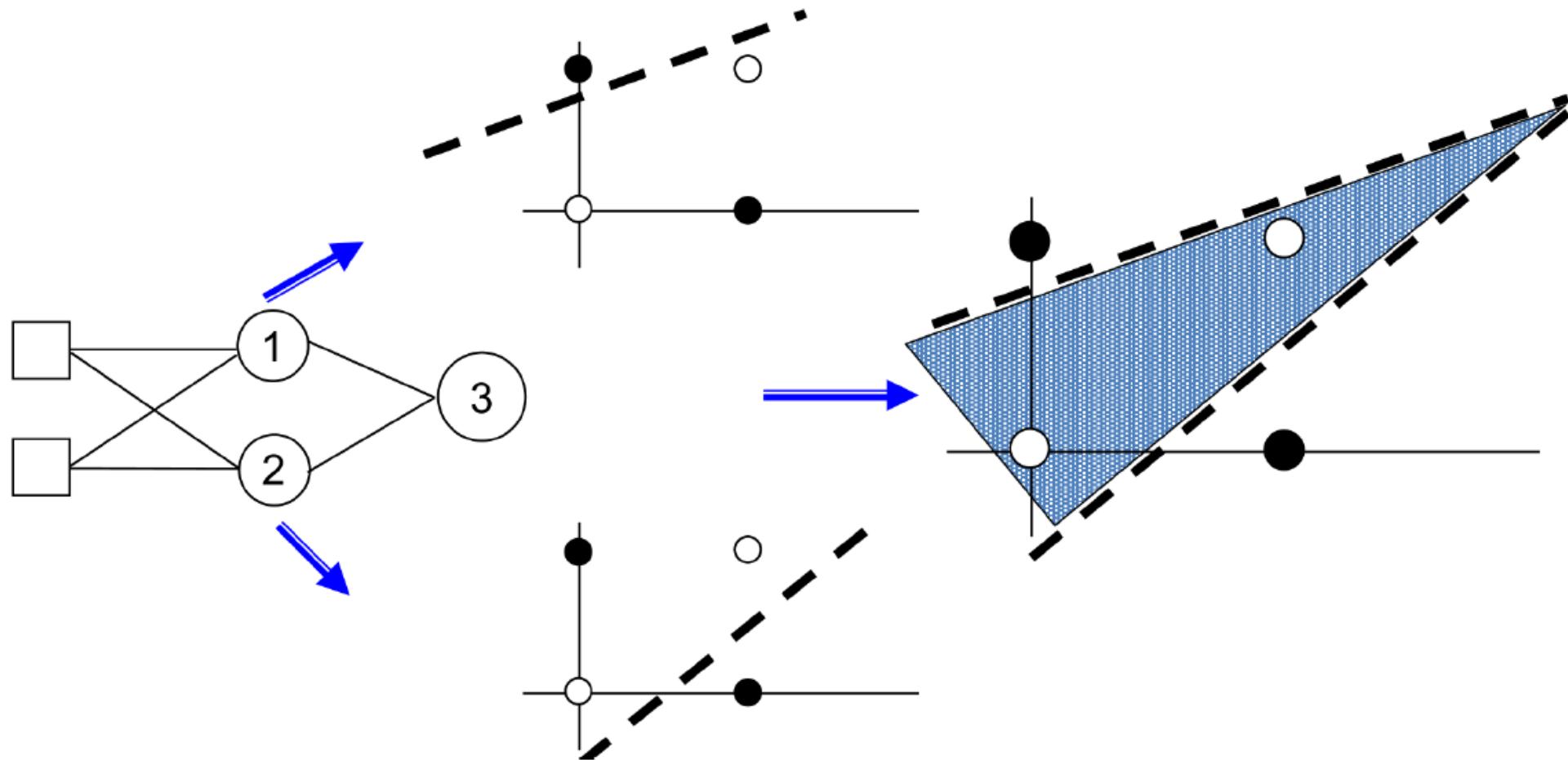
XOR Problem



$$a = w_0x_0 + w_1x_1 + w_2$$

같은 단층 퍼셉트론 모델을 이용해서 수작업으로 얻든,
랜덤함수로 얻든, 학습으로 얻든 x_0, x_1 의 적당한 조합이
XOR을 작동하게 한다면 민斯基의 주장이 틀린 것.

XOR Problem



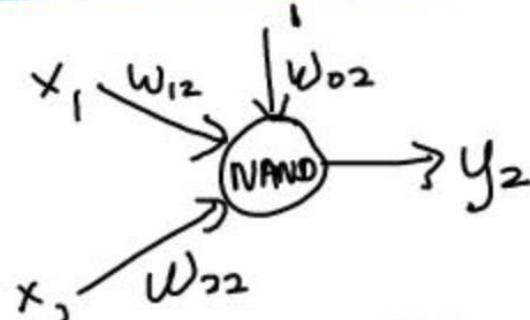
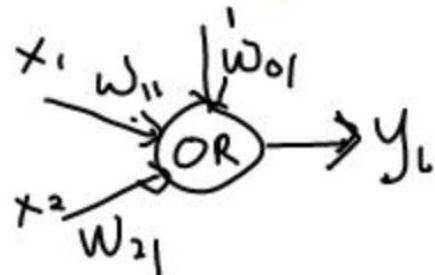
Introduce two neurons to create two hyperplanes and another neuron to connect two hyperplanes.

I Solving XOR Problem

$$\begin{aligned} & x_1 \oplus x_2 \\ &= x_1 x_2' + x_1' x_2 \\ &= (x_1 + x_2)(x_1' + x_2') \\ &= (x_1 + x_2)(x_1 x_2)' \\ &= (x_1 \text{ OR } x_2) \text{ AND } \underbrace{(x_1 \text{ AND } x_2)'}_{\text{NAND}} \end{aligned}$$

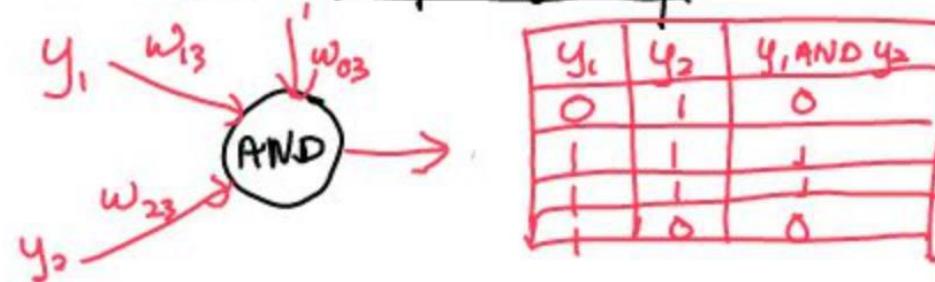
Solving XOR Problem

x_1	x_2	$y_1 = x_1 \text{ OR } x_2$	$y_2 = (x_1 \text{ AND } x_2)'$	$x_1 \text{ XOR } x_2 = y_1 \text{ AND } y_2$
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0



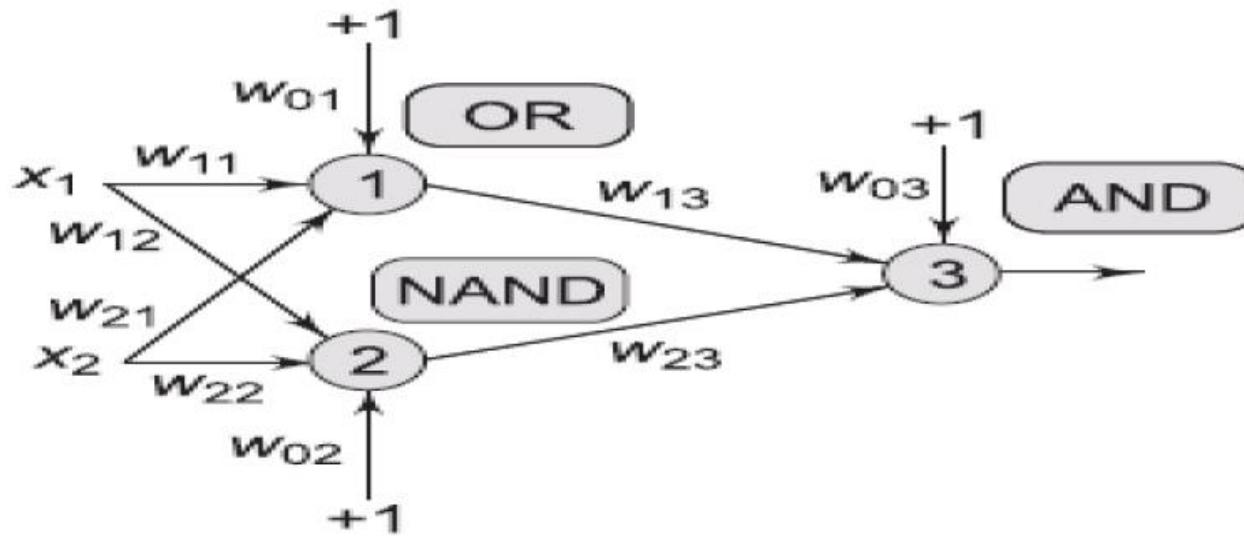
x_1	x_2	y_1
0	0	0
0	1	1
1	0	1

x_1	x_2	y_2
0	0	1
0	1	1
1	0	1



Multilayer Perceptron (MLP)

- Combining OR and NAND perceptrons using a logical AND (which can be implemented using a third binary perceptron), XOR problem can be solved



$$I = \text{step} \left\{ \sum_{i=1}^2 \omega_{i3} \left[\text{step} \left(\sum_{j=1}^2 \omega_{ij} x_j + \omega_{0j} \right) \right] + \omega_{03} \right\}$$

$\underbrace{\phantom{\sum_{i=1}^2 \omega_{i3}}}_{\text{perceptron } \#3}$ $\underbrace{\phantom{\sum_{j=1}^2 \omega_{ij} x_j}}_{\text{perceptron } \#1, \#2}$

Multilayer perceptron model

- 다수 / 다층 뉴런의 필요성
 - 단일 퍼셉트론은 복잡한 패턴 분류가 불가능. 선형 분리만 가능
 - 뉴런의 수를 늘리고 층을 추가하면 복잡한 구조의 의사결정 경계 (decision boundary)를 생성할 수 있음
- 다층퍼셉트론 Multilayer Perceptron, MLP
 - 여러 개의 퍼셉트론이 여러 층으로 쌓인 다층신경망 구조
 - 각 층 안에서는 뉴런간 연결이 없음. 인접한 두 층 사이의 뉴런 간에는 완전 연결됨

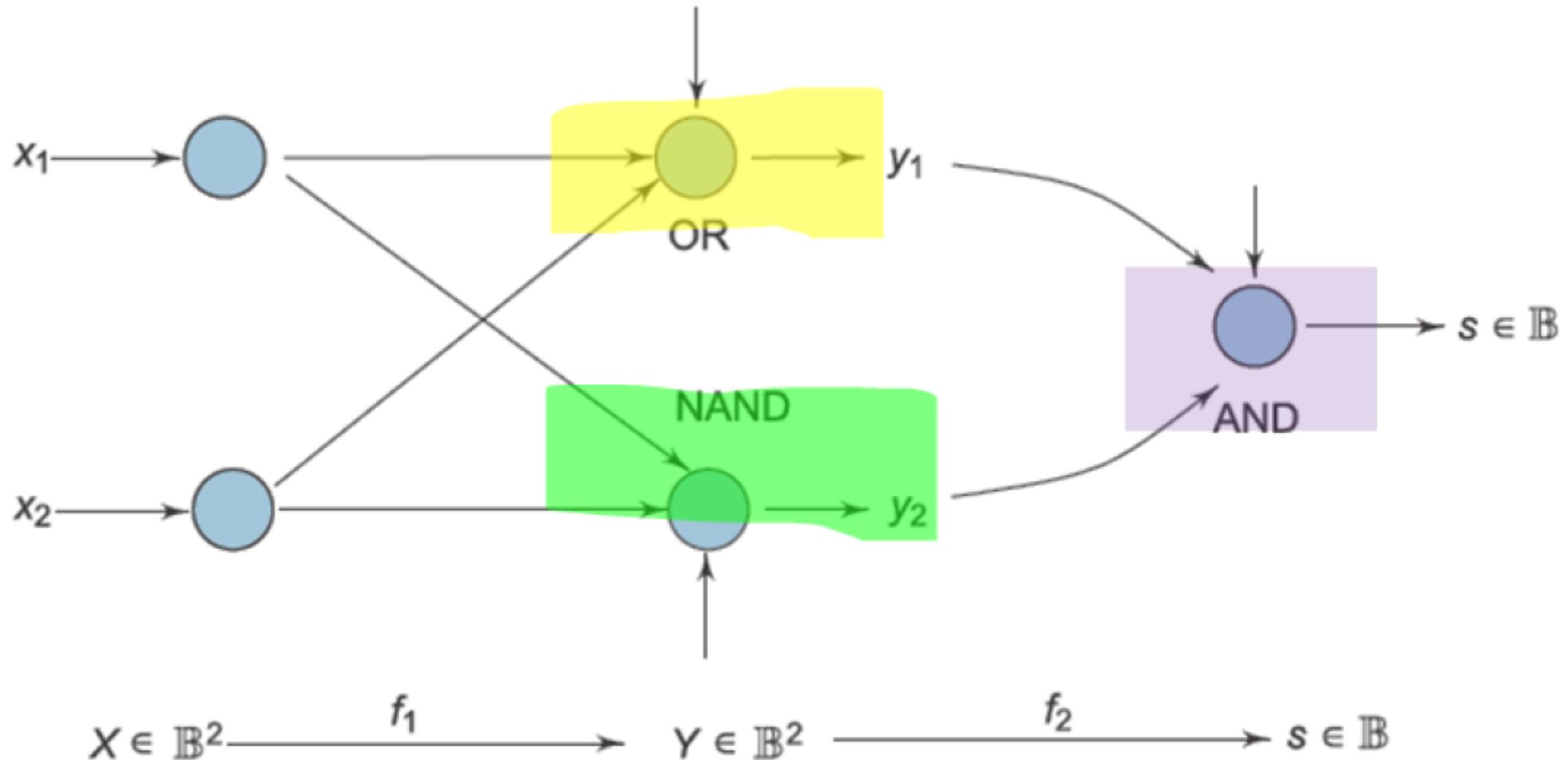
I To gain further insight...

- Since a perceptron can solve only a linearly separable problem, to solve a linearly non-separable problem **we have to make it linearly separable.**

How can this transformation be effected ?

- There are two ways:
 - If somehow we can **reduce the number of points p**, then it might be possible that the mapping from the reduced set of points into the range space may become linearly separable.
 - We might **increase the dimension** since the number of possible linear dichotomies then increases, and the probability that the linearly nonseparable problem at hand becomes linearly separable, increases.

Solution 1: Reduce the Number of Points



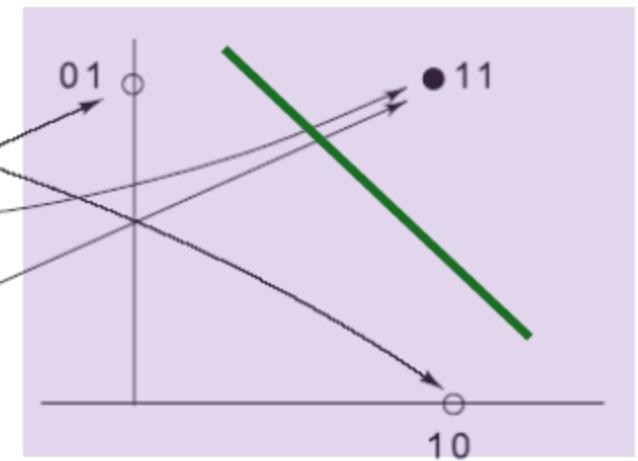
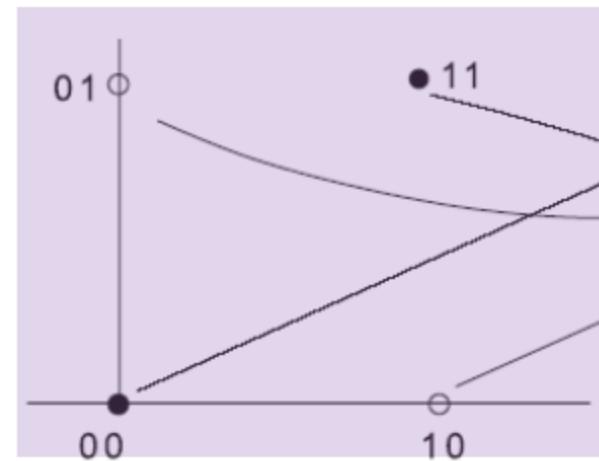
$$y_1 = x_1 + x_2; y_2 = \overline{x_1 x_2}$$

Solution 1: Reduce the Number of Points

$$y_1 = \underline{x_1 + x_2}; y_2 = \underline{\overline{x_1}x_2}$$

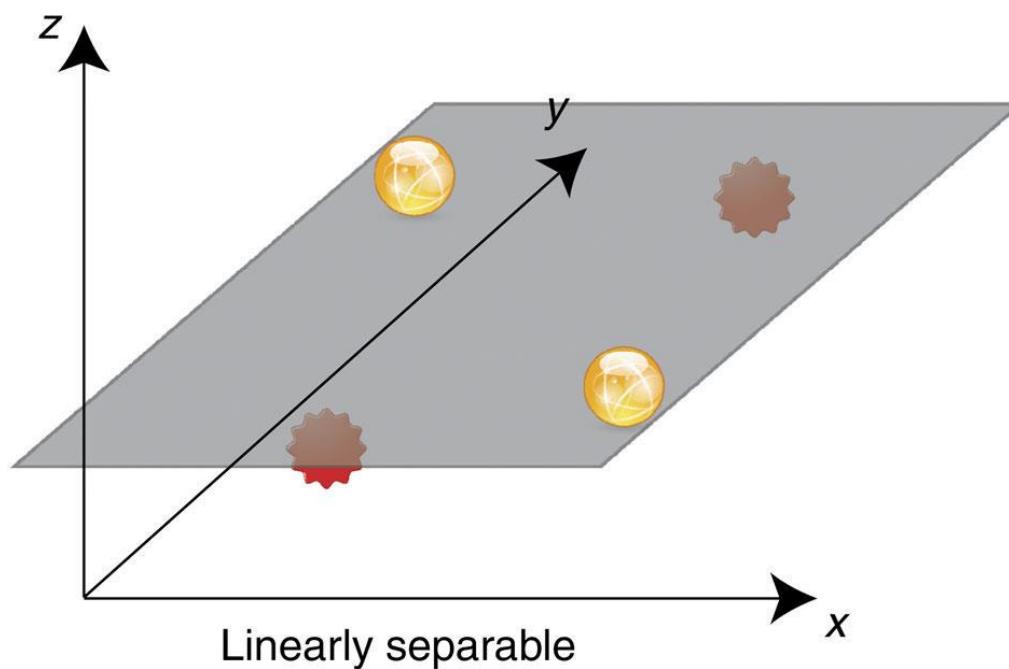
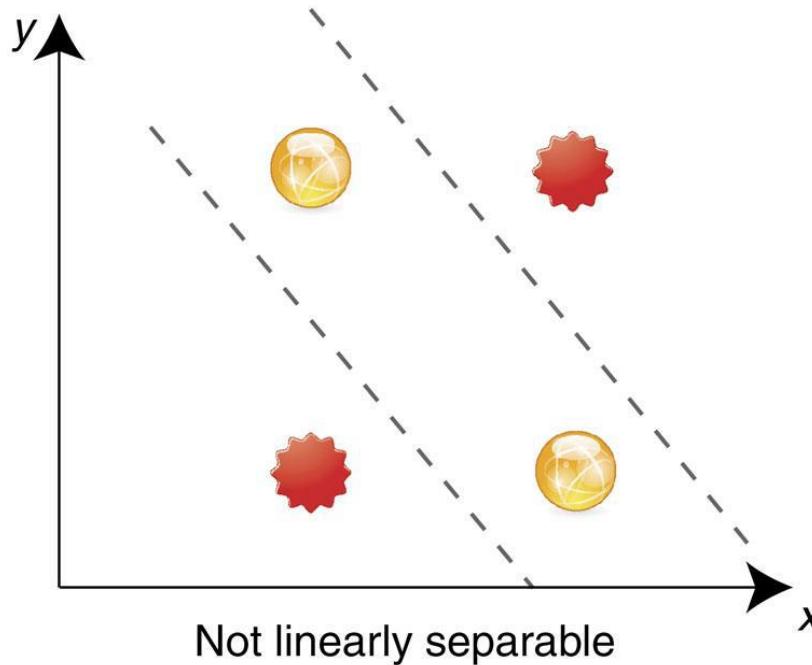
x_1	x_2	y_1	y_2	s
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

