인공신경망 & 딥러닝

머신러닝 기반 빅데이터 엔지니어링 과정
빅데이터 X Campus (단국대학교)
2018.08
컴퓨터공학과 최상일 교수

목차

01

인공신경망

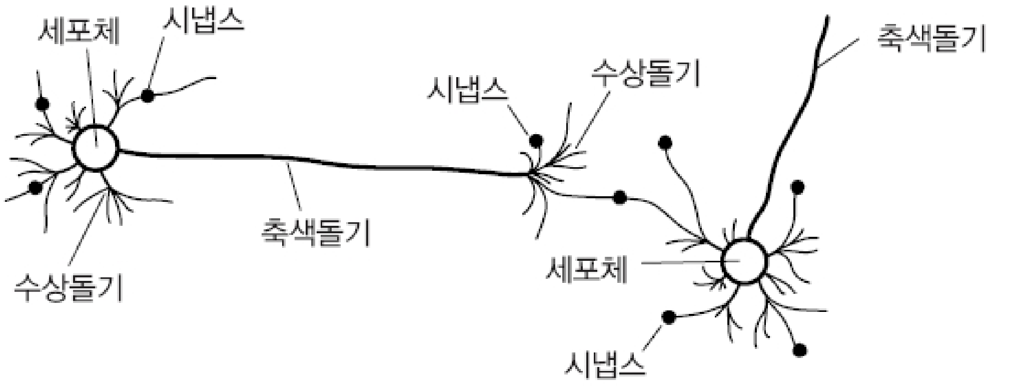
기계 학습

- 컴퓨터가 경험, 예, 유추를 통해 학습할 수 있게 하는 적응 메커니즘
- 학습 능력은 시간이 흐르면서 지능형 시스템의 성능을 개선함
- 적응형 시스템의 기초를 형성
- 기계 학습의 접근법의 예
  - 통계적 의사결정론(statistical decision theory)
  - 인공 신경망(artificial neural network)
  - 유전 알고리즘(genetic algorithm)

- 기계 학습의 예 - 체스 게임
  - 팁 블루와 체스 게임 선수의 체스 게임
  - 팁 블루 – IBM에서 만든 초당 2억 개의 포지션을 분석할 수 있는 컴퓨터
    - 체스 프로그램은 경험을 통해 성능이 향상됨
    - 기계는 반드시 학습 능력이 있어야 함

인공 신경망

- 인간 뇌를 기반으로 한 추론 모델



[그림 6-1] 생물학적인 신경망

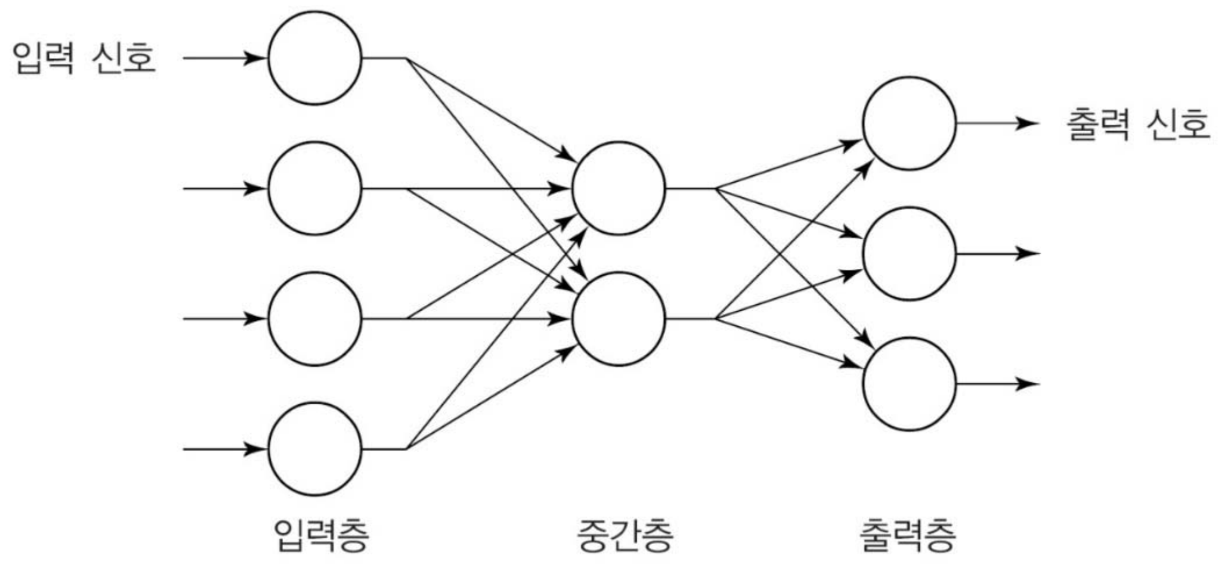
- 뉴런 : 기본적인 정보처리 단위
- 인공 신경망의 주요 특징 : 적응성