Original input New input Output



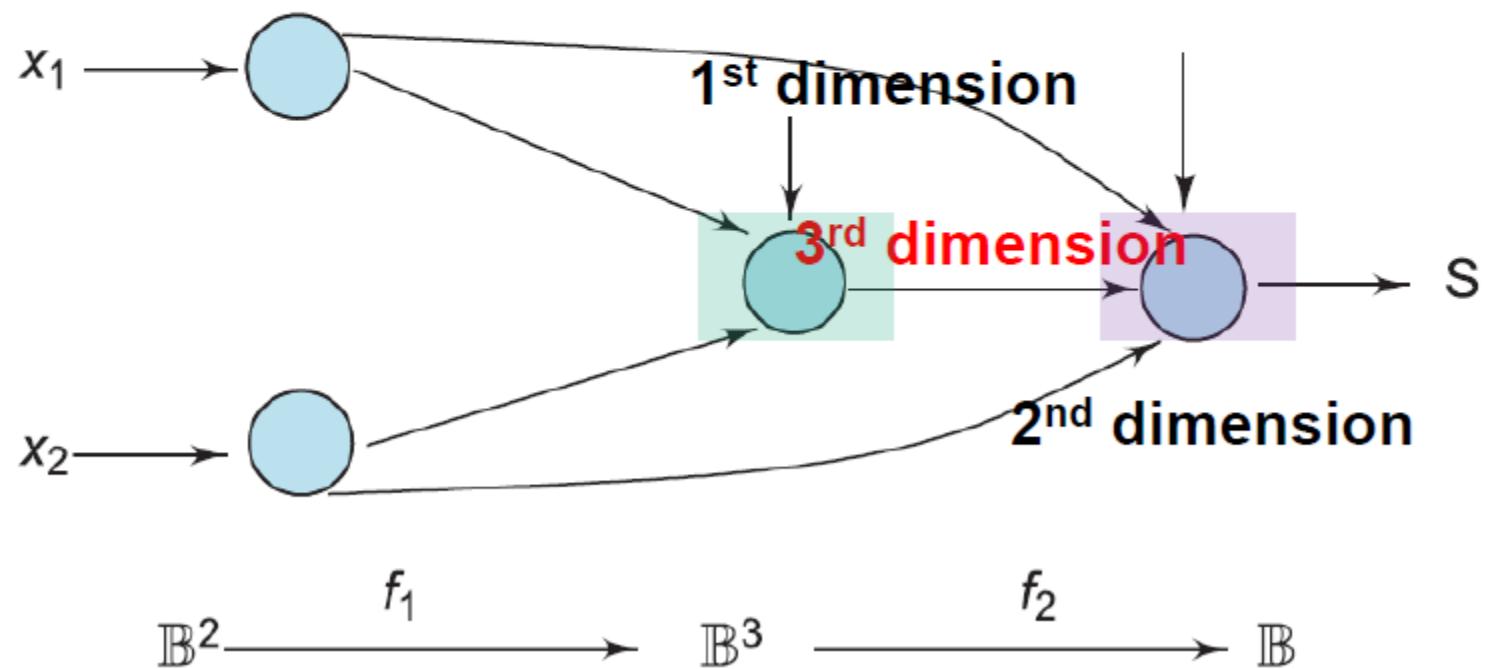
4 points reduce to 3 points

Solution 2: Increase the Dimension



Solution 2: Increase the Dimension

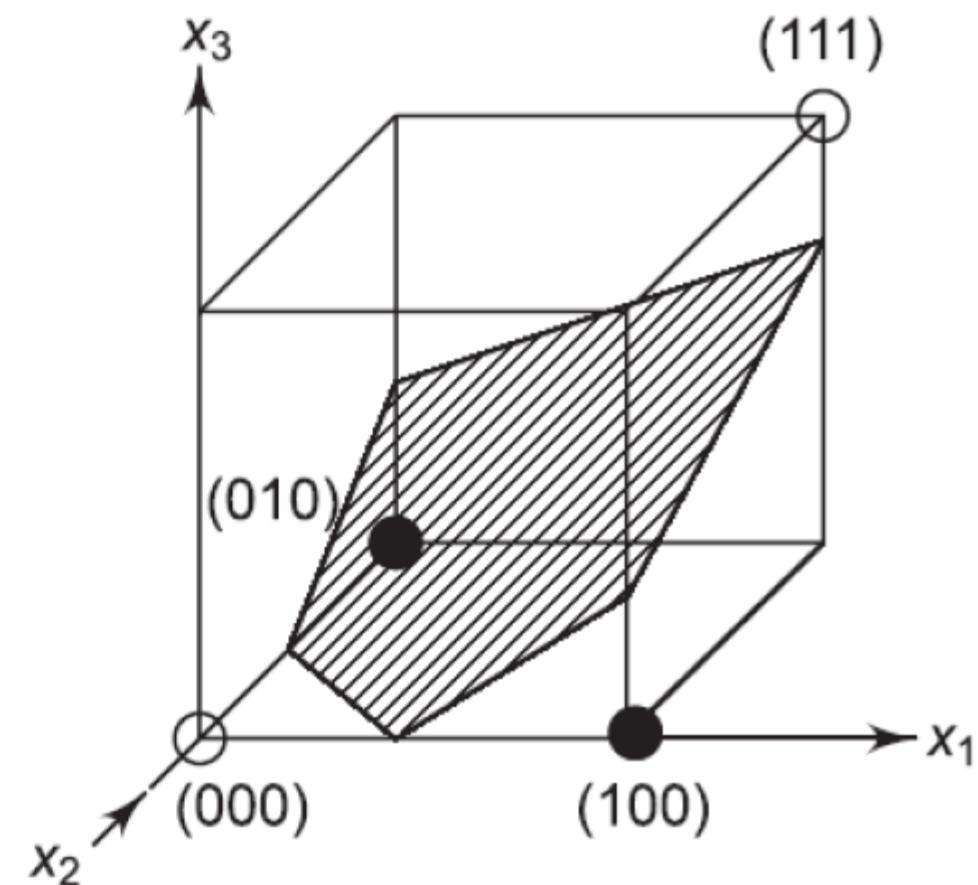
- We are now looking for an appropriate mapping $f_1 : \mathbb{B}^2 \rightarrow \mathbb{B}^3$ such that $f_2 : \mathbb{B}^3 \rightarrow \mathbb{B}$ is linearly separable.



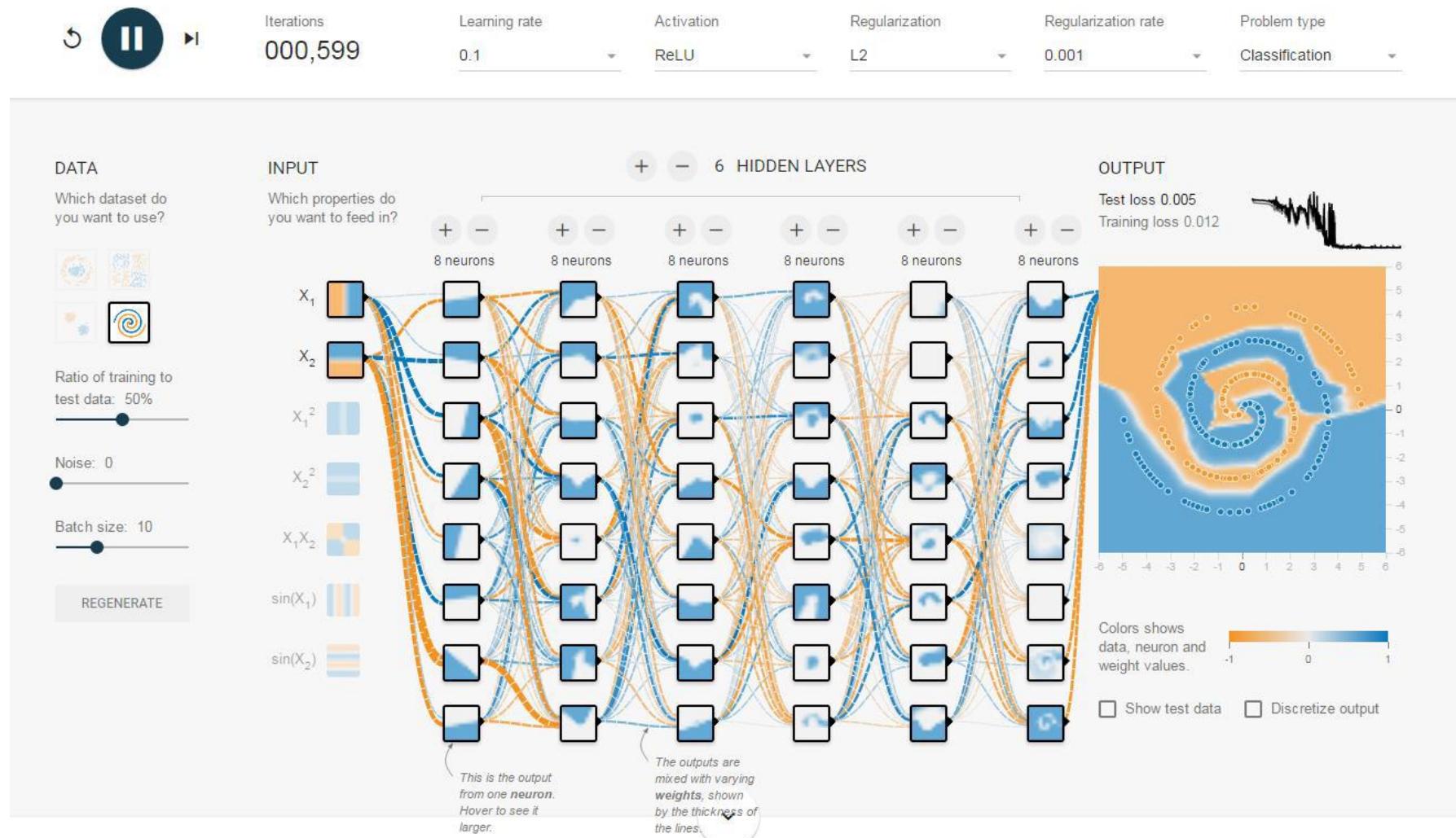
The AND Function Added As The Third Dimension

Third Dimension

x_1	x_2	$x_3 (= x_1 x_2)$	$x_1 \oplus x_2$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

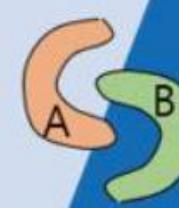
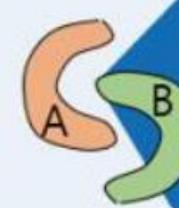
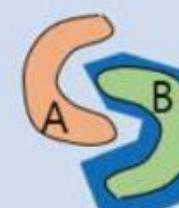


The Geometry Of MLP



The Geometry Of MLP

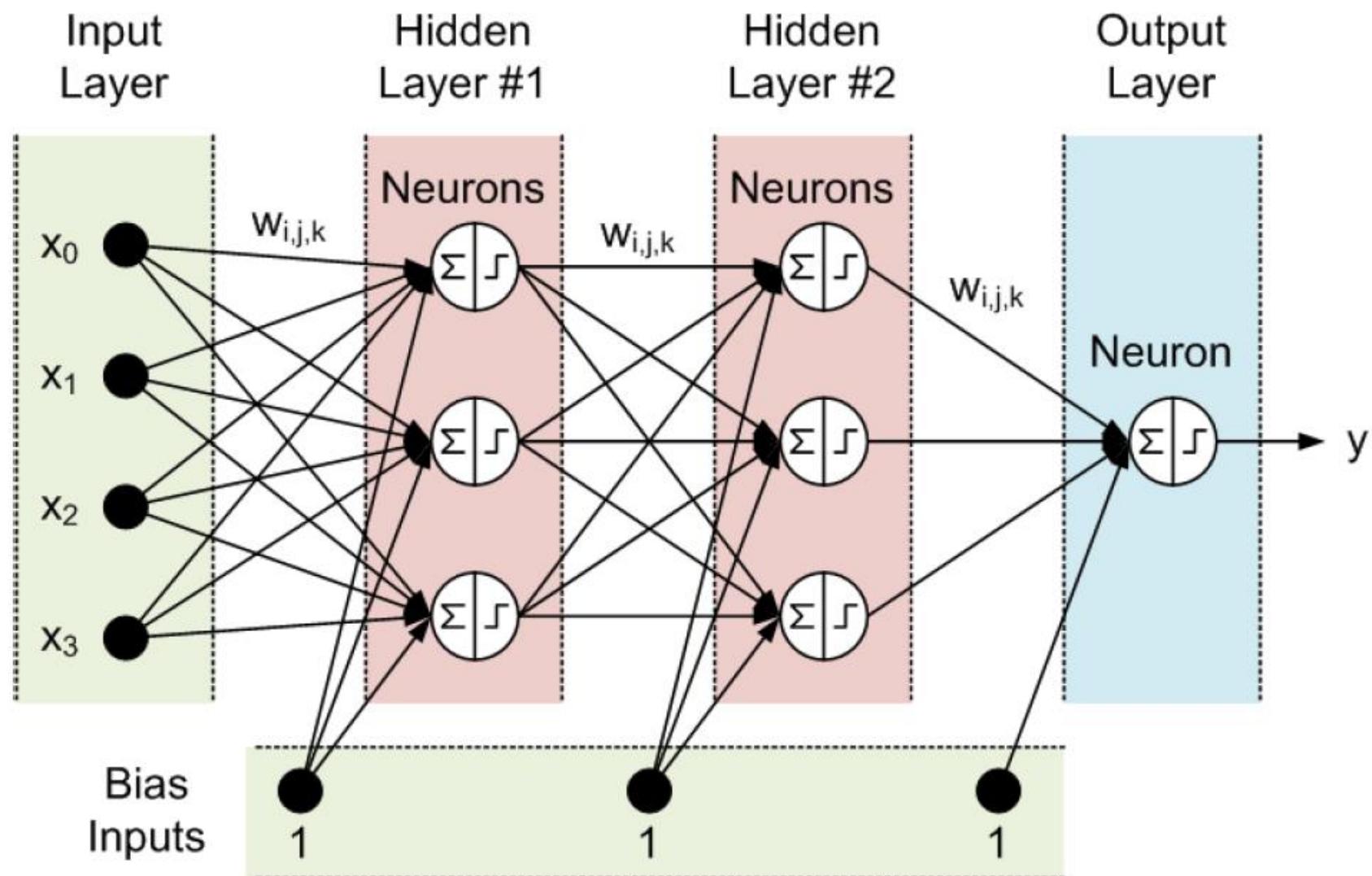
- 다층퍼셉트론의 혁신
 - 은닉 뉴런층 추가
 - 주어진 데이터의 입력값과 출력값 모두에 대응되지 않음
 - 데이터의 정보를 계층적으로 처리하게 됨
 - 시그모이드 활성 함수를 사용
 - 비선형 모델로 확장 가능
 - 은닉 뉴런층의 학습을 가능하게 하는 **오류역전파 기법 error backpropagation** 개발로 이어짐

Structure	Regions	XOR	Meshed Regions
Single layer	Halfplane bounded by hyperplane		
Two layers	Convex Open or closed regions		
Three layers	Arbitrary (limited by # of nodes)		

Multiple boundaries needed
(e.g. XOR problem)
→ **Multiple units**

More complex regions needed
(e.g. Polygons)
→ **Multiple layers**

Multilayer Perceptron Model(MLP)



● Linear neuron

● Sigmoidal neuron

● Sigmoid/Softmax/Linear neuron

Multilayer Perceptron Model(MLP)

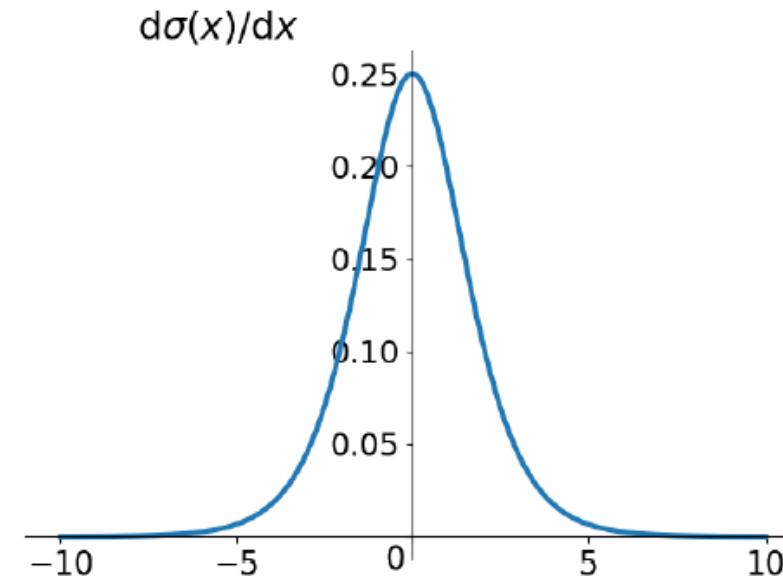
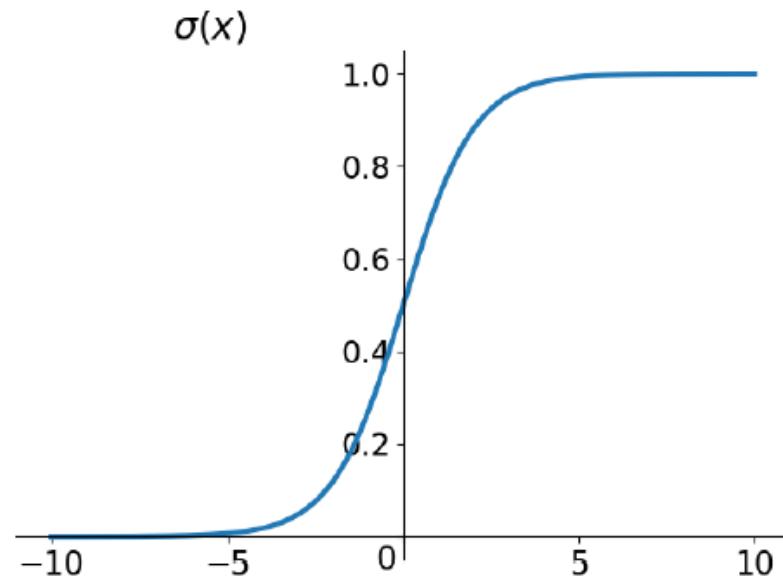
❖ 다층 신경망

- **다층 신경망의 구조**
 - 다층 신경망의 층에는 입력층, 출력층, 은닉층이 있다.
 - 각 층에는 각각 자신만의 특정 함수가 있다.
- **입력층**
 - 외부에서 받아들인 입력 신호를 은닉층의 모든 뉴런으로 보낸다.
 - 계산을 위한 뉴런은 거의 들어 있지 않다.
- **출력층**
 - 은닉층에서 출력 신호, 즉 자극 패턴을 받아들이고 전체 신경망의 출력 패턴을 정한다.
- **은닉층**
 - 입력의 특성을 파악한다. 뉴런의 가중치는 입력 패턴에 숨겨져 있는 특성을 나타낸다.
 - 출력층이 출력 패턴을 정할 때, 이 특성을 사용한다.
 - 은닉층은 목표 출력을 ‘숨기고’ 있다. 즉, 은닉층의 목표 출력은 해당 층에서 자체적으로 결정된다.
 - 신경망에 은닉층이 두 개 이상 들어갈 수 있다.

Multilayer Perceptron Model(MLP)

- Activation function in **hidden nodes**:
Sigmoid/Logistic Function (Traditional)

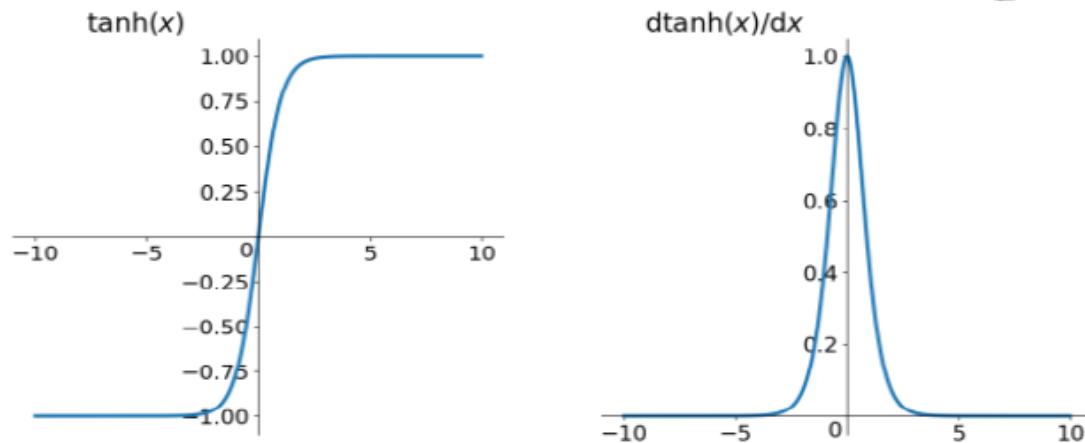
$$\sigma(z) = \frac{1}{1+e^{-z}}$$



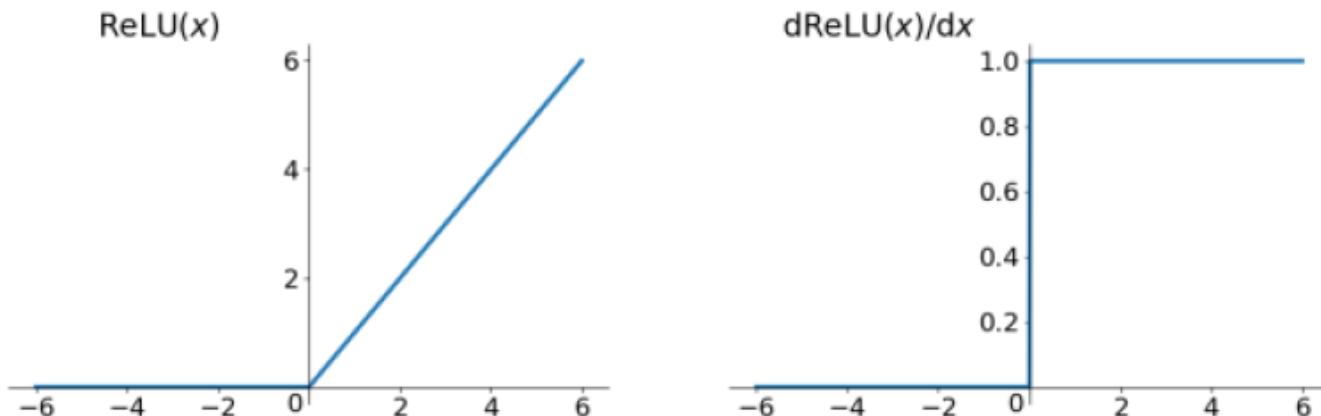
- Why?

Multilayer Perceptron Model(MLP)

- Hyperbolic Tangent: $\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ **(Traditional)**



- Rectifier Linear: $\text{ReLU} = \max(0, x)$ **(since 2012)**



Multilayer Perceptron Model(MLP)

$$s_h = \sum_i w_{hi}^{(1)} x_i$$

은닉 유닛층의 h번째 노드 출력값
(활성함수 전)

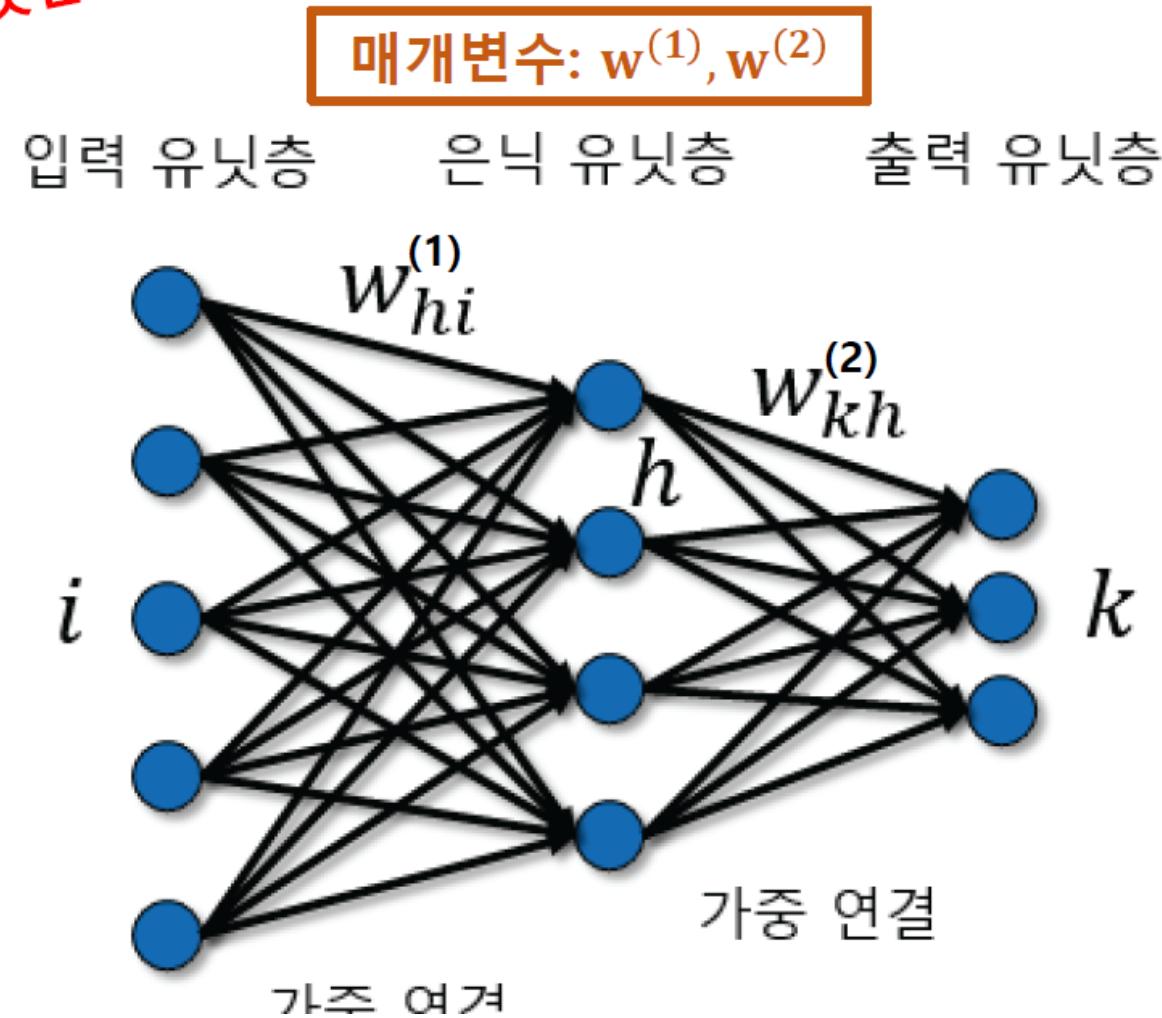
입력 유닛층의 i번째 노드 출력값

은닉 유닛층의 h번째 노드 출력값

입력 유닛층의 i번째 노드와
은닉 유닛층 h번째 노드를 잇는 가중치

$$f_{MLP_h} = activation(s_h)$$

$$\text{ex) } f_{MLP_h} = \sigma(s_h) = \frac{1}{1 + \exp(-s_h)}$$



Multilayer Perceptron Model(MLP)

- 모델의 수식 구성

- 출력값 벡터

$$f_{out} = (f_{out1}, f_{out2}, \dots, f_{outK}) = f(\mathbf{x}; \mathbf{w})$$

- 출력값의 계산과정

- 은닉 뉴런의 출력값 x_h

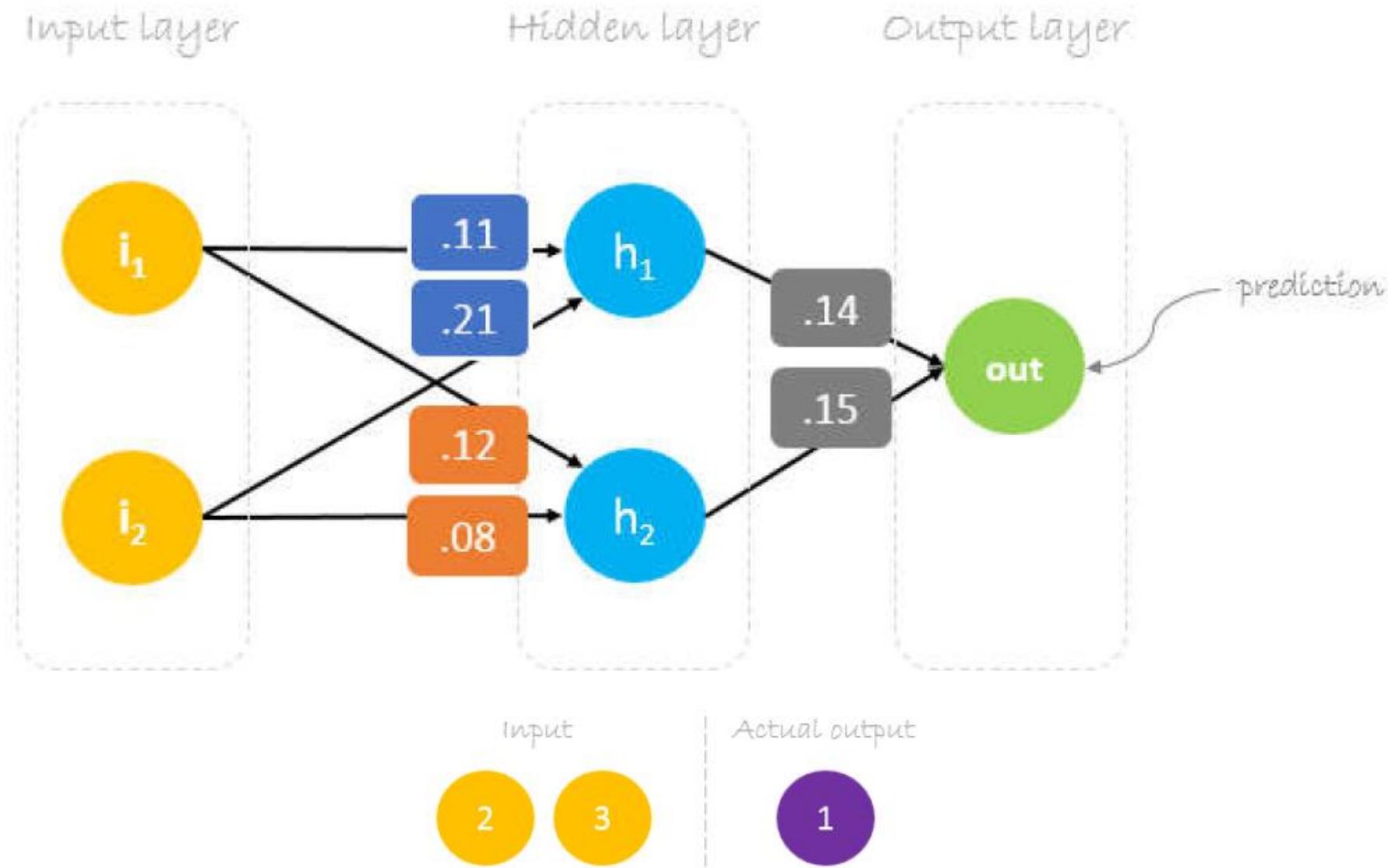
$$s_h = \sum_{i=0}^I w_{hi}^{(1)} x_i, \quad x_h = \sigma(s_h) = \frac{1}{1+\exp(-s_h)}$$

- 출력 뉴런의 출력값 f_k

$$s_k = \sum_{h=0}^H w_{kh}^{(2)} x_h, \quad f_k = \sigma(s_k) = \frac{1}{1+\exp(-s_k)}$$

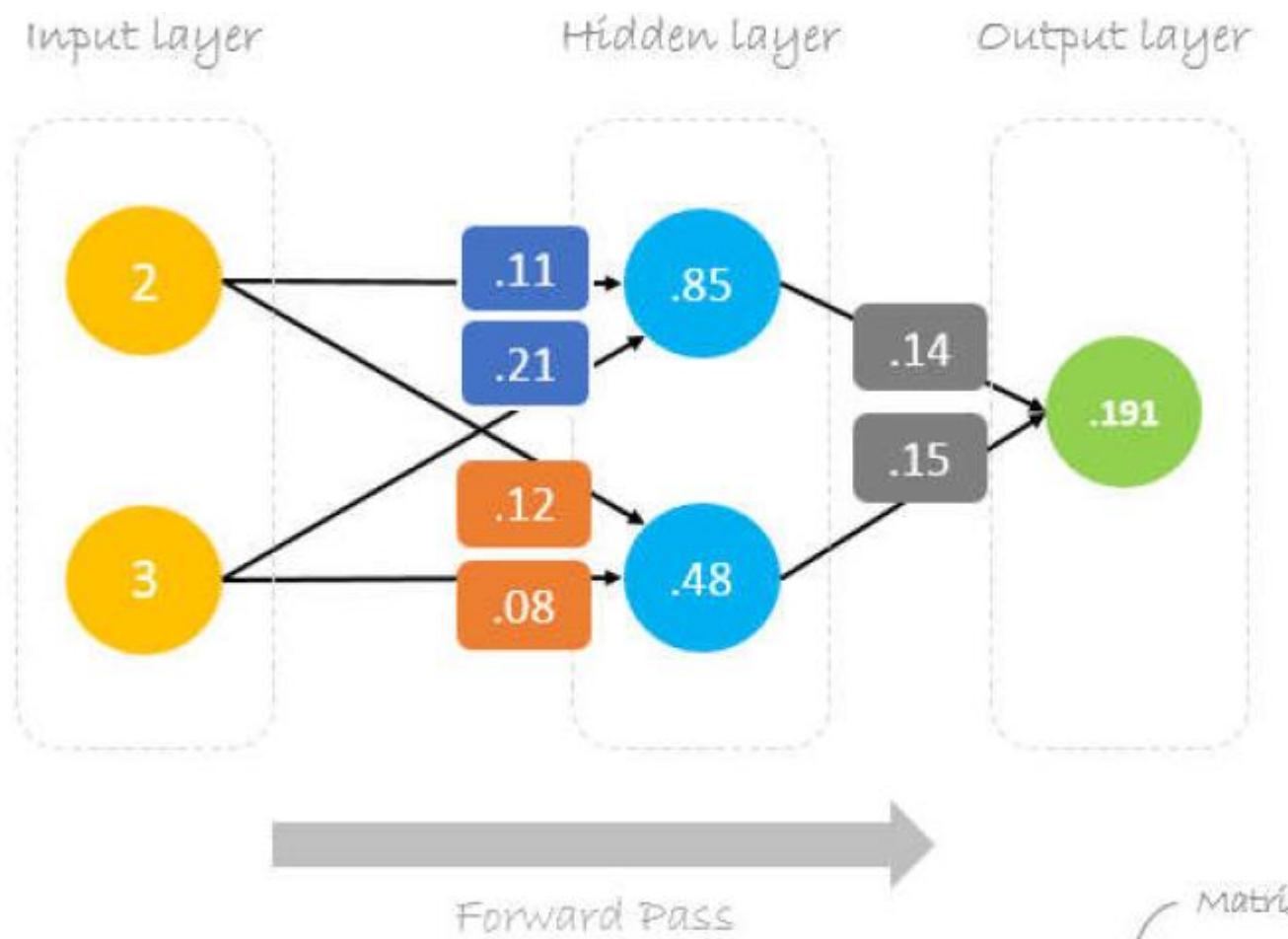
$$f_{out} = f_k(\mathbf{x}, \mathbf{w}) \quad \text{은닉층의 출력}$$
$$= \sigma \left(\sum_{h=0}^H w_{kh}^{(2)} \sigma \left(\sum_{i=0}^I w_{hi}^{(1)} x_i \right) \right)$$

Example 2



현재 주어진 신경망 모델에 input과 output이 다음과 같을 때, 각 node의 결과를 구하여라 활성화함수는 없다고 가정한다.

Example 2 Solution



$$[2 \ 3] \cdot \begin{bmatrix} 0.11 & 0.12 \\ 0.21 & 0.08 \end{bmatrix} = [0.85 \ 0.48] \cdot \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} = [0.191]$$

$$2 \times 0.11 + 3 \times 0.21 = 0.85$$

$$2 \times 0.12 + 3 \times 0.08 = 0.48$$

$$0.85 \times 0.14 + 0.48 \times 0.15 = 0.191$$

Matrix multiplication

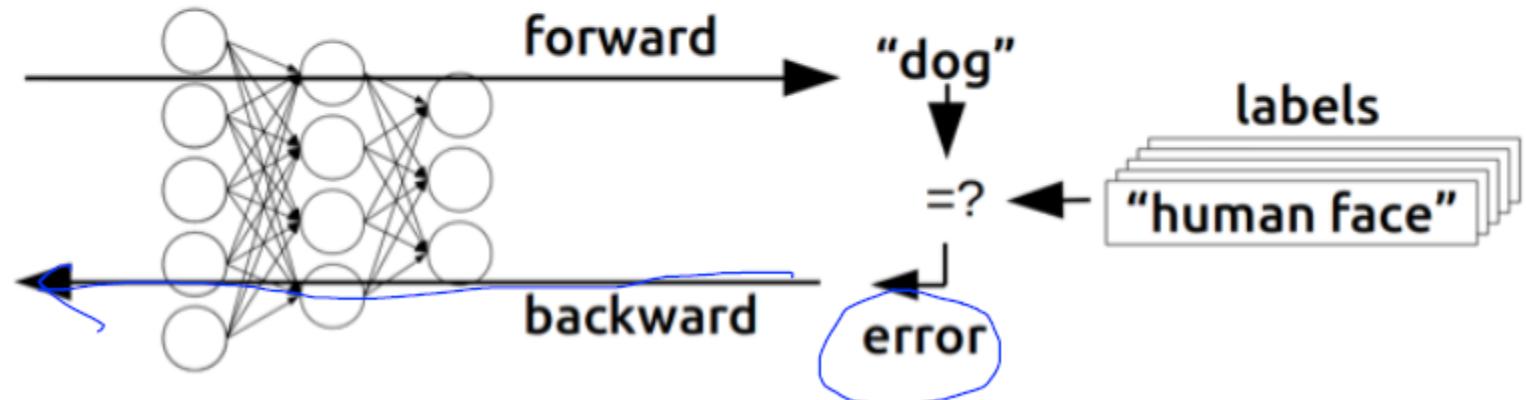
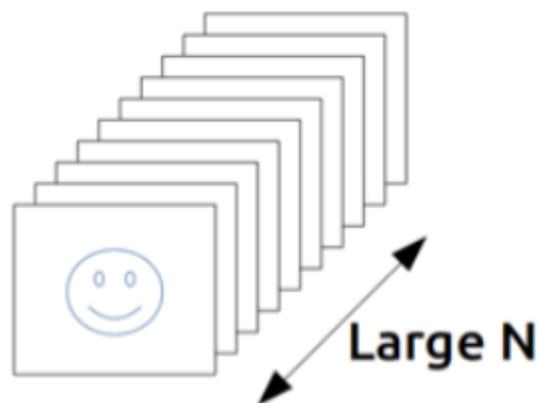
Details

Back Propagation

I Back Propagation

(1974, 1982 by Paul Werbos, 1986 by Hinton)

Training



Introduction

- We have learned the basic of Backpropagation indeed – Gradient Descent
- Backpropagation is an optimization method for all major neural networks (MLP, Convolution NNs, Recurrent Networks etc).

Backpropagation – Upgraded GD

iteration	Action
0	Initialize network
1	Take first instance , do forward pass, error computation, backpropogate errors and update all weights in MLP.
2	Take second instance , do forward pass, error computation, backpropogate errors and update all weights in MLP.
3	Take third instance , do forward pass, error computation, backpropogate errors and update all weights in MLP.
.	
.	
.	
Q (total number of training data)	Take Qth instance , do forward pass, error computation, backpropogate errors and update all weights in MLP.
(One epoch is done)	

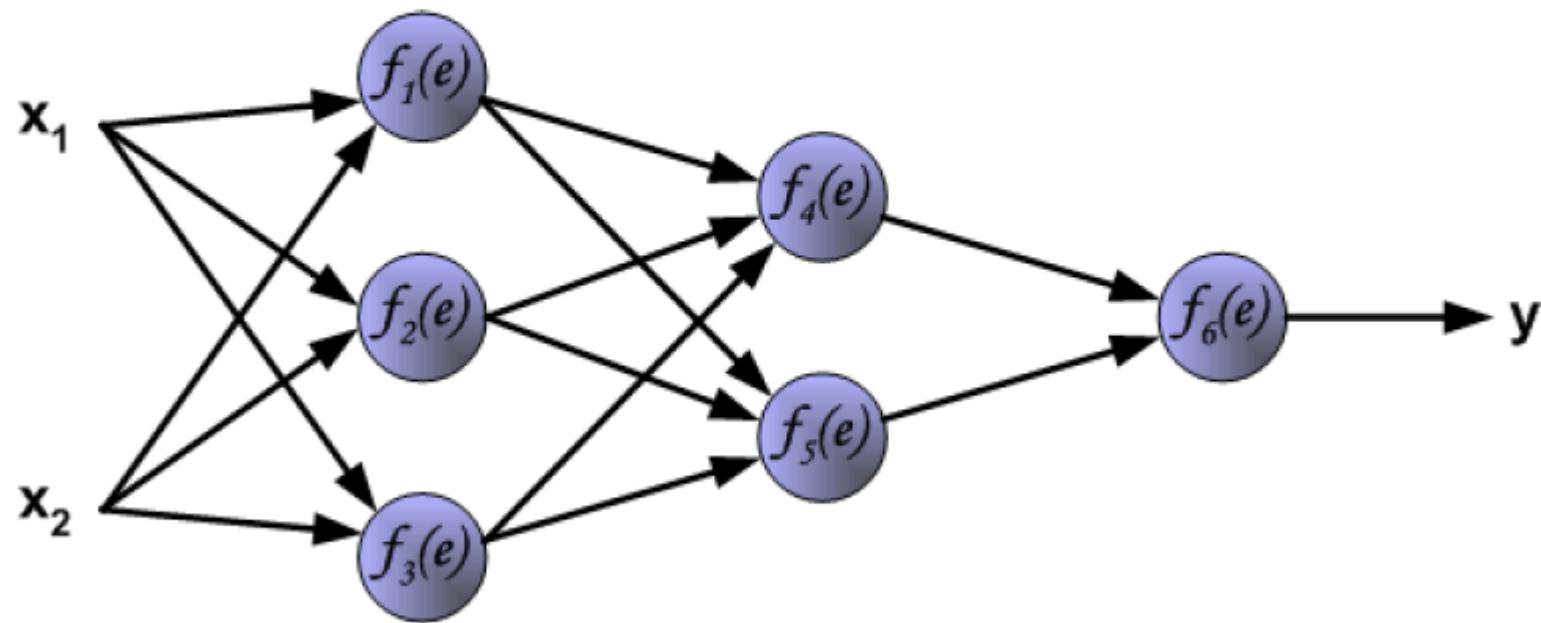
ASSUME SGD setup



Repeat all over again and again until error reduces to certain value or achieve maximum number of iteration.