뇌의 동작 원리

- 인간 뇌의 특징
  - 100억개의 뉴런과 각 뉴런을 연결하는 6조 개의 시냅스의 결합체
  - 시냅스를 통해 신호를 주고 받음으로써 정보를 저장하고 학습
  - 현존하는 어떤 컴퓨터보다 빠르게 기능을 수행
  - 매우 복잡하고, 비선형적이며, 병렬적인 정보 처리 시스템
  - 정보가 신경망 전체에 동시에 저장되고 처리
  - 경험을 통한 학습 능력
    - ‘잘못된 답’으로 이끄는 뉴런들 사이의 연결은 약화
    - ‘올바른 답’으로 이끄는 연결은 강화

- 인공 신경망의 특징
  - 인간 뇌를 기반으로 모델링
  - 인간 뇌의 적응성을 활용하여 ‘학습 능력’을 구현
  - 그러나 아직 인간의 뇌를 흉내내기에 많이 미흡

- 인간의 뇌 모델링
  - 생물학적인 뇌의 뉴런과 비슷하게 모델링
  - ‘뉴런’이라는 아주 단순하지만 내부적으로 매우 복합하게 연결된 프로세스들로 구성
  - 가중치 있는 링크들로 뉴런들을 연결
  - 각각의 뉴런은 연결을 통해 여러 입력 신호를 받지만 오직 하나의 신호만 출력