Illustration

- $w_{(xm)n}$ - weights of connections between network input x_m and neuron n in hidden layer.
- y_n - output signal of neuron n.
- $f_i(e)$ - sigmoid function



A two-hidden layers MLP

(Source: Adapted from http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html)

I Back Propagation Algorithm

- 출력 뉴런 j 에 대한 가중치 벡터 \mathbf{w}_j 변경 η : 학습률 learning rate

$$\mathbf{w}_j \leftarrow \mathbf{w}_j + \Delta \mathbf{w}_j \quad \text{활성함수가 시그모이드인 경우}$$
$$\Delta \mathbf{w}_j = \eta(y_j - f_j) f_j (1 - f_j) \mathbf{x}_j$$

- $x_i^{(l-1)} = f_i^{(l-1)}$: $l-1$ 층 i 번째 뉴런의 출력값

- $s_j^{(l)}$: l 층 j 번째 뉴런의 출력값(활성함수 전)

$$s_j^{(l)} = \mathbf{w}_j^{(l)} \cdot \mathbf{x}_j^{(l-1)} = \sum_i w_{ji}^{(l)} x_i^{(l-1)} \rightarrow \frac{\partial s_j^{(l)}}{\partial \mathbf{w}_j^{(l)}} = x^{(l-1)}$$

$$x_j^{(l)} = f_j^{(l)} = \sigma(s_j^{(l)}) \rightarrow \frac{\partial f_j^{(l)}}{\partial s_j^{(l)}} = f_j^{(l)}(1 - f_j^{(l)})$$

Back Propagation Algorithm

- 출력 뉴런 j 에 대한 가중치 벡터 w_j 변경
 - 학습 데이터에 대한 손실함수

$$L_N = \sum_{d=1}^N L_d, \quad L_d = \frac{1}{2} \left(y^{(d)} - f_{MLP}^{(d)} \right)^2$$

- l 번째 층 j 번째 뉴런의 델타값 $\delta_j^{(l)}$
 - 해당 뉴런의 (활성함수 전) 출력값에 대한 오차 변화율의 음수값
 - 교정할 오차 수치를 의미함

$$\delta_j^{(l)} \triangleq -\frac{\partial L_d}{\partial s_j^{(l)}}$$

I Back Propagation Algorithm

- 델타값의 계산

- 연쇄규칙을 사용
- 목표 출력값이 주어지는 출력뉴런에 대한 델타값
($l =$ 마지막층의 증수일 때)

$$\frac{\partial L_d}{\partial s_j^{(l)}} = \frac{\partial L_d}{\partial f_j^{(l)}} \cdot \frac{\partial f_j^{(l)}}{\partial s_j^{(l)}} = - (y^{(l)} - f_j^{(l)}) f_j^{(l)} (1 - f_j^{(l)})$$

$$\delta_j^{(l)} = - \frac{\partial L_d}{\partial s_j^{(l)}} = (y^{(l)} - f_j^{(l)}) f_j^{(l)} (1 - f_j^{(l)})$$

I Back Propagation Algorithm

- 델타값의 계산

- l 번째 층에 있는 j 번째 뉴런에 연결된 값들의 경사도

$$s_j^{(l)} = \mathbf{w}_j^{(l)} \cdot \mathbf{x}_j^{(l-1)}$$

$$\begin{aligned}\frac{\partial L_d}{\partial \mathbf{w}_j^{(l)}} &= \frac{\partial L_d}{\partial f_j^{(l)}} \cdot \frac{\partial f_j^{(l)}}{\partial s_j^{(l)}} \cdot \frac{\partial s_j^{(l)}}{\partial \mathbf{w}_j^{(l)}} = - (y^{(l)} - f_j^{(l)}) \frac{\partial f_j^{(l)}}{\partial s_j^{(l)}} \mathbf{x}_j^{(l-1)} \\ &= - (y^{(l)} - f_j^{(l)}) f_j^{(l)} (1 - f_j^{(l)}) \mathbf{x}_j^{(l-1)} \\ &= - \delta_j^{(l)} \mathbf{x}_j^{(l-1)}\end{aligned}$$

Back Propagation Algorithm

- 델타값의 계산

- l 번째 층이 은닉층인 경우 목표 출력값 대신

$l+1$ 번째 층의 델타값 $\delta^{(l+1)}$ 들을 활용하여 델타값 $\delta^{(l)}$ 을 계산하고
가중치 수정에 활용함

$$\begin{aligned}\frac{\partial L_d}{\partial s_j^{(l)}} &= \sum_{k \in \text{Upper}(j)} \frac{\partial L_d}{\partial s_k^{(l+1)}} \cdot \frac{\partial s_k^{(l+1)}}{\partial f_j^{(l)}} \cdot \frac{\partial f_j^{(l)}}{\partial s_j^{(l)}} \\ &= \sum_{k \in \text{Upper}(j)} -\delta_k^{(l+1)} w_{kj}^{(l+1)} f_j^{(l)} (1 - f_j^{(l)}) \\ &= -f_j^{(l)} (1 - f_j^{(l)}) \sum_{k \in \text{Upper}(j)} \delta_k^{(l+1)} w_{kj}^{(l+1)}\end{aligned}$$

$$\frac{\partial L_d}{\partial w_j^{(l)}} = \boxed{\frac{\partial L_d}{\partial s_j^{(l)}}} \cdot \frac{\partial s_j^{(l)}}{\partial w_j^{(l)}} = \boxed{-\delta_j^{(l)}} \mathbf{x}_j^{(l-1)}$$

I Back Propagation Algorithm

- 델타값의 계산

- l 층 j 번째 뉴런에 연결된 가중치 벡터의 변경식

$$\mathbf{w}_j \leftarrow \mathbf{w}_j + \Delta \mathbf{w}_j$$
$$\Delta \mathbf{w}_j = -\eta \frac{\partial L_d}{\partial \mathbf{w}_j^{(l)}} = \eta \delta_j^{(l)} \mathbf{x}_j^{(l-1)}$$

$$\delta_j^{(l)} = \begin{cases} \frac{\partial f_j^{(l)}}{\partial s_j^{(l)}} \frac{\partial L_d}{\partial f_j^{(l)}} & \text{for } j \in \text{Outputs} \\ \frac{\partial f_j^{(l)}}{\partial s_j^{(l)}} \sum_{k \in \text{Upper}(j)} \delta_k^{(l+1)} w_{kj}^{(l+1)} & \text{for } j \in \text{Hiddens} \end{cases}$$

$(l = \text{마지막 층의 층수일 때})$

I Back Propagation Algorithm

- 델타값의 계산

- l 층 j 번째 뉴런에 연결된 가중치 벡터의 변경식

$$\mathbf{w}_j \leftarrow \mathbf{w}_j + \Delta \mathbf{w}_j$$
$$\Delta \mathbf{w}_j = -\eta \frac{\partial L_d}{\partial \mathbf{w}_j^{(l)}} = \eta \delta_j^{(l)} \mathbf{x}_j^{(l-1)}$$

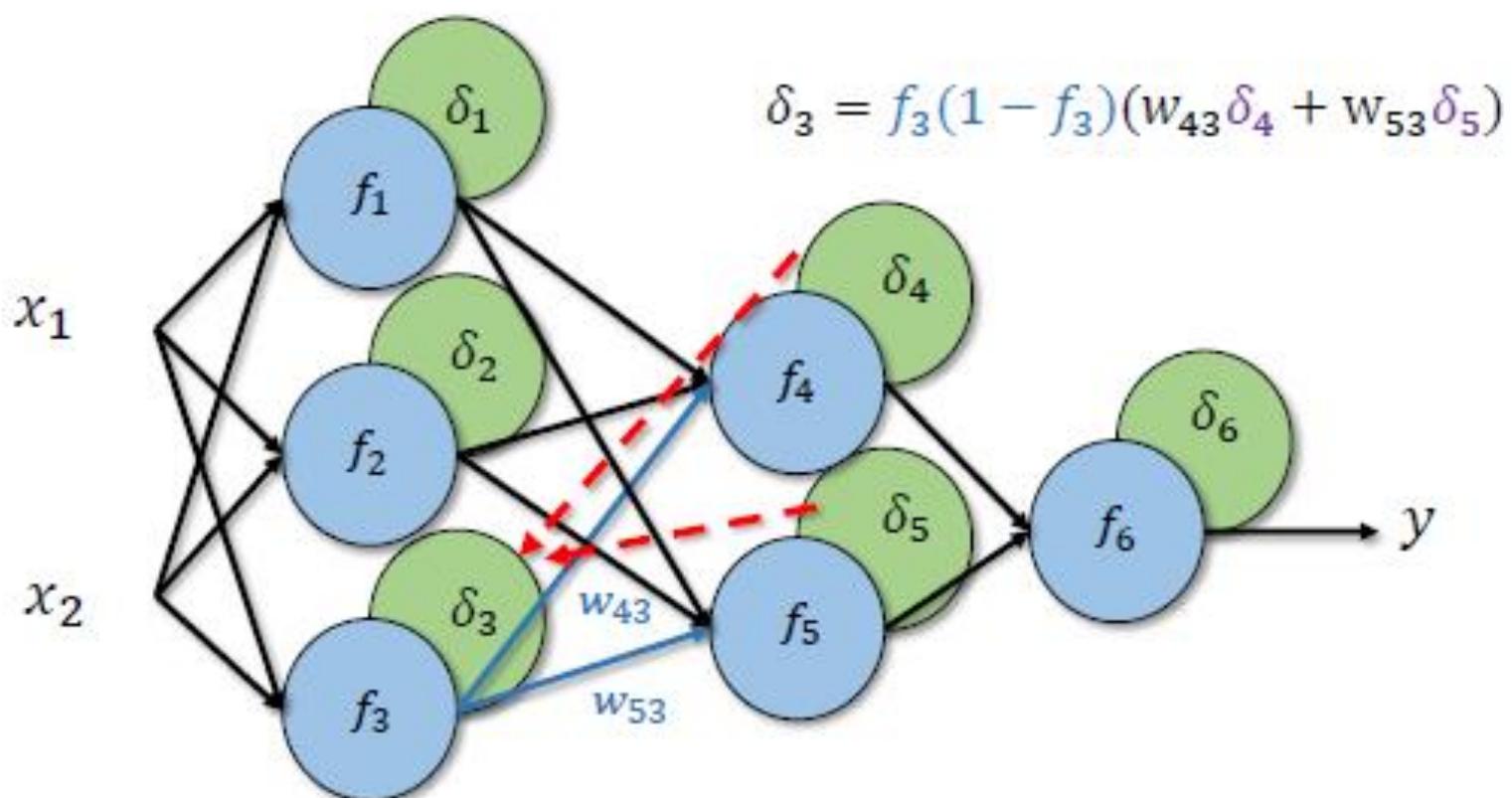
시그모이드 활성함수와

오차제곱 손실함수를 사용하는 경우

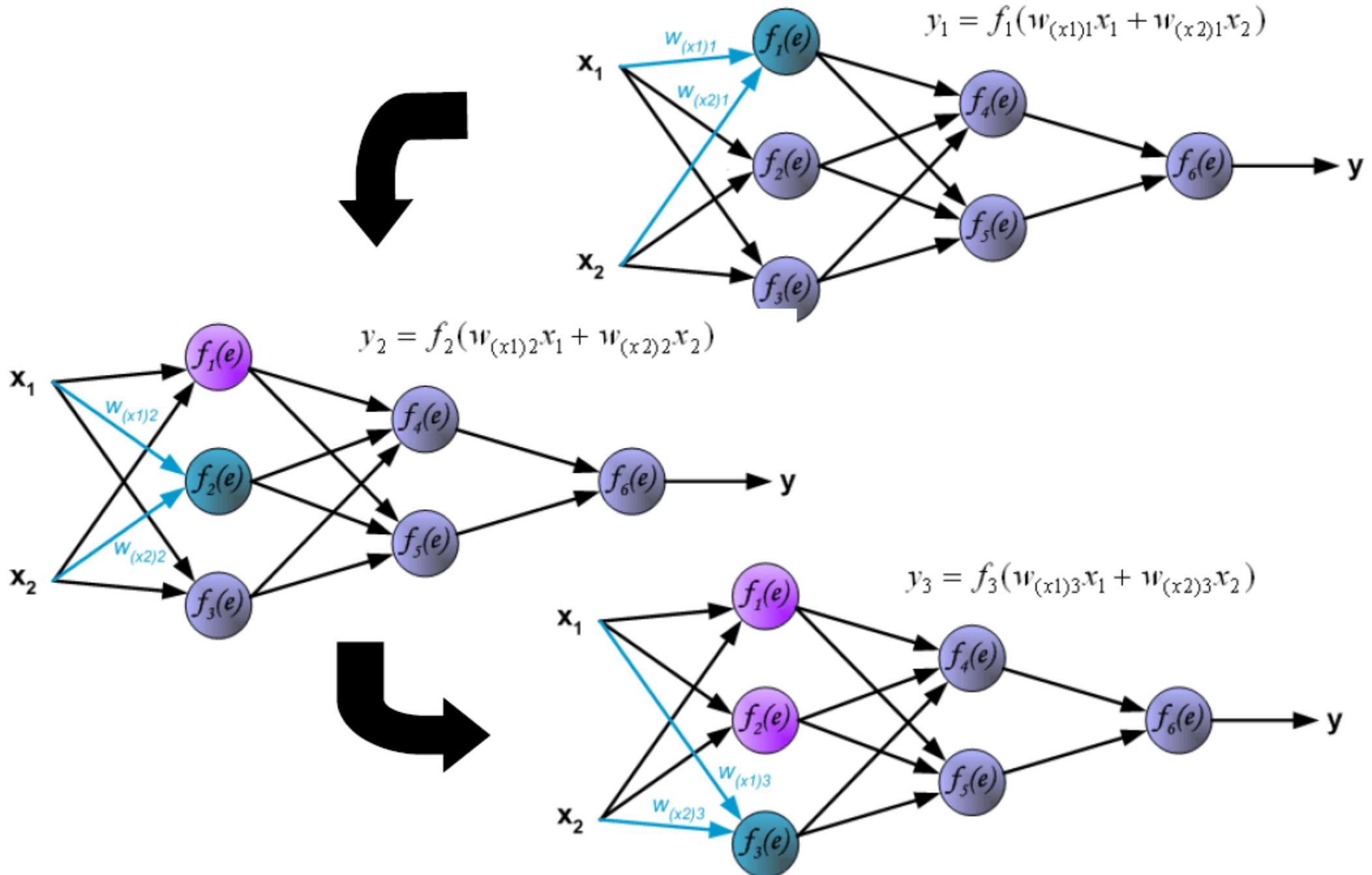
$$\delta_j^{(l)} = \begin{cases} f_j^{(l)} (1 - f_j^{(l)}) (y_j - f_j^{(l)}) & \text{for } j \in \text{Outputs} \\ f_j^{(l)} (1 - f_j^{(l)}) \sum_{k \in \text{Upper}(j)} \delta_k^{(l+1)} w_{kj}^{(l+1)} & \text{for } j \in \text{Hidden} \end{cases}$$

Back Propagation Algorithm

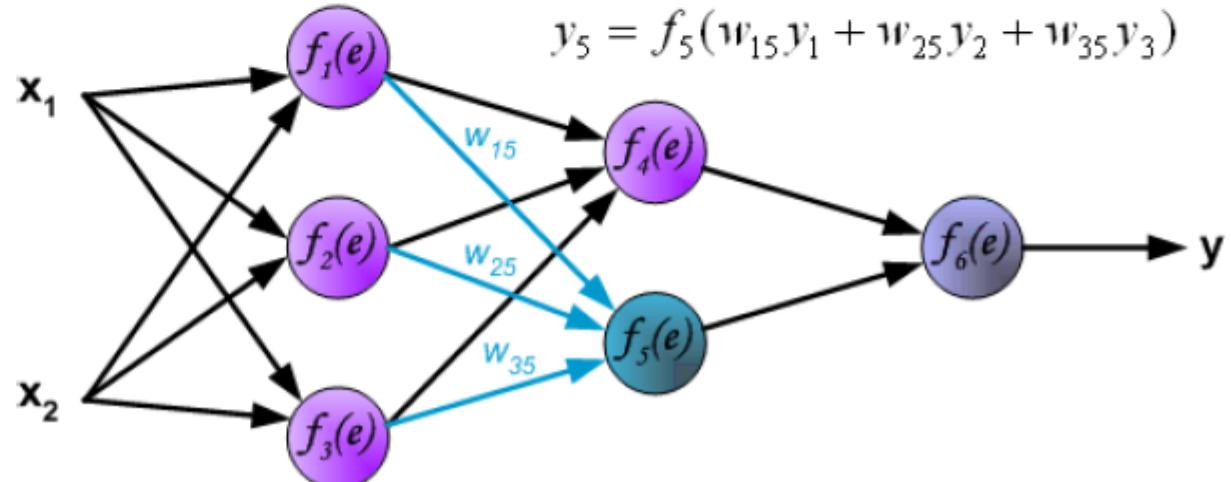
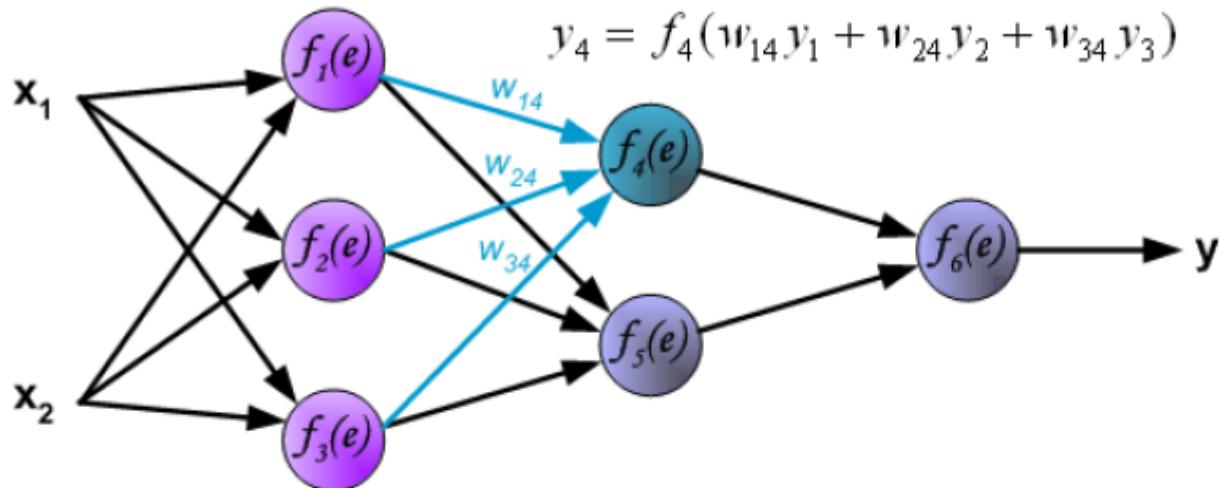
- 델타값의 계산



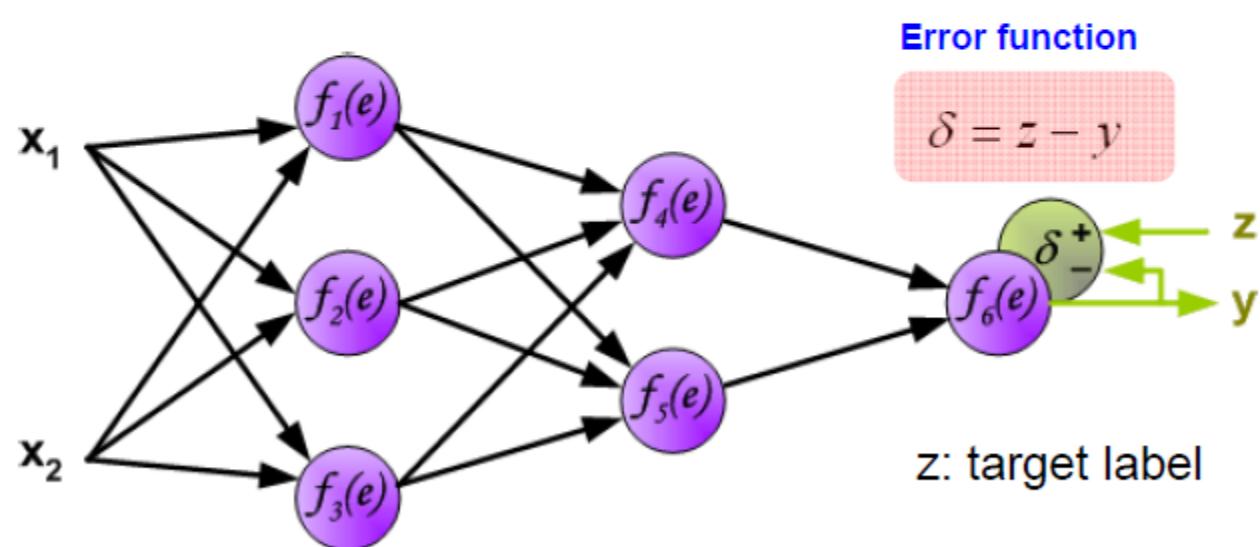
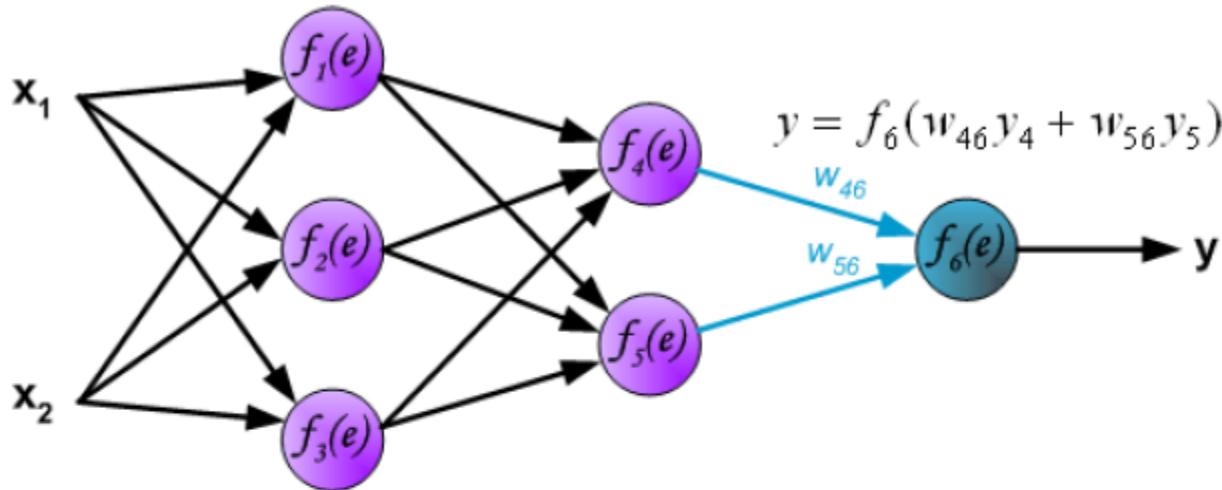
Forward pass: Input Layer



Forward pass: Hidden Layer

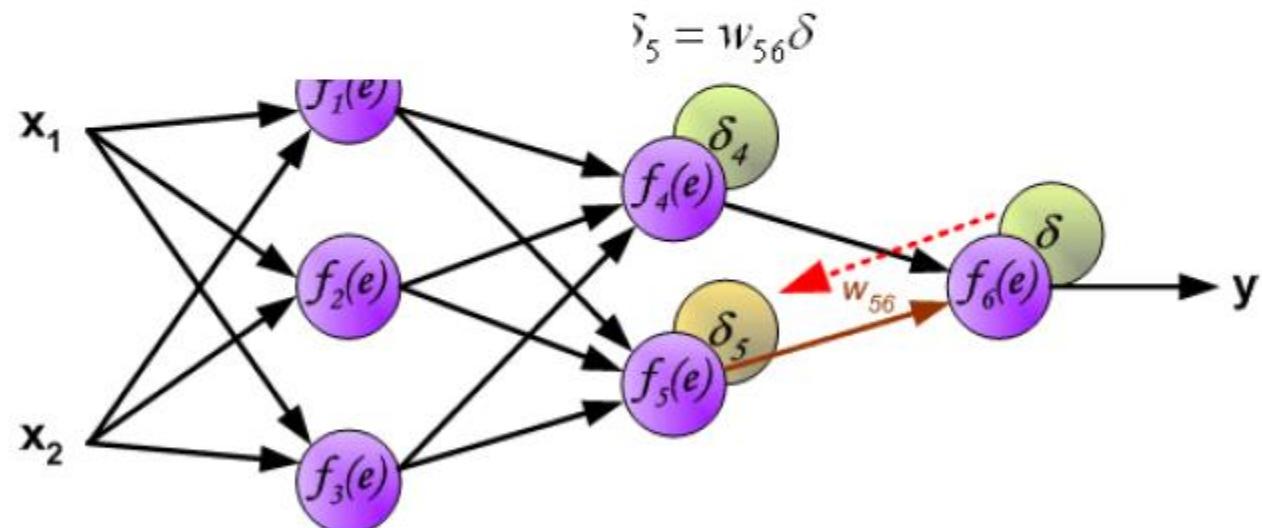
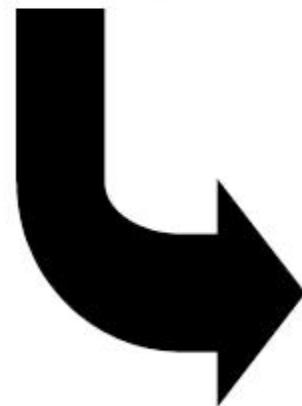
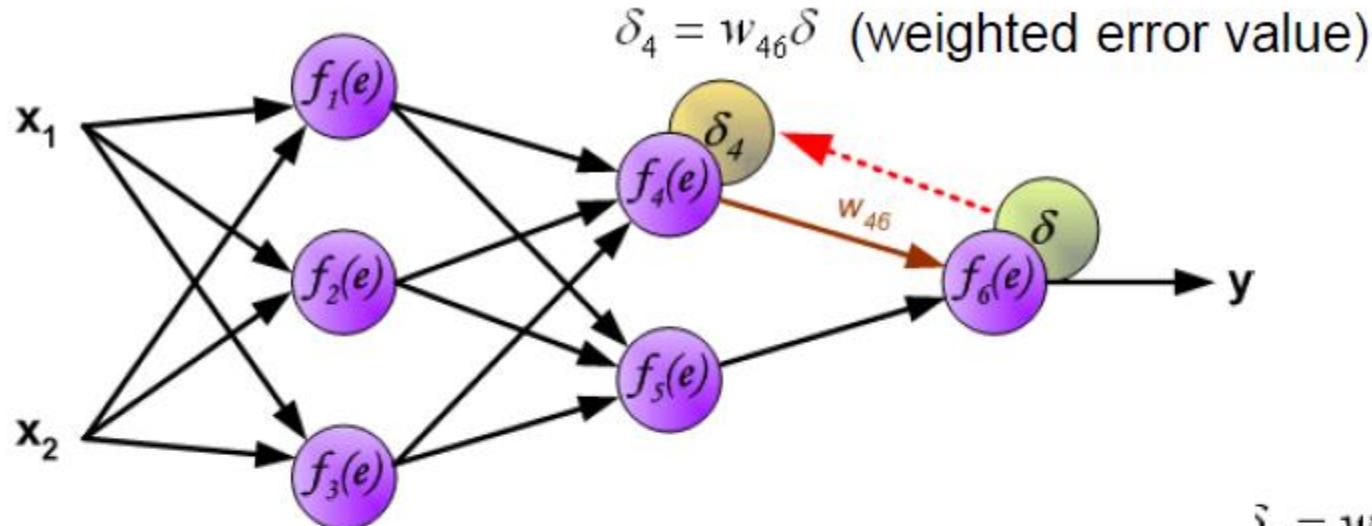


Forward pass: Output Layer



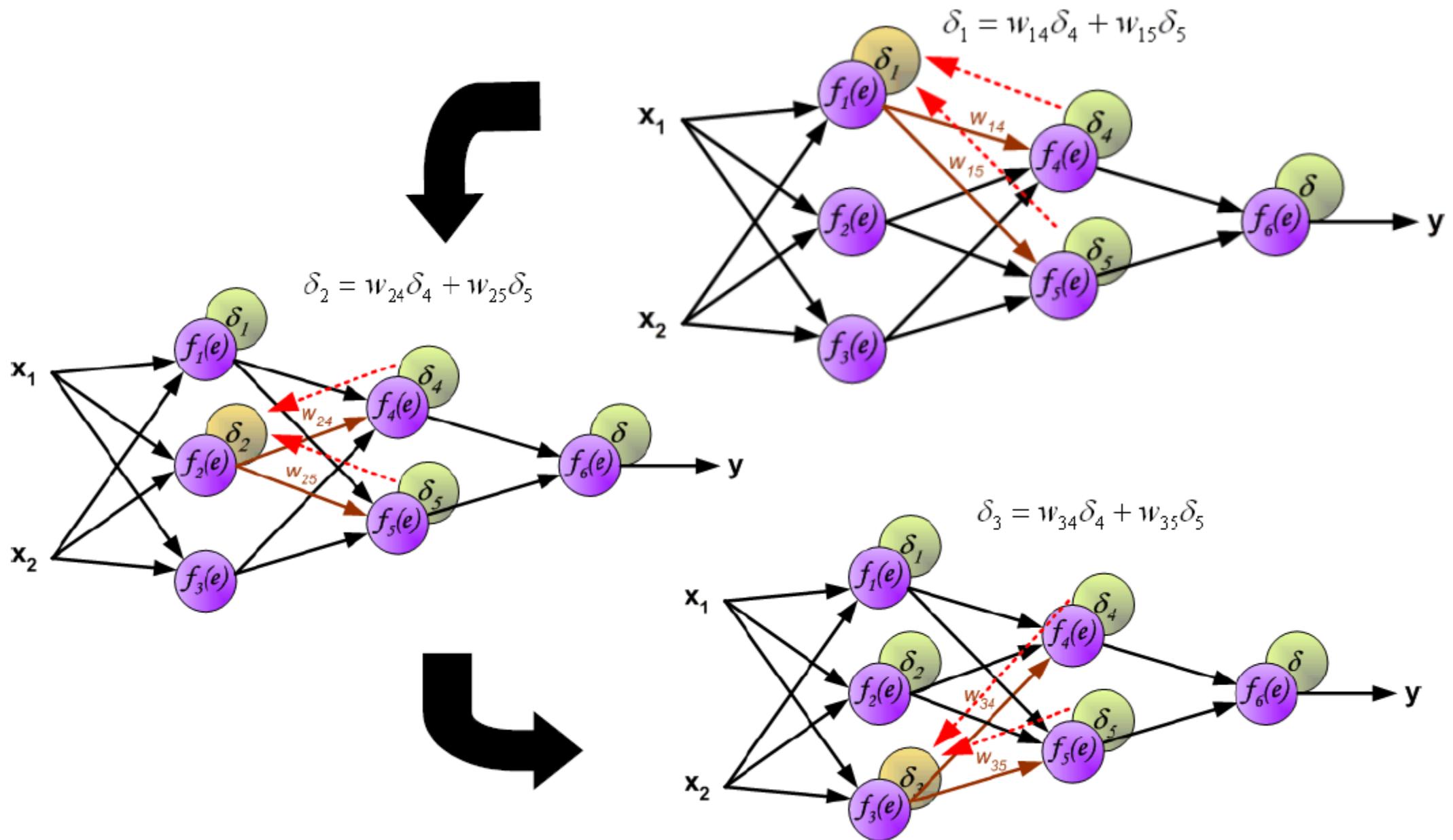
2nd Stage: Error Computation

Backward pass : Output Layer

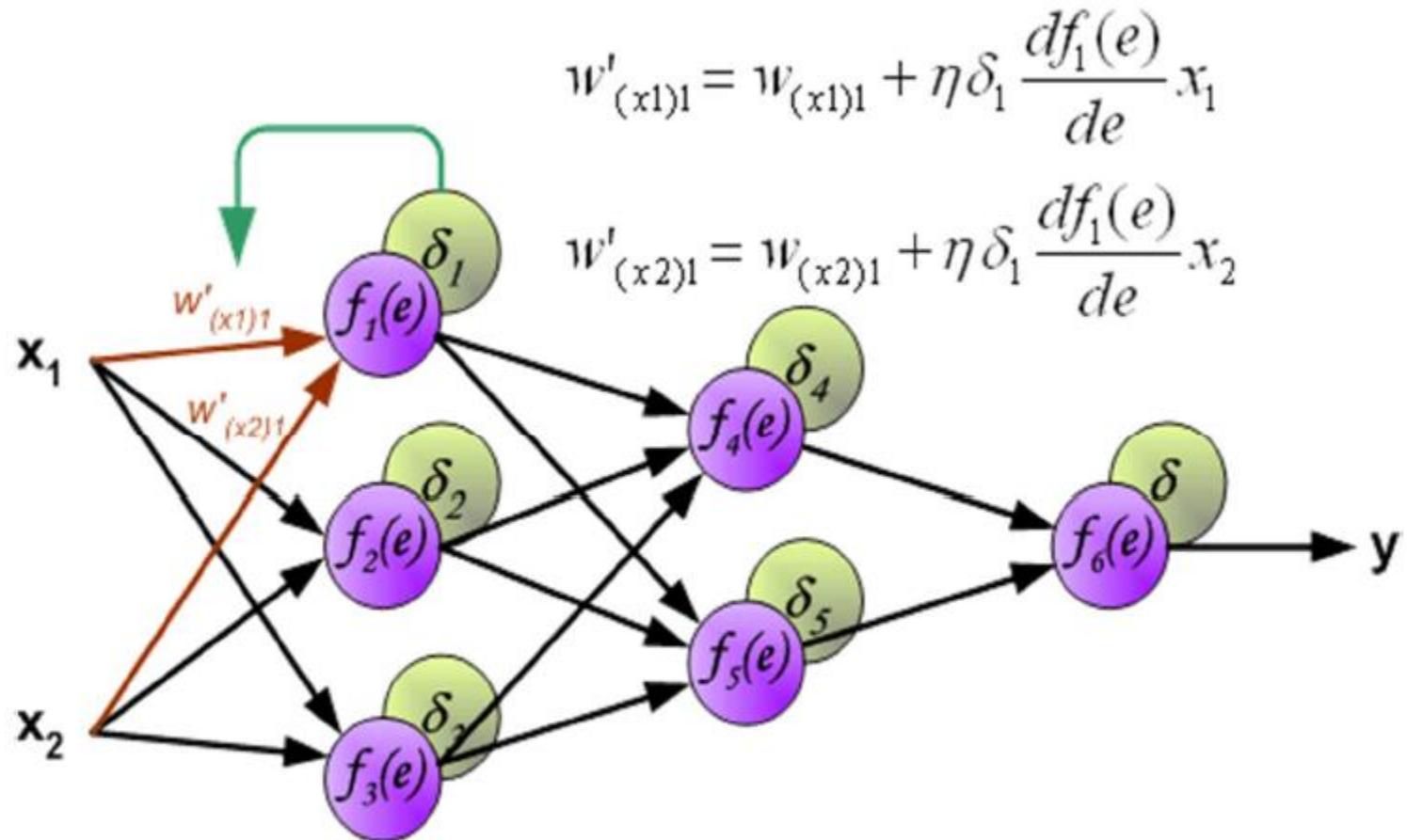


3rd Stage: Error Backpropagation

Backward pass : Hidden Layer

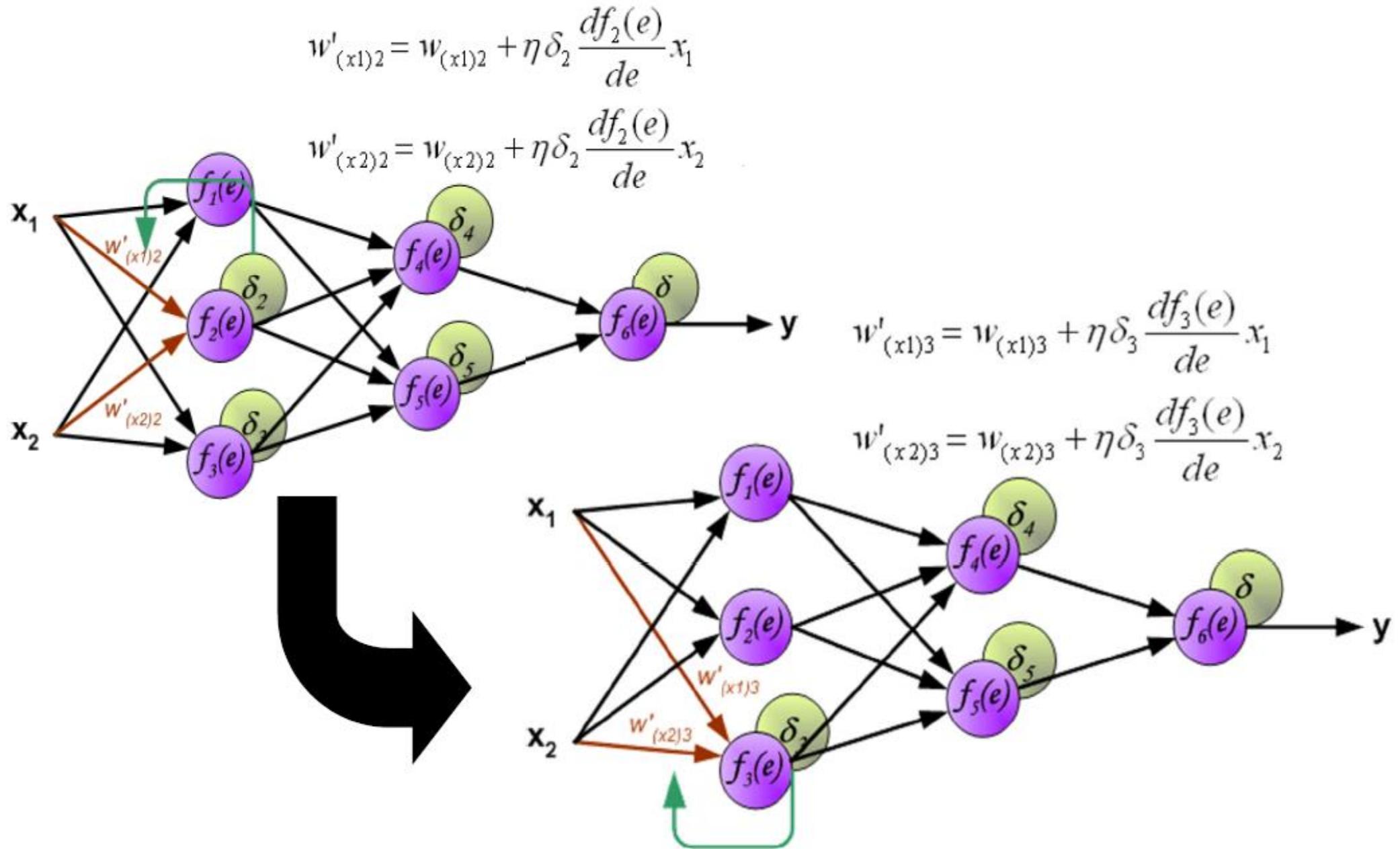


Weights Updating



4nd Stage: Weights Update

Weights Updating



Weights Updating

$$w'_{14} = w_{14} + \eta \delta_4 \frac{df_4(e)}{de} y_1$$

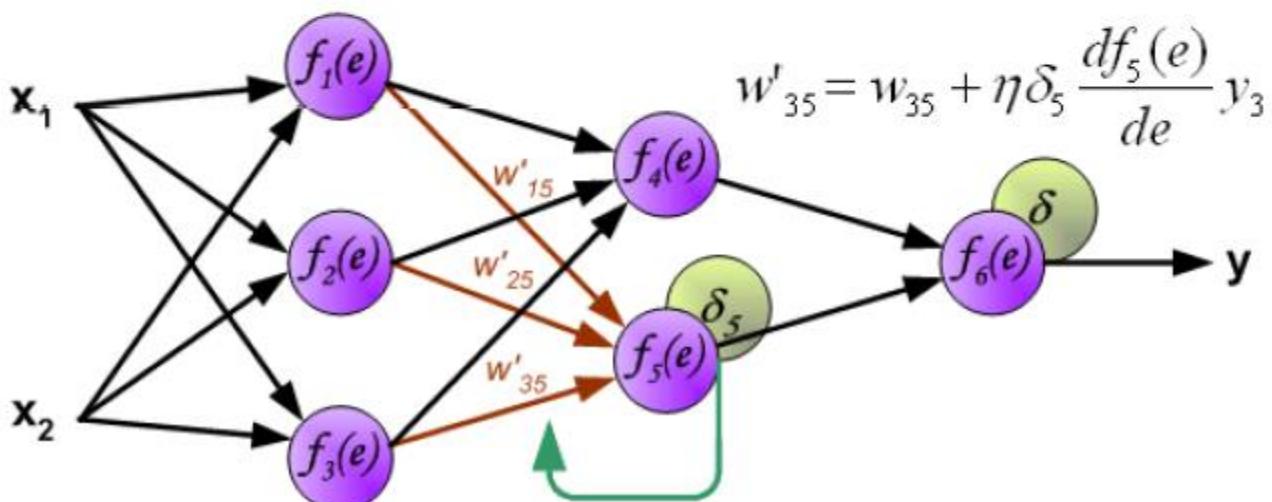
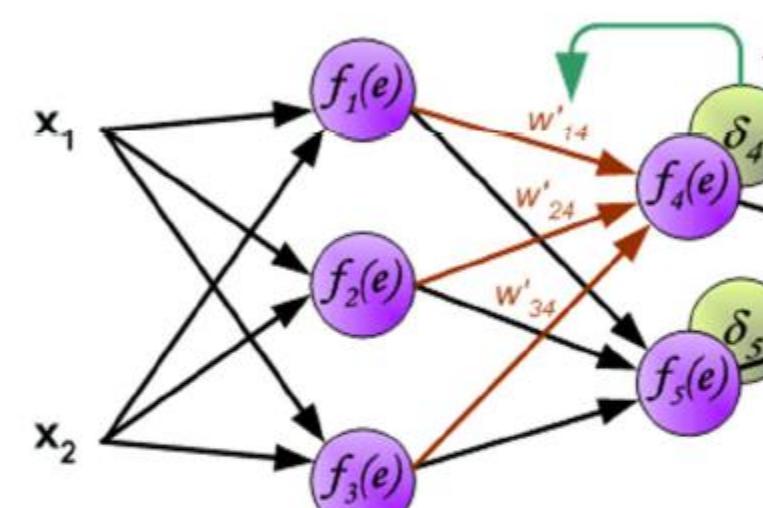
$$w'_{24} = w_{24} + \eta \delta_4 \frac{df_4(e)}{de} y_2$$

$$w'_{34} = w_{34} + \eta \delta_4 \frac{df_4(e)}{de} y_3$$

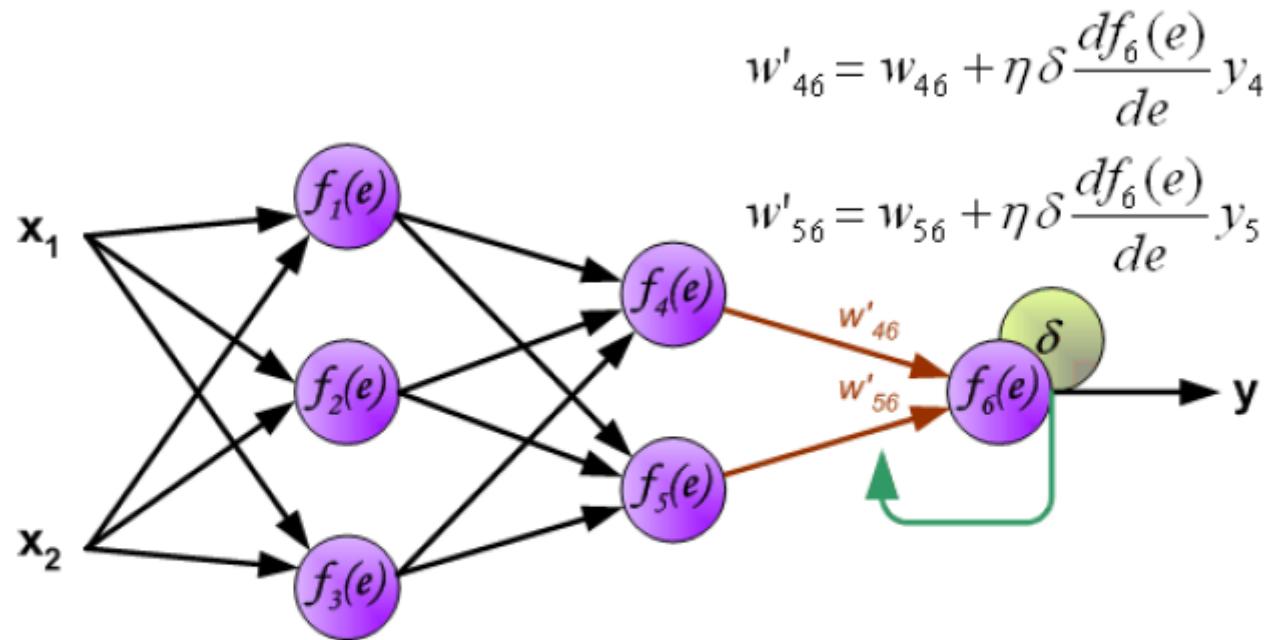
$$w'_{15} = w_{15} + \eta \delta_5 \frac{df_5(e)}{de} y_1$$

$$w'_{25} = w_{25} + \eta \delta_5 \frac{df_5(e)}{de} y_2$$

$$w'_{35} = w_{35} + \eta \delta_5 \frac{df_5(e)}{de} y_3$$

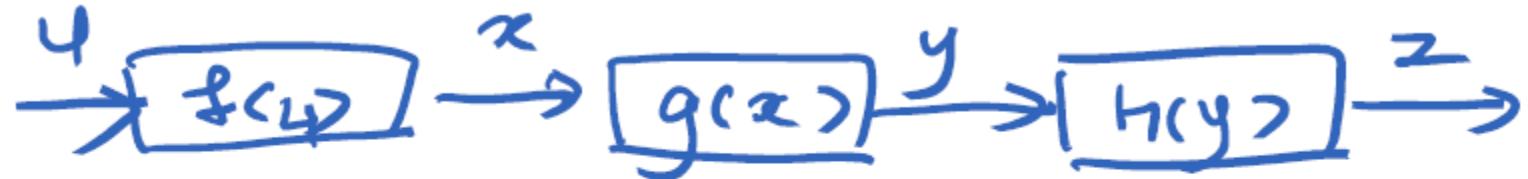


Finally ... Done



**Every weight in the MLP was updated,
Repeat the same for next instance.**

I Background Knowledge: Chain of Calculus



$$x = f(u)$$

$$y = g(x)$$

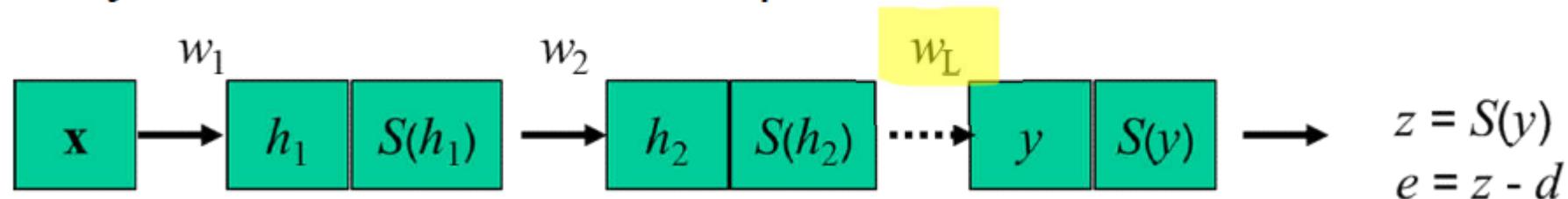
$$z = h(y)$$

$$\frac{\partial z}{\partial u} = \frac{\partial z}{\partial y} \times \frac{\partial y}{\partial x} \times \frac{\partial x}{\partial u}$$

$$= h'(y)g'(x)f'(u)$$

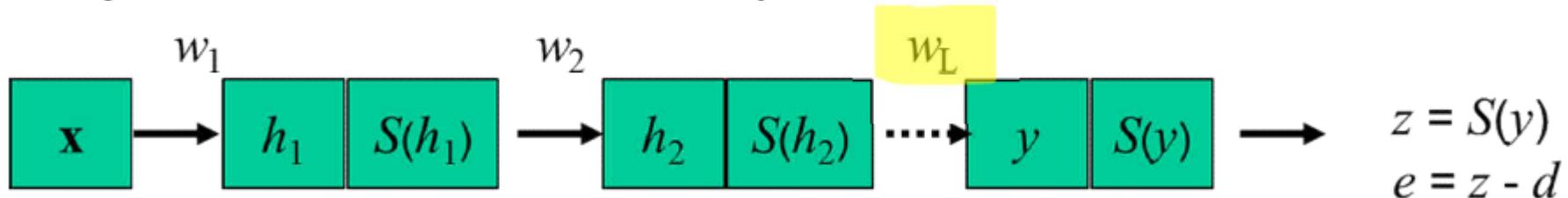
| Example 3

- Assume only one neuron for each layer and there are L layers in total. Assume square error loss $0.5e^2$.