[그림 6-2] 전형적인 인공 신경망의 구조

인공 신경망 모델링

- 생물학적인 신경망과 인공 신경망의 유사점

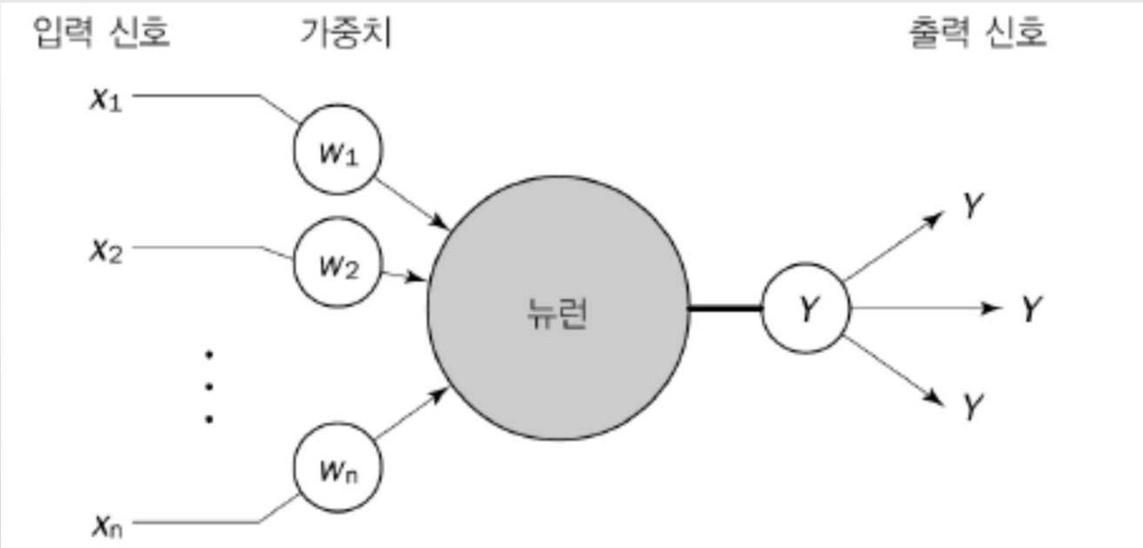
[표 6-1] 생물학적인 신경망과 인공 신경망 사이의 유사점

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

- 인공 신경망의 학습
  - 뉴런 사이를 연결하는 신경망의 **가중치**를 반복적으로 조정하여 학습
    - ➢ 뉴런 사이의 링크(link)
    - ➢ 가중치 : 장기 기억을 위한 기본적인 수단; 각 뉴런의 입력 강도, 즉 중요도를 표현
- 인공 신경망의 가중치 조정
  - 신경망의 구조를 먼저 선택 → 사용할 학습 알고리즘 선택 → 신경망 학습
  - 신경망 학습
    - ➢ 신경망의 가중치를 초기화 → 학습 데이터 샘플들을 이용하여 해당 가중치를 갱신

단순한 계산 요소로서의 뉴런

- 뉴런의 특징
  - 복수의 링크로부터 받은 입력 신호를 합하여 활성화 수준을 계산하여 출력 링크로 출력
    - ➢ 입력 신호 : 미가공 데이터 또는 다른 뉴런의 출력
    - ➢ 출력 신호 : 문제의 최종적인 해(solution) 또는 다른 뉴런의 입력
  - 뉴런의 예



단순한 계산 요소로서의 뉴런

뉴런의 계산

- 뉴런의 출력 결정
  - 선형 결합과 활성화 함수
  - 전이 함수, 즉 활성화 함수(activation function)를 사용
  - 활성화 함수를 이용한 출력 결정 순서

    1. 뉴런은 입력 신호의 가중치 합을 계산하여 임계값 θ와 비교

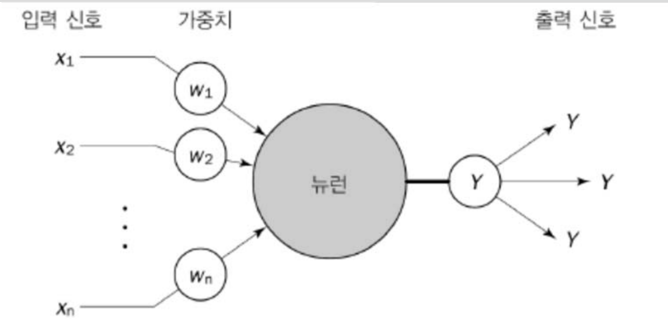
    2. 가중치 합이 임계값보다 작으면 '-1' 출력

    3. 가중치 합이 임계값과 같거나 크면 뉴런은 활성화되고, '+1' 출력

  - 뉴런의 입출력

$$X = \sum_{i=1}^{n} x_i w_i$$

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



X는 뉴런으로 들어가는 입력의 순 가중합

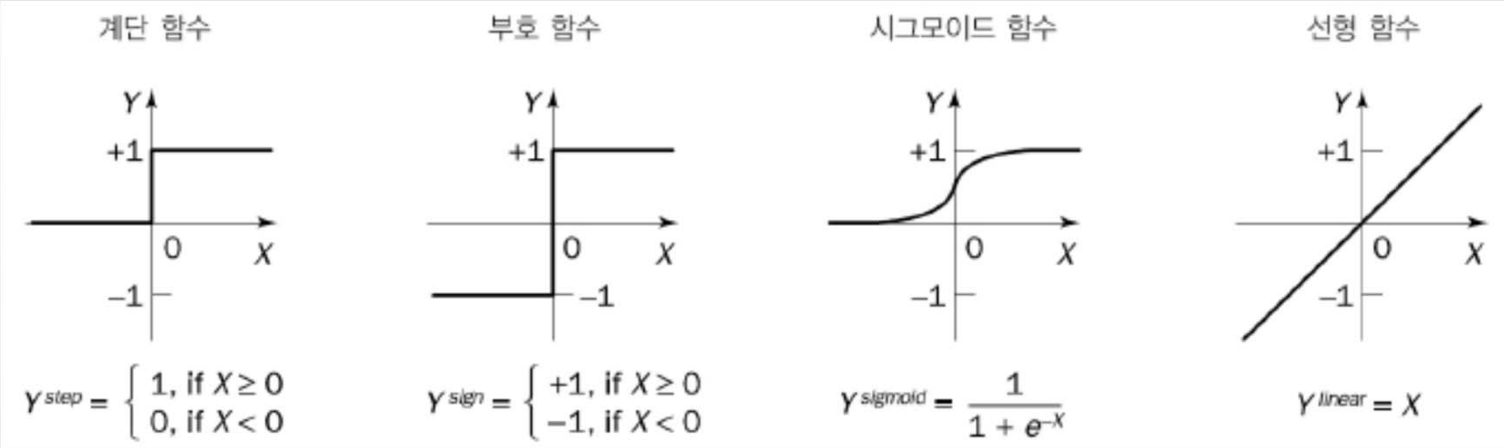
x_i는 입력 i의 값, w_i 는 입력i의 가중치, n은 뉴런의 입력 개수, Y는 뉴런의 출력