Example 3

- Assume only one neuron for each layer and there are L layers in total. Assume square error loss $0.5e^2$.



$$f = \frac{1}{2}e^2 = \frac{1}{2}(z-d)^2$$

$$\frac{\partial f}{\partial w_L} = \frac{\partial f}{\partial z} \times \frac{\partial z}{\partial y} \times \frac{\partial y}{\partial w_L}$$

$$= (z-d) s'(y) s(h_{L-1})$$

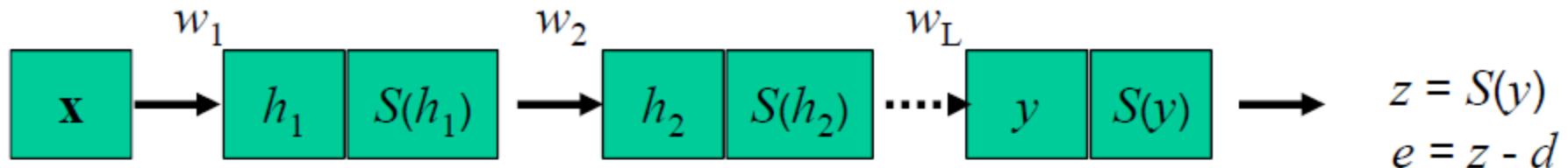
$$= e s'(y) s(h_{L-1})$$

$$\Delta w_L = -\eta e s'(y) s(h_{L-1})$$

3 terms

$$y = w_L s(h_{L-1})$$
$$z = s(y)$$

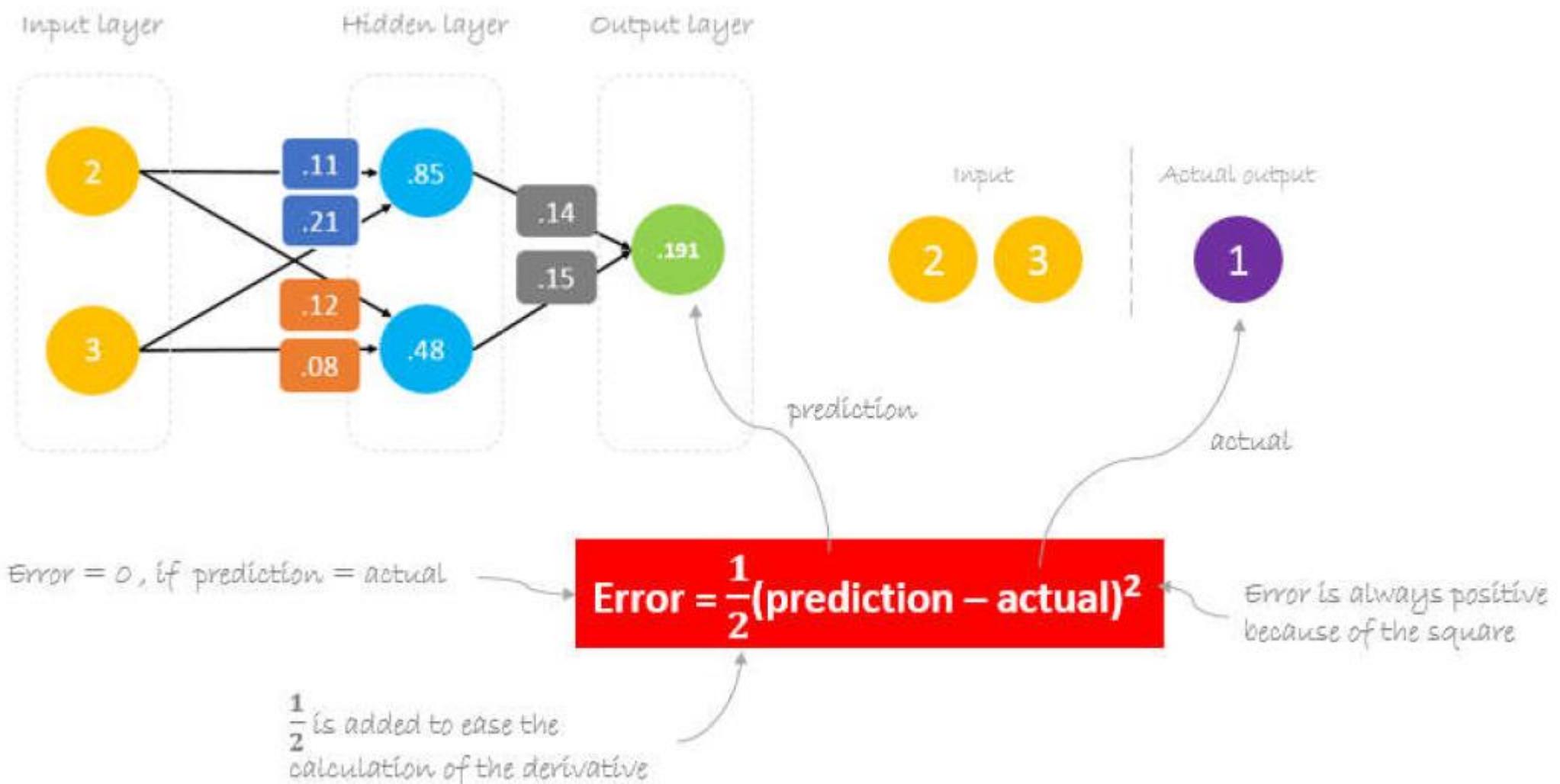
| Example 3



$$\Delta w_1 = -\eta e S'(y) \prod_{i=1}^{L-1} S'(h_i) \prod_{i=2}^L w_i \mathbf{x}$$

$$\Delta w_L = -\eta e S'(y) S(h_{L-1})$$

Example 4



다음과 같이 학습된 모델의 Error를 계산하여 BP를 거친 후에 보정된 가중치를 계산하여라.

Learning rate $a = 0.05$ 로 설정했다고 가정

Example 4 Solution

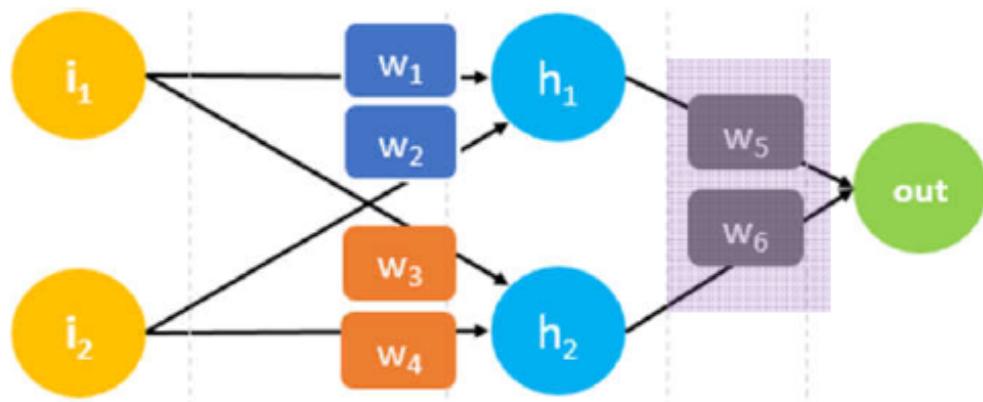
$$\text{Error} = \frac{1}{2}(0.191 - 1.0)^2 = 0.327$$

$$*W_x = W_x - a \left(\frac{\partial \text{Error}}{\partial W_x} \right)$$

Diagram illustrating the components of the weight update formula:

- Old weight**: The starting point for the update.
- New weight**: The result of the update.
- Derivative of Error with respect to weight**: The gradient term, calculated as $\frac{\partial \text{Error}}{\partial W_x}$.
- Learning rate**: The step size for the update, represented by the variable a .

Example 4 Solution



$$*W_6 = W_6 - a \left(\frac{\partial \text{Error}}{\partial W_6} \right)$$

$$*W_6 = W_6 - a \Delta h_2$$

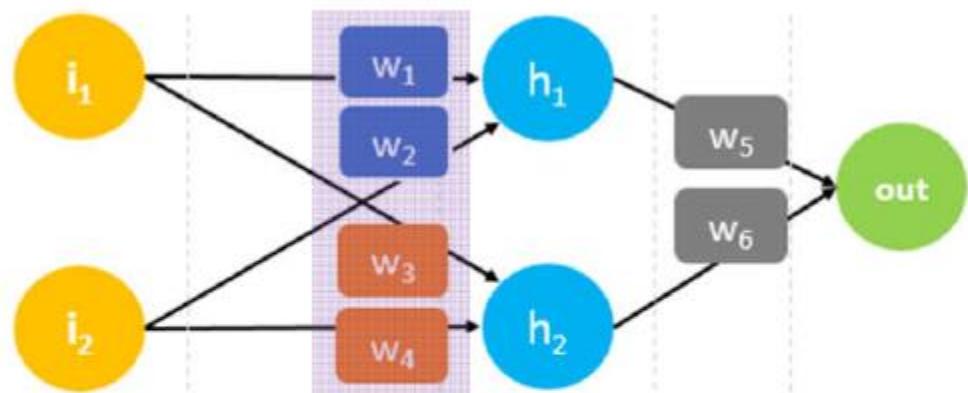
$$*W_5 = W_5 - a \Delta h_1$$

Derivation:

$$\begin{aligned} \frac{\partial \text{Error}}{\partial W_6} &= \frac{\partial \text{Error}}{\partial \text{prediction}} * \frac{\partial \text{prediction}}{\partial W_6} && \text{chain rule} \\ &= \frac{1}{2}(\text{predictoin} - \text{actula})^2 * \frac{\partial (\text{i}_1 w_1 + \text{i}_2 w_2) w_5 + (\text{i}_1 w_3 + \text{i}_2 w_4) w_6}{\partial \text{W}_6} \\ &= \frac{1}{2}(\text{predictoin} - \text{actula})^2 * \frac{\partial (\text{i}_1 w_1 + \text{i}_2 w_2) w_5}{\partial \text{W}_6} + \frac{\partial (\text{i}_1 w_3 + \text{i}_2 w_4) w_6}{\partial \text{W}_6} \\ &= 2 * \frac{1}{2}(\text{predictoin} - \text{actula}) \frac{\partial (\text{predictoin} - \text{actula})}{\partial \text{prediciton}} * (\text{i}_1 w_3 + \text{i}_2 w_4) && \text{h}_2 = \text{i}_1 w_3 + \text{i}_2 w_4 \\ \frac{\partial \text{Error}}{\partial W_6} &= (\text{predictoin} - \text{actula}) * (\Delta h_2) && \Delta = \text{prediction} - \text{actual} \\ \frac{\partial \text{Error}}{\partial W_6} &= \Delta h_2 && \text{delta} \end{aligned}$$

Same derivation procedure for $\frac{\partial \text{Error}}{\partial W_5}$

Example 4 Solution



$$\frac{\partial \text{Error}}{\partial W_1} = \frac{\partial \text{Error}}{\partial \text{prediction}} * \frac{\partial \text{prediction}}{\partial h_1} * \frac{\partial h_1}{\partial W_1}$$

chain rule

$$\frac{\partial \text{Error}}{\partial W_1} = \frac{\frac{1}{2}(\text{predictoin} - \text{actula})^2}{\partial \text{prediciton}} * \frac{\partial (h_1) w_5 + (h_2) w_6}{\partial h_1} * \frac{\partial i_1 w_1 + i_2 w_2}{\partial w_1}$$

$$\frac{\partial \text{Error}}{\partial W_1} = 2 * \frac{1}{2}(\text{predictoin} - \text{actula}) \frac{\partial (\text{predictoin} - \text{actula})}{\partial \text{prediciton}} * (w_5) * (i_1)$$

$$\frac{\partial \text{Error}}{\partial W_1} = (\text{predictoin} - \text{actula}) * (w_5 i_1)$$

$\Delta = \text{prediction} - \text{actual}$

$\leftarrow \text{delta}$

$$\frac{\partial \text{Error}}{\partial W_1} = \Delta w_5 i_1$$

Same derivation procedure for $\frac{\partial \text{Error}}{\partial W_2}, \frac{\partial \text{Error}}{\partial W_3}, \frac{\partial \text{Error}}{\partial W_4}$

$$*W_6 = W_6 - a (h_2 \cdot \Delta)$$

$$*W_5 = W_5 - a (h_1 \cdot \Delta)$$

$$*W_4 = W_4 - a (i_2 \cdot \Delta w_6)$$

$$*W_3 = W_3 - a (i_1 \cdot \Delta w_6)$$

$$*W_2 = W_2 - a (i_2 \cdot \Delta w_5)$$

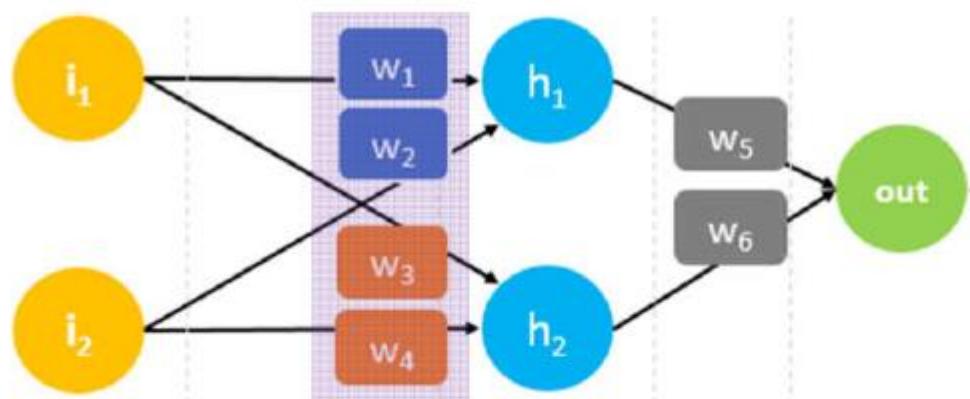
$$*W_1 = W_1 - a (i_1 \cdot \Delta w_5)$$

$$\text{Error} = \frac{1}{2}(\text{prediction} - \text{actual})^2$$

$$\text{prediction} = (h_1) w_5 + (h_2) w_6$$

$$h_1 = i_1 w_1 + i_2 w_2$$

Example 4 Solution



$$\begin{aligned} *w_6 &= w_6 - a(h_2 \cdot \Delta) \\ *w_5 &= w_5 - a(h_1 \cdot \Delta) \\ *w_4 &= w_4 - a(i_2 \cdot \Delta w_6) \\ *w_3 &= w_3 - a(i_1 \cdot \Delta w_6) \\ *w_2 &= w_2 - a(i_2 \cdot \Delta w_5) \\ *w_1 &= w_1 - a(i_1 \cdot \Delta w_5) \end{aligned}$$

Rewrite in Matrix Form

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - a \Delta \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - \begin{bmatrix} ah_1 \Delta \\ ah_2 \Delta \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - a \Delta \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} \cdot \begin{bmatrix} w_5 & w_6 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - \begin{bmatrix} a i_1 \Delta w_5 & a i_1 \Delta w_6 \\ a i_2 \Delta w_5 & a i_2 \Delta w_6 \end{bmatrix}$$

Example 4 Solution

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - a \Delta \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - \begin{bmatrix} ah_1 \Delta \\ ah_2 \Delta \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - a \Delta \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} \cdot [w_5 \quad w_6] = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - \begin{bmatrix} a i_1 \Delta w_5 & a i_1 \Delta w_6 \\ a i_2 \Delta w_5 & a i_2 \Delta w_6 \end{bmatrix}$$

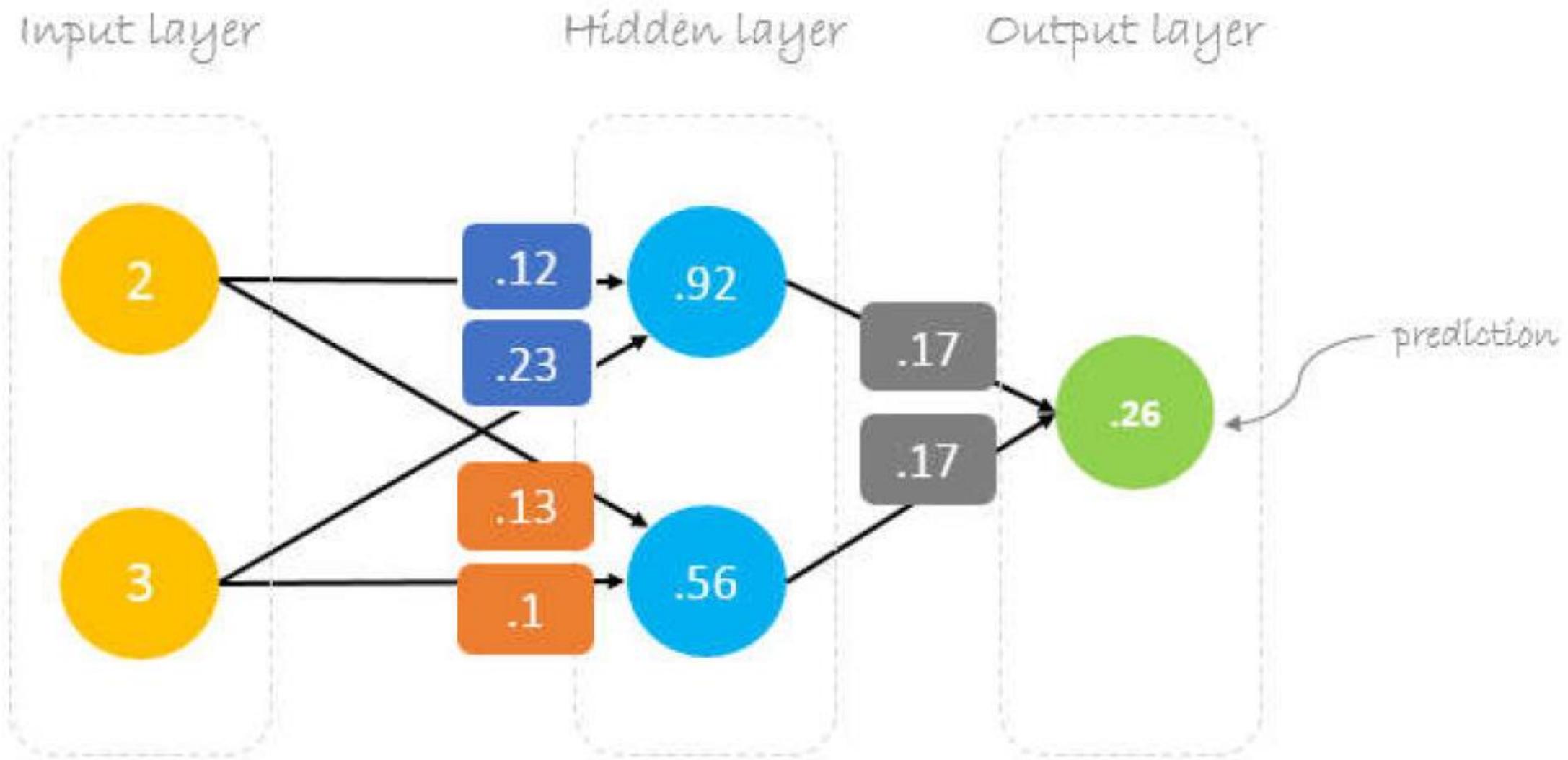
$$\Delta = 0.191 - 1 = -0.809 \quad \text{← Delta = prediction - actual}$$

$$a = 0.05 \quad \text{← Learning rate, we smartly guess this number}$$

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - 0.05(-0.809) \begin{bmatrix} 0.85 \\ 0.48 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - \begin{bmatrix} -0.034 \\ -0.019 \end{bmatrix} = \begin{bmatrix} 0.17 \\ 0.17 \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - 0.05(-0.809) \begin{bmatrix} 2 \\ 3 \end{bmatrix} \cdot [0.14 \quad 0.15] = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - \begin{bmatrix} -0.011 & -0.012 \\ -0.017 & -0.018 \end{bmatrix} = \begin{bmatrix} .12 & .13 \\ .23 & .10 \end{bmatrix}$$