단순한 계산 요소로서의 뉴런

- 뉴런의 활성화 함수
  - 부호 활성화 함수를 사용하는 뉴런의 실제 출력

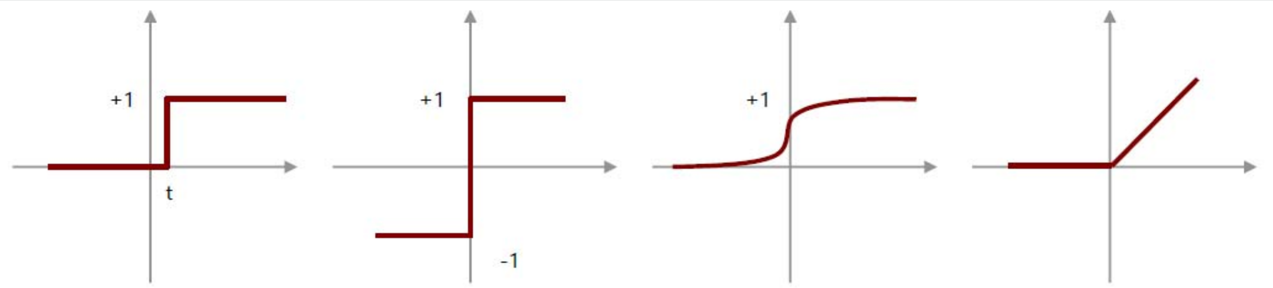
$$Y = sign\left[\sum_{i=1}^{n} x_i w_i - \theta\right]$$

  - 대표적인 활성화 함수들
    - ➢ 계단 함수, 부호 함수, 시그모이드 함수, 선형 함수



단순한 계산 요소로서의 뉴런

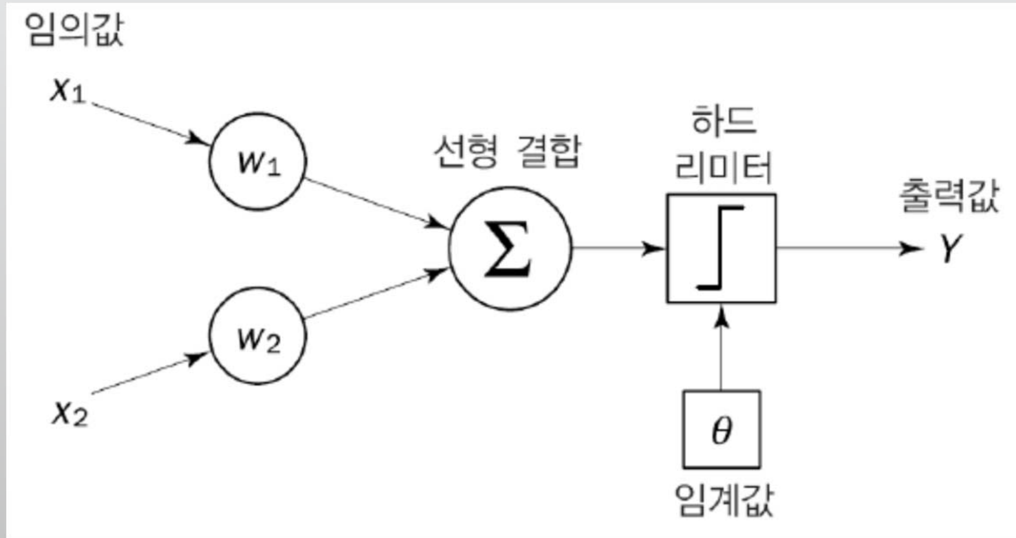
- 하드 리밋 함수(**hard limit function**)
  - 계단(**step**) 활성화 함수, 부호(**sign**) 활성화 함수
  - 분류와 패턴인식에서 주로 사용
- 시그모이드 함수(**sigmoid function**)
  - 양과 음의 무한대 사이에 있는 입력값을 **0~1** 사이에 있는 적당한 값으로 변환
  - 역전파 신경망에서 사용
- 선형 활성화 함수(**linear activation function**)
  - 뉴런의 입력에 가중치가 적용된 것과 같은 값을 출력
  - 선형 근사에 주로 사용
- **ReLu(Rectified Linear)** 함수 **: Vanishing Gradient** 문제 해결