Example 4 Solution



I Neuron network data

데이터

$$D_N = \{(\mathbf{x}^{(d)}, y^{(d)})\}_{d=1}^N \sim P$$



I Split of data

$$D_N = D_{train} \cup D_{test} \cup D_{valid}$$

- 분할된 데이터의 역할
 - 학습용 **training**
 - 모델(의 매개변수)를 조정하는 데 직접 사용
 - 테스트 test
 - 모델의 최종 성능을 측정하는 데 사용
 - 학습에 쓰이지 않음
 - 검증용 validation data
 - 모델의 성능을 중간에 점검할 때 사용
 - 모델을 직접적으로 조정하는 데 쓰이지 않으나 간접적으로 학습에 기여할 수 있음
- 데이터를 분할할 때에는 각 데이터가 갖는 분포가 유사해야 함

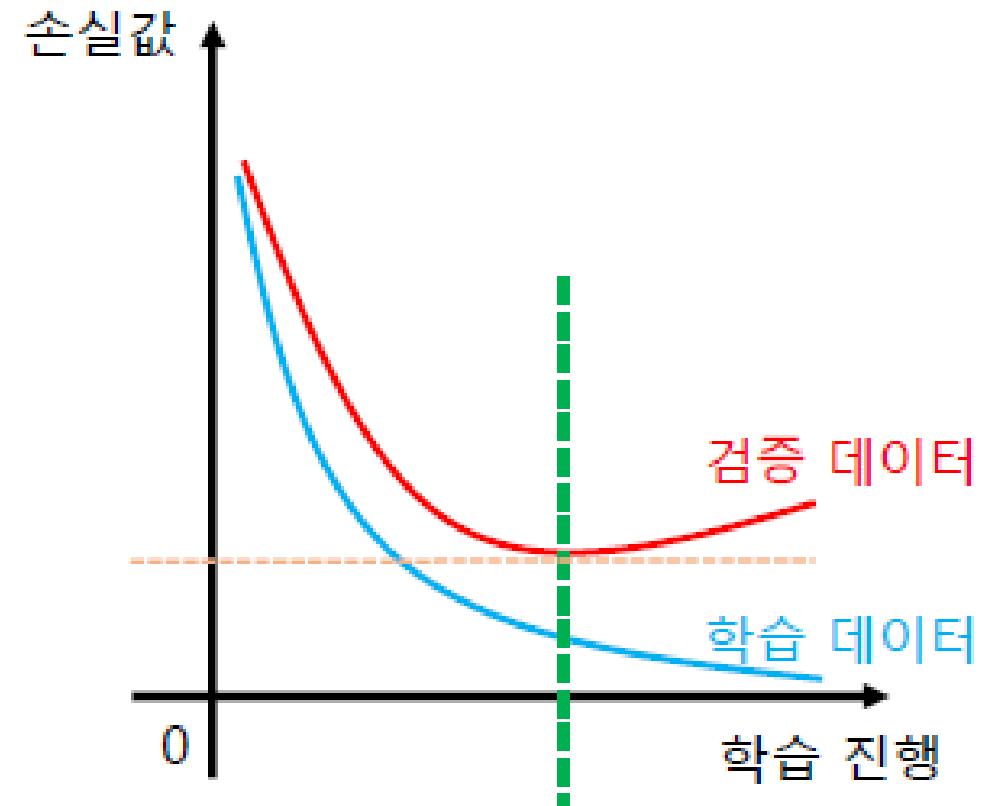
$$D_{train} = \{(\mathbf{x}^{(d)}, y^{(d)})\}_{d=1}^A$$

$$D_{test} = \{(\mathbf{x}^{(d)}, y^{(d)})\}_{d=1}^B$$

$$D_{valid} = \{(\mathbf{x}^{(d)}, y^{(d)})\}_{d=1}^C$$

Application of validation data

- 데이터 학습을 반복할수록 학습 데이터에 대한 손실값은 줄어듬
- 그러나 학습에 쓰이지 않은 검증 데이터의 경우, 손실값이 감소를 멈추고 다시 증가하는 시점이 있음
- 과적합 overfitting:**
모델이 주어진 학습 데이터에 대한 손실값을 최소화하는 과정에서 학습이 과도하게 진행되고, 이로 인해 학습하지 않은 데이터에 대한 성능은 오히려 떨어지는 현상
- early stopping:**
과적합을 예방하기 위해 검증 데이터에 대한 손실값이 증가하기 시작할 때 학습을 멈추는 기법

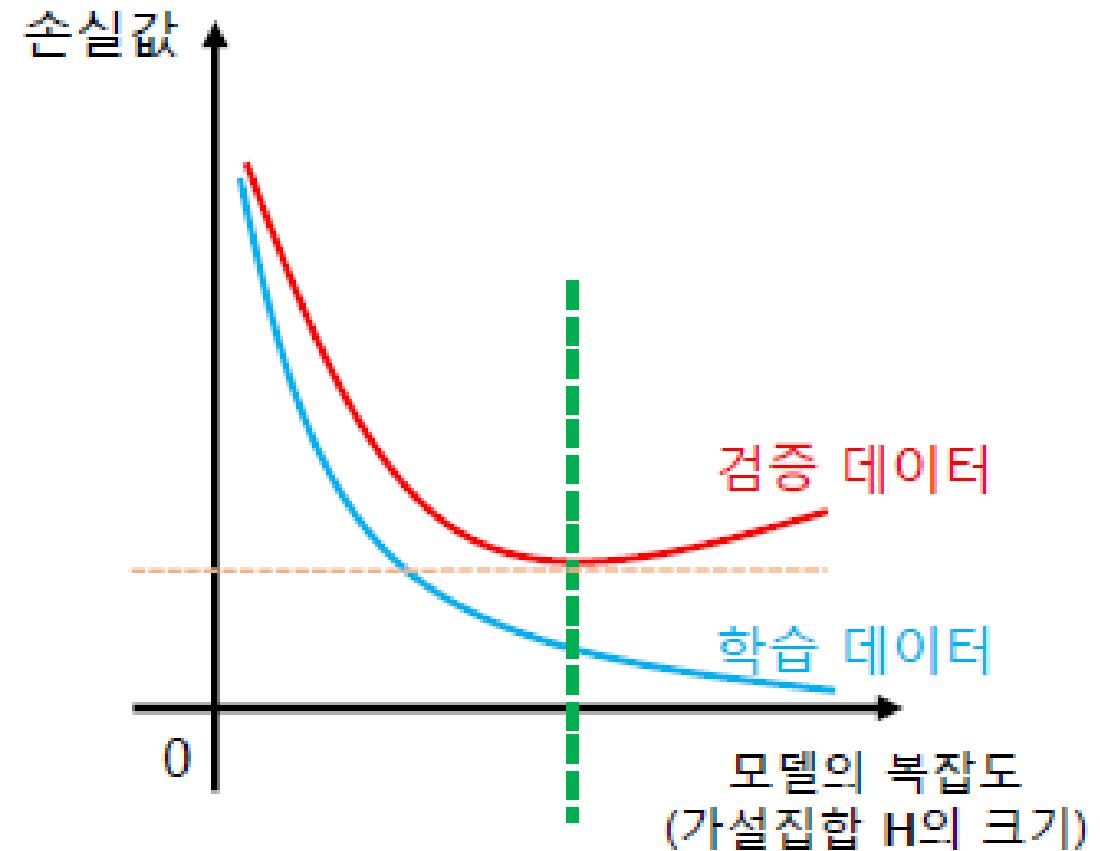


■ MLP Architectures

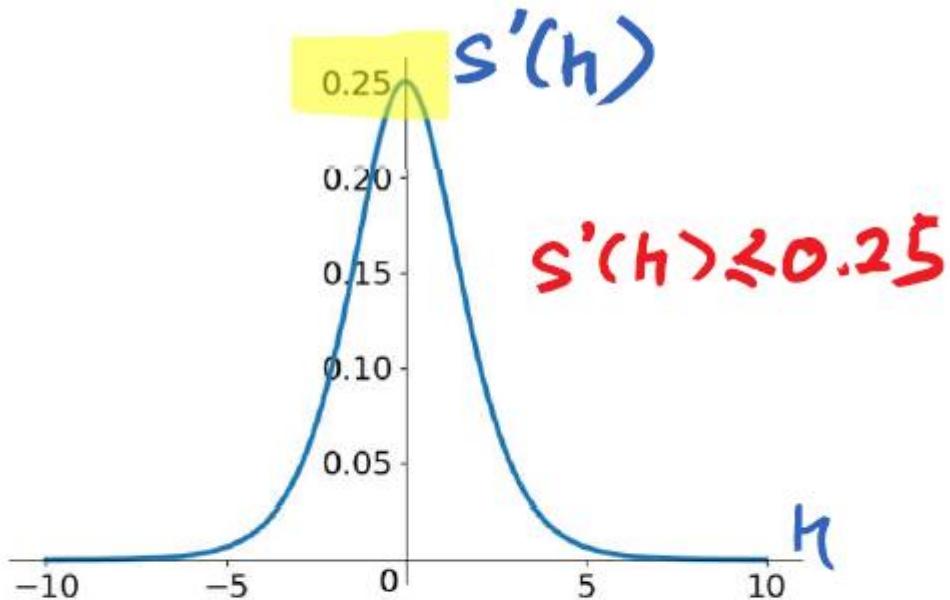
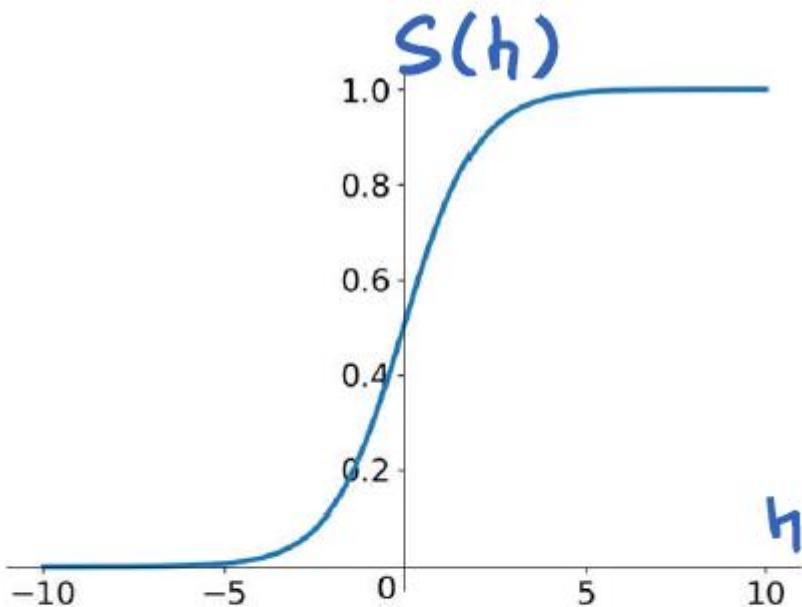
- 가중치의 개수
 - 각 가중치 행렬당 (전 층의 뉴런 수) \times (다음 층의 뉴런 수) 만큼 존재
 - 뉴런 개수가 785 – 100 – 10인 신경망의 경우
 - 첫 번째 행렬: 78500개
 - 두 번째 행렬: 1000개
- 활성 함수
 - 시그모이드
 - Rectified Linear Unit (ReLU)
 - Tanh (hyper tangent) 등
- 출력을 처리하는 함수
 - Softmax 등

I Overfitting and Complexity in model

- 모델이 복잡할수록(가설집합이 클수록) 더 복잡한 데이터를 학습할 수 있게 됨
- 주어진 학습 데이터에 비해 모델의 복잡도가 과도하게 큰 경우, 학습이 진행되면서 과적합이 일어나 테스트 성능이 오히려 떨어지는 경우가 발생할 수 있음
- 반대로 모델의 복잡도가 지나치게 낮으면 과소적합underfitting이 일어나 학습을 아무리 진행해도 성능을 충분히 높이기 어려운 경우가 생길 수 있음
- 따라서 학습하는 데이터에 따라 적절한 수준의 복잡도를 가진 모델을 정하는 것이 효율적인 학습을 하는 데 도움이 됨



Why sigmoid activation function is a bad choice?



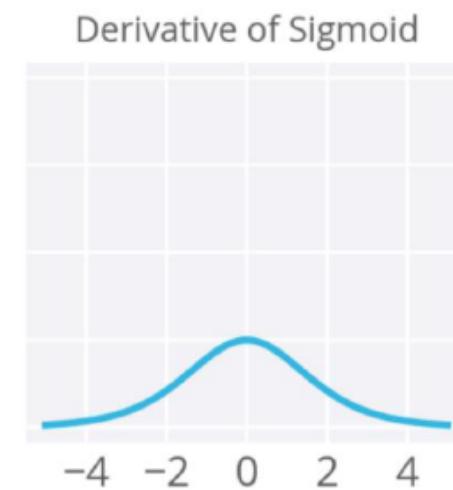
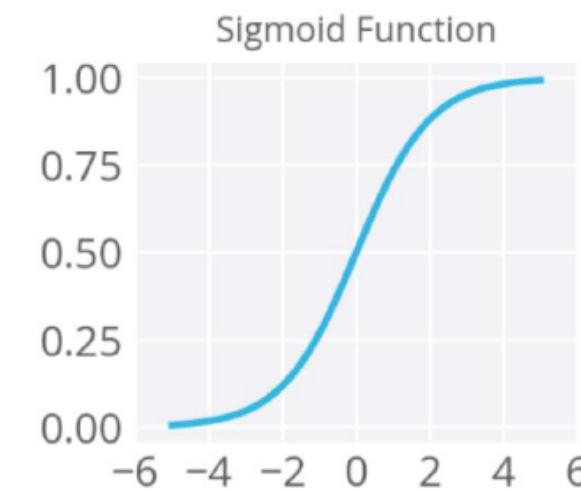
$$\Delta w_1 = -\eta e S'(y) \prod_{i=1}^{L-1} S'(h_i) \prod_{i=2}^L w_i \mathbf{x}$$

$$\Delta w_L = -\eta e S'(y) S(h_{L-1})$$

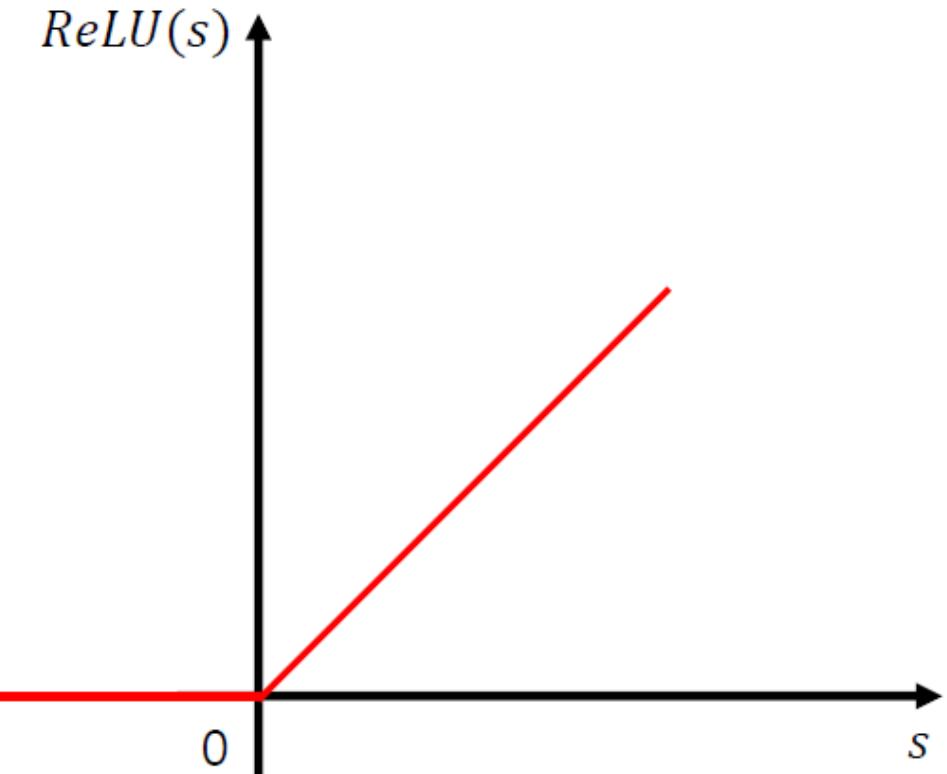
Activation function

- 경사도 소멸 vanishing gradient 현상
 - 깊은 신경망에서 시그모이드 활성 함수를 쓰는 경우 발생
 - $\frac{\partial f_j^{(l)}}{\partial s_j^{(l)}} = \frac{\partial \sigma(s_j^{(l)})}{\partial s_j^{(l)}} = \sigma(s_j^{(l)}) (1 - \sigma(s_j^{(l)}))$, $0 < \sigma(s_j^{(l)}) < 1$, $0 < \frac{\partial f_j^{(l)}}{\partial s_j^{(l)}} \leq \frac{1}{4}$
 - 출력층에서 먼 층 일수록 뉴런의 가중치가 변화되는 양이 지수적으로 감소하여 학습에 악영향을 줌

$$\delta_j^{(l)} = \begin{cases} f_j^{(l)} (1 - f_j^{(l)}) (y_j - f_j^{(l)}) & \text{for } j \in \text{Outputs} \\ f_j^{(l)} (1 - f_j^{(l)}) \sum_{k \in \text{Upper}(j)} \delta_k^{(l+1)} w_{kj}^{(l+1)} & \text{for } j \in \text{Hiddens} \end{cases}$$



■ Activation function

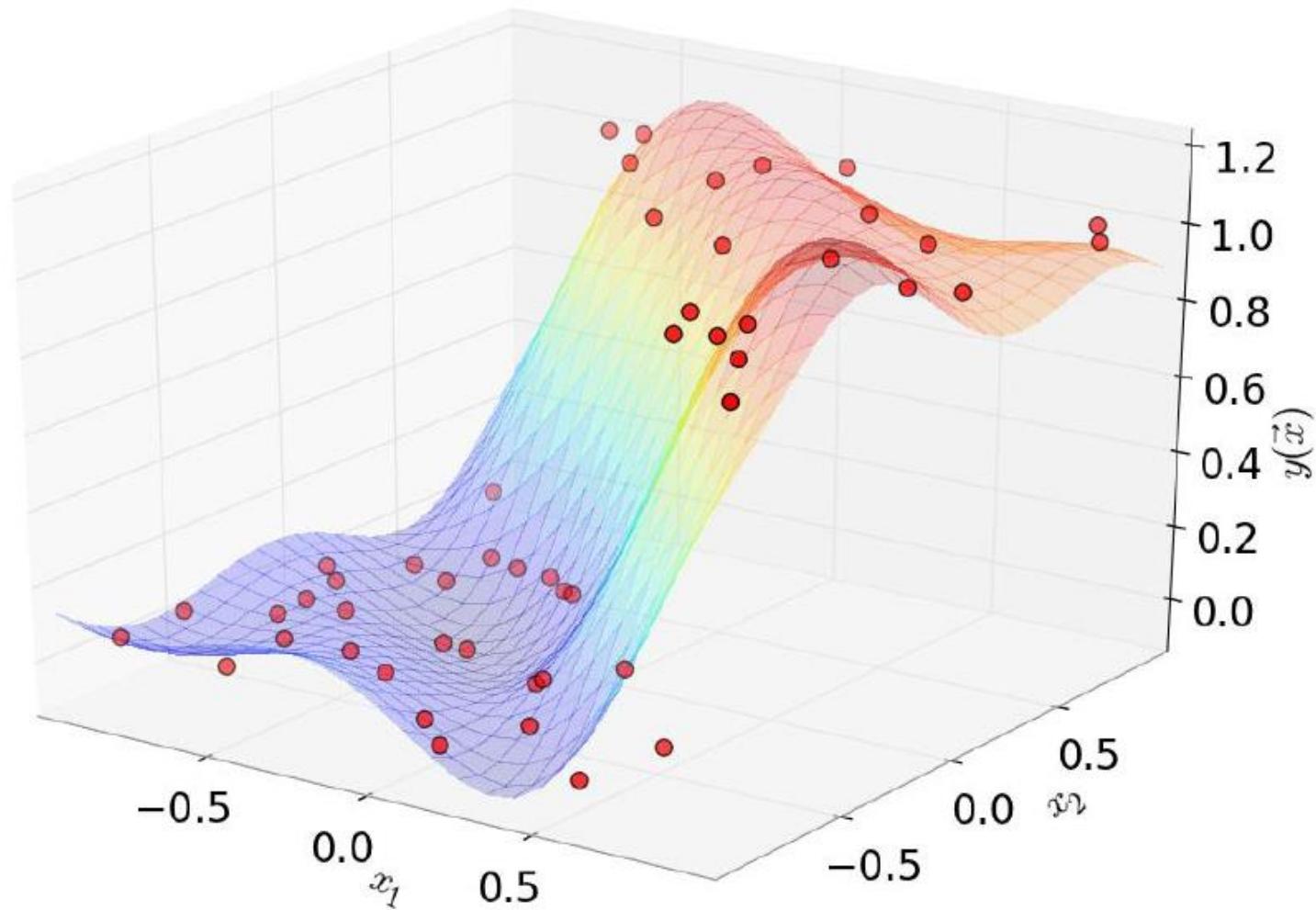


- 대안: Rectified Linear Unit(ReLU)

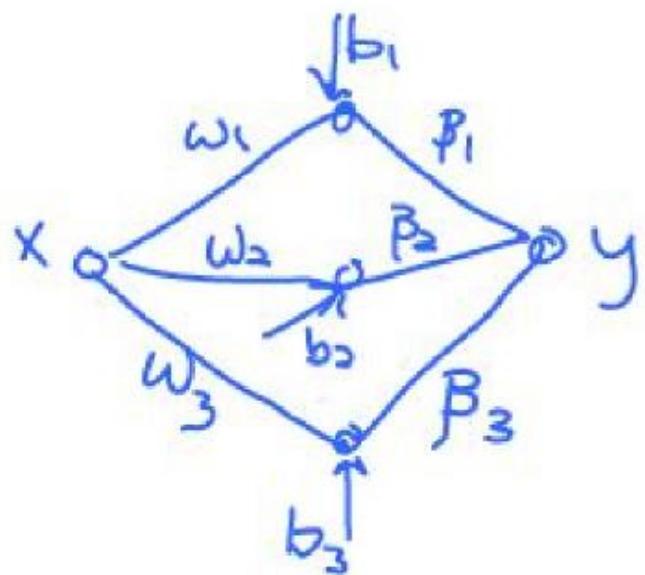
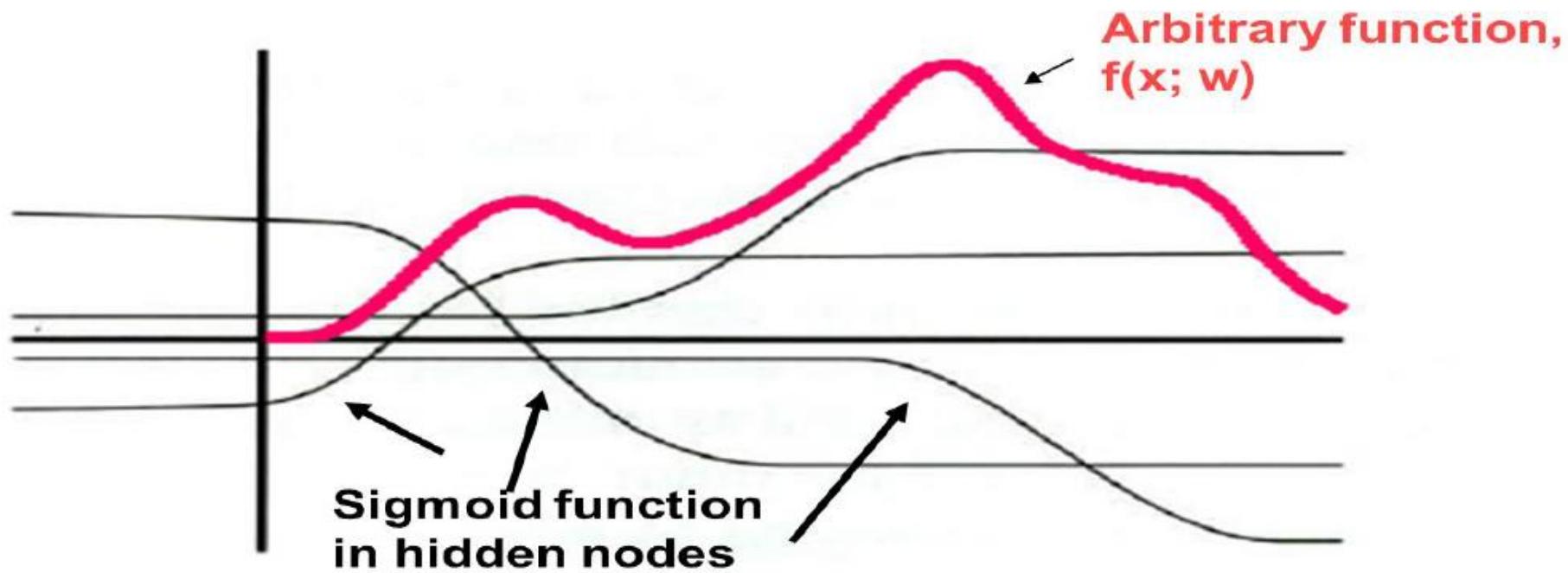
- $ReLU(s) = \begin{cases} s & \text{if } s > 0 \\ 0 & \text{if } s \leq 0 \end{cases}$ $\frac{\partial ReLU(s)}{\partial s} = \begin{cases} 1 & \text{if } s > 0 \\ 0 & \text{if } s \leq 0 \end{cases}$
- (0이 아닌 지점에서) 미분 가능, 비선형 연속함수
- 출력층에서 먼 층이더라도 뉴런 가중치의 변화량이 지수적으로 감소하지 않으므로 층수가 많은(깊은) 신경망에서도 원활한 학습이 가능함

MLP for Regression

- As a regression model, output neuron adopts linear function.



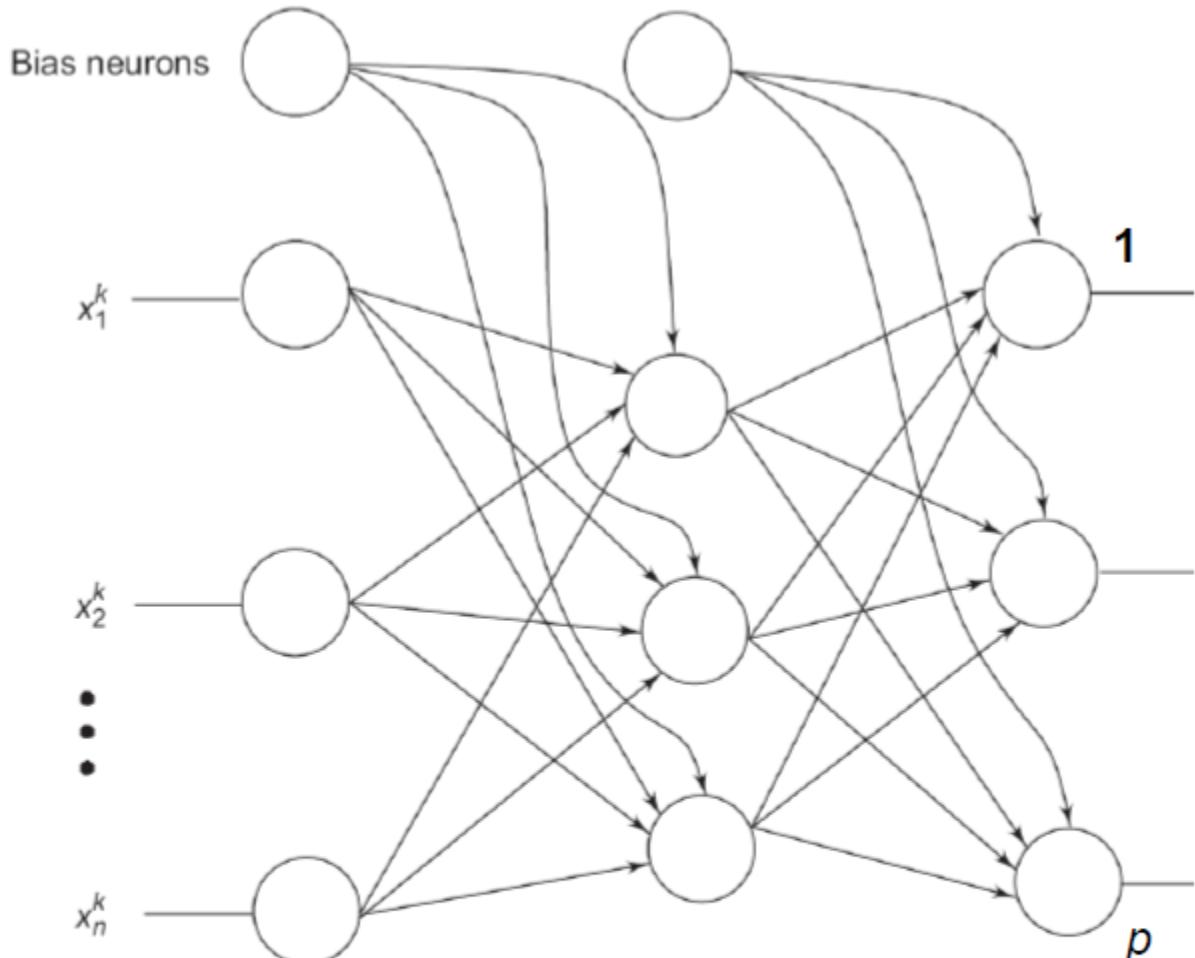
MLP for Regression



$$y = \beta_1 \text{Sigmoid}(w_1x + b_1) + \beta_2 \text{Sigmoid}(w_2x + b_2) + \beta_3 \text{Sigmoid}(w_3x + b_3)$$

Loss Function for Regression

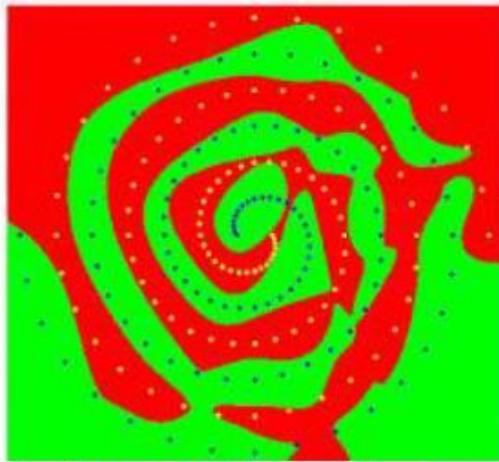
- Linear function is used at the output layer of MLP.
- The instantaneous summed squared error ε_k is the sum of the squares of each individual output error e_j^k , scaled by one-half:



$$E_k = (e_1^k, \dots, e_p^k)^T = (d_1^k - \mathcal{S}(y_1^k), \dots, d_p^k - \mathcal{S}(y_p^k))^T$$

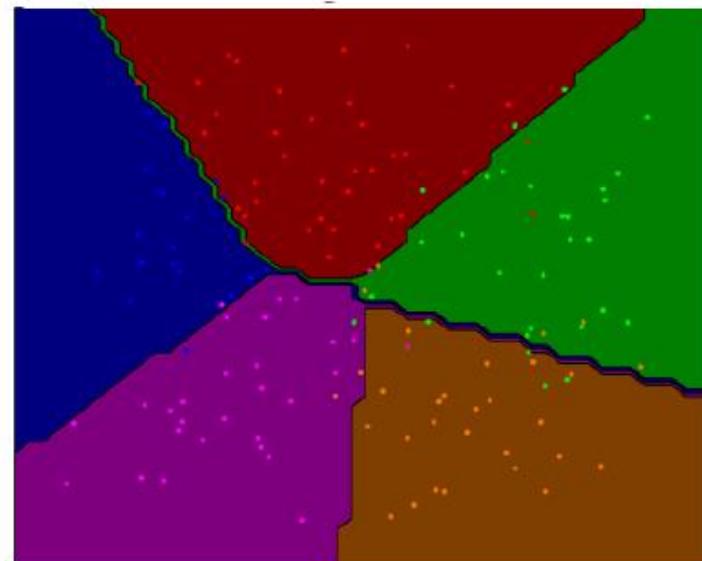
Application of MLP(Classification)

- As a (non-linear) classifier, output neuron adopts
 - Sigmoidal function – Binary classification



- Softmax function – Multi-class classification

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}$$



■ Application of MLP(Classification)

- 목표 출력값의 종류가 2가지인 분류 문제인 경우 신경망의 출력값을 하나로 한 뒤 시그모이드 함수를 이용하여 풀 수 있음
- 목표 출력값의 종류가 3가지 이상인 경우
 - One-hot vector
 - $y^{(d)} = (y_1^{(d)}, y_2^{(d)}, \dots, y_K^{(d)})$, $y_k^{(d)} = \begin{cases} 1 & \text{if answer} \\ 0 & \text{if not} \end{cases}$
 - $out = (out_1, out_2, \dots, out_K)$
 - $Softmax(out)_j = \frac{e^{out_j}}{\sum_{k=1}^K e^{out_k}}$
 - $\sum_{j=1}^K Softmax(out)_j = 1$, $0 < Softmax(out)_j < 1$
 - $Softmax(out)_j$: j번째 출력이 정답일 확률(의 예측값)

■ Application of MLP(Classification)

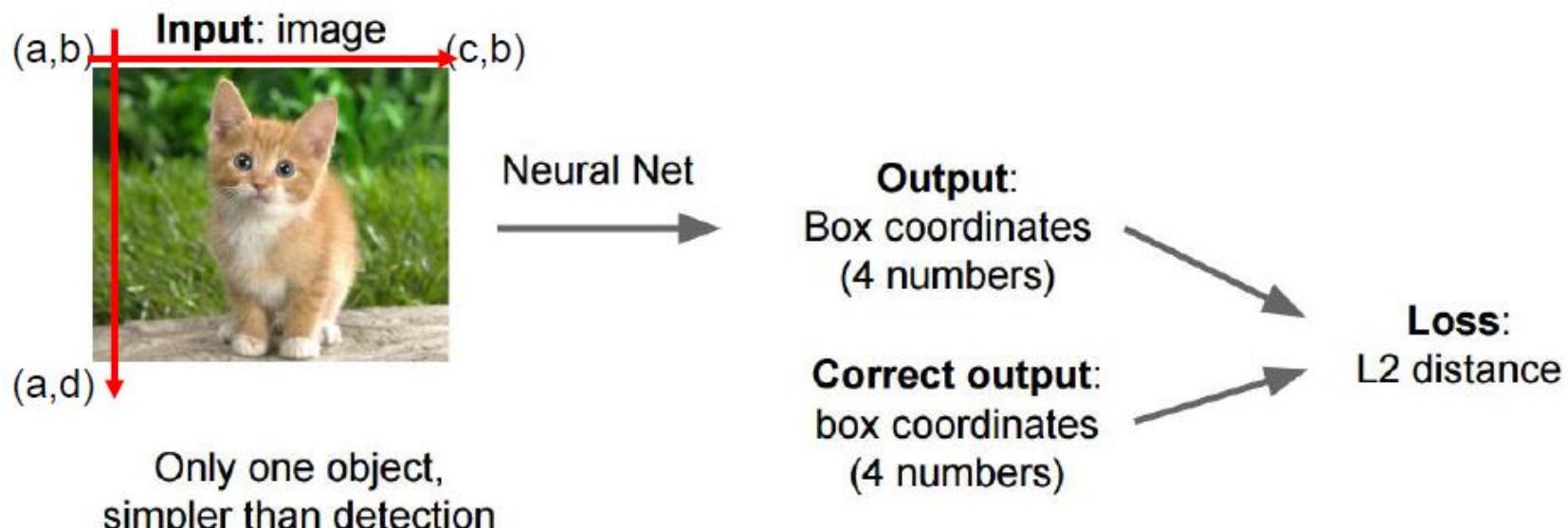
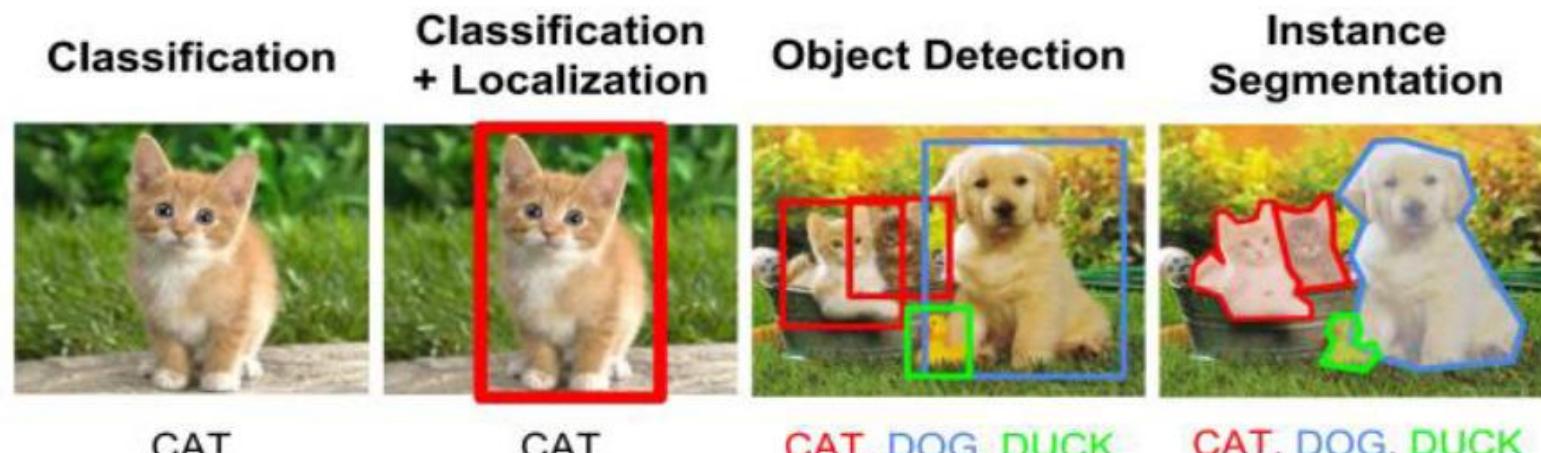
- 손실 함수: cross-entropy

- $CE(out^{(d)}, y^{(d)}) = -\sum_{k=1}^K y_k^{(d)} \log(out_k^{(d)})$
- $y^{(d)}$ 가 one-hot vector이고 $y_i^{(d)}$ 가 정답일 때

$$CE(out^{(d)}, y^{(d)}) = -\sum_{k=1}^K y_k^{(d)} \log(out_k^{(d)}) = -\log(out_i^{(d)})$$

$$\frac{\partial CE(out^{(d)}, y^{(d)})}{\partial out_k^{(d)}} = -\frac{\partial \log(out_i^{(d)})}{\partial out_k^{(d)}} = \begin{cases} -\frac{1}{out_i^{(d)}} & \text{if } k = i \\ 0 & \text{otherwise} \end{cases}$$

Usage of Classification and Regression

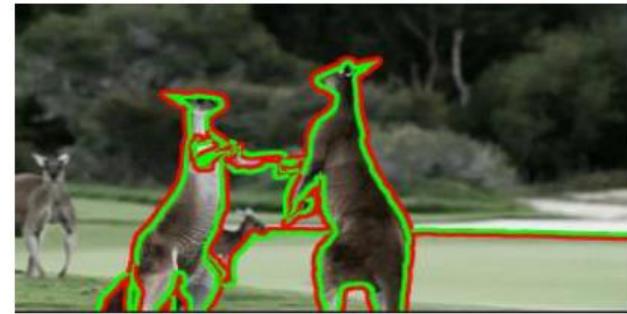


Localization as a Function Approximation (Regression) Problem

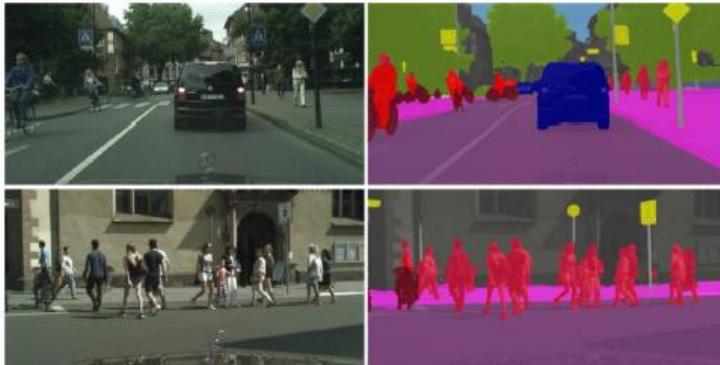
Usage of Classification and Regression



Object Detection

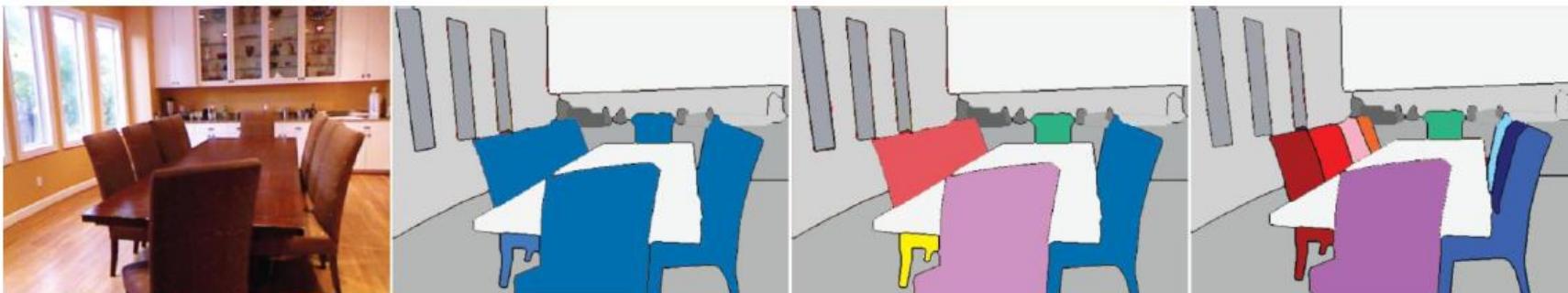


Boundary Detection



Semantic segmentation

Named-entity-recognition (Sequence labeling)



Input Image

Semantic Segmentation

Boundary Segmentation

Semantic Instance Segmentation

More examples for classification in computer vision

In 1917, Einstein applied the general theory of relativity to model the large-scale structure of the universe. He was visiting the United States when Adolf Hitler came to power in 1933 and did not go back to Germany, where he had been a professor at the Berlin Academy of Sciences. He settled in the U.S., becoming an American citizen in 1940. On the eve of World War II, he endorsed a letter to President Franklin D. Roosevelt alerting him to the potential development of "extremely powerful bombs of a new type" and recommending that the U.S. begin similar research. This eventually led to what would become the Manhattan Project. Einstein supported defending the Allied forces, but largely denounced using the new discovery of nuclear fission as a weapon. Later, with the British philosopher Bertrand Russell, Einstein signed the Russell-Einstein Manifesto, which highlighted the danger of nuclear weapons. Einstein was affiliated with the Institute for Advanced Study in Princeton, New Jersey, until his death in 1955.

Tag colours:

LOCATION TIME PERSON ORGANIZATION MONEY PERCENT DATE



THANK YOU