Step	Sign	Sigmoid	Rectified Linear
$f(x) =$ $1 \text{ if } x \geq t(\text{threshold}), \text{else } 0$	$f(x) =$ $+1 \text{ if } x \geq 0, \text{else } -1$	$f(x) =$ $1/1 + e^{-x}$	$f(x) =$ $\max(0, x)$

퍼셉트론(perceptron)

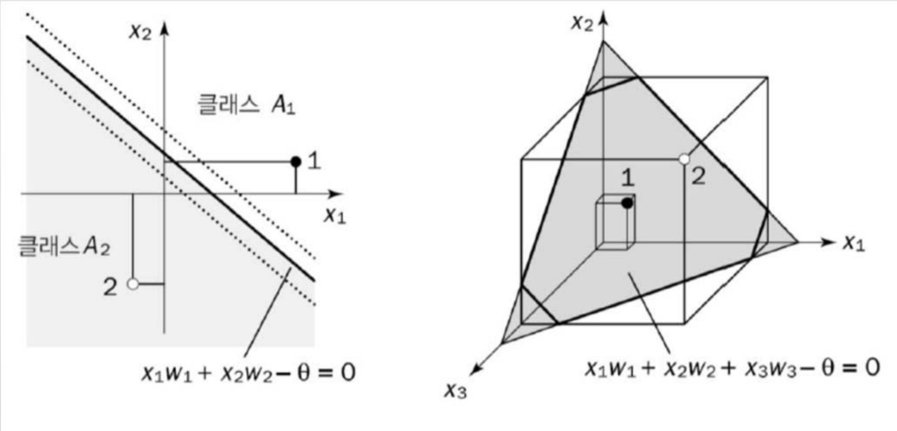
- 단일 뉴런의 학습
  - 퍼셉트론 (프랭크 로젠블랫 1958)
    - 간단한 인공 신경망 학습 알고리즘
    - 신경망의 가장 간단한 형태
    - 조정 가능한 시냅스 가중치(선형결합기)와 하드 리미터(hard limiter)를 포함한 단일 뉴런으로 구성
  - 입력 노드가 두 개인 단층 퍼셉트론



퍼셉트론

- 퍼셉트론의 동작 원리
  - 맥클록과 피츠의 뉴런 모델에 기반
  - 입력의 가중합을 하드 리미터에 입력 : 입력이 양이면 '**+1**', 음이면 '**-1**'을 출력
- 기본적인 퍼셉트론의 경우, 초평면(**hyperplane**)으로 **n**차원 공간을 두 개의 결정 영역 구분
  - 초평면을 선형 분리 함수로 정의

$$\sum_{i=1}^{n} x_i w_i - \theta = 0$$

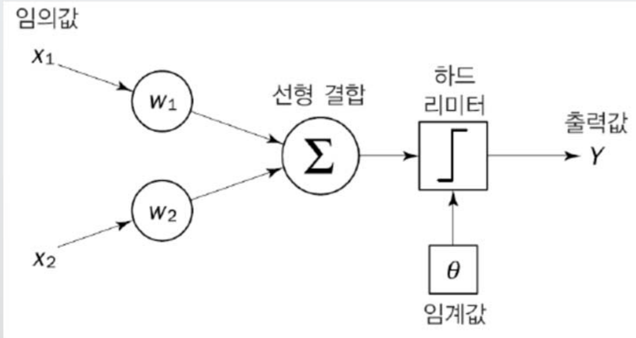


(a) 입력이 두 개인 퍼셉트론 (b) 입력이 세 개인 퍼셉트론

  - 경계선 오른편에 있는 점 '**1**' : 클래스 A_1 ; 경계선 왼편에 있는 점 '**2**' : 클래스 A_2
  - 임계값 θ : 결정 경계의 위치 조정

퍼셉트론 기본 학습

- 퍼셉트론 기본 학습 방법
  - 가중치를 조절하여 실제 출력과 목표 출력 간의 차이를 줄임
  - [−0.5, 0.5] 범위에서 초기 가중치를 임의로 할당
    - 정답이 있는 학습 샘플에 적용
    - 정답과 일치하는 출력을 얻도록 가중치를 갱신
  - 오차 계산 식 : $e(p) = Y_d(p) - Y(p)$
    - p번째 반복(퍼셉트론에 주어진 p번째 학습 샘플)
    - Y(p) : 실제 출력 ; Y_d(p) : 목표 출력



- 퍼셉트론 학습 규칙
  - <u>오차 e(p)가 양 → 퍼셉트론의 출력 Y(p)를 증가</u>
  - 오차 e(p)가 음 → Y(p)를 감소
  - <u>각 퍼셉트론 입력에 대해 $x_i(p) \times w_i(p)$ 값이 총합 X(p)에 기여</u>
    - $x_i(p)$ 가 양일 때, 가중치 $w_i(p)$ 가 커지면 Y(p)도 커짐
    - $X_i(p)$ 가 음일 때, 가중치 $w_i(p)$ 가 커지면 Y(p)가 줄어듬

$$Y = sign\left[\sum_{i=1}^{n} x_i w_i - \theta\right]$$

퍼셉트론

- 가중치 갱신

$$w_i(p+1) = w_i(p) + \alpha \times x_i(p) \times e(p)$$

  - $e(p) > 0$ 일때, $x_i(p) > 0$ ➔ $w_i(p+1)$ 증가
    $x_i(p) < 0$ ➔ $w_i(p+1)$ 감소
  - $e(p) < 0$ 일때, $x_i(p) > 0$ ➔ $w_i(p+1)$ 감소
    $x_i(p) < 0$ ➔ $w_i(p+1)$ 증가
  - α (1보다 작은 양의 상수) : 학습률(learning rate)

- 분류 작업을 위한 퍼셉트론 훈련 알고리즘 : 4단계 알고리즘
  - 1단계: 초기화
    - 초기 가중치 $w_1, w_2, \ldots, w_n$과 임계값 θ를 [-0.5, 0.5] 구간의 임의의 값으로 설정
  - 2단계: 활성화
    - 입력 $x_1(p), x_2(p), \ldots, x_n(p)$와 목표 출력 $Y_d(p)$를 적용하여 퍼셉트론을 활성화
    - 반복 횟수 p=1에서 실제 출력을 계산

$$Y(p) = step\left[\sum_{i=1}^{n} x_i(p) w_i(p) - \theta\right]$$

n : 퍼셉트론의 입력 개수;   step : 계단 활성화함수(activation function)

퍼셉트론

- 3단계: 가중치 학습
  - 퍼셉트론의 가중치를 갱신

$$w_i(p+1) = w_i(p) + \Delta w_i(p)$$
$$\Delta w_i(p) = \alpha \times x_i(p) \times e(p)$$

$\Delta w_i(p)$: p번째 반복했을 때의 가중치 보정값

- 4단계: 반복
  - 반복 횟수 p값을 1 증가
  - 2단계로 돌아가서 w_i가 수렴할 때까지 과정을 반복

퍼셉트론 학습 방법 예

- 기본적인 논리 연산자 학습
  - AND, OR, Exclusive-OR와 같은 기본적인 논리 연산자의 기능을 수행하도록 학습
  - 연산자 AND, OR, Exclusive-OR의 진리표

입력 변수		AND	OR	Exclusive-OR
x_1	x_2	$x_1 \cap x_2$	$x_1 \cup x_2$	$x_1 \oplus x_2$
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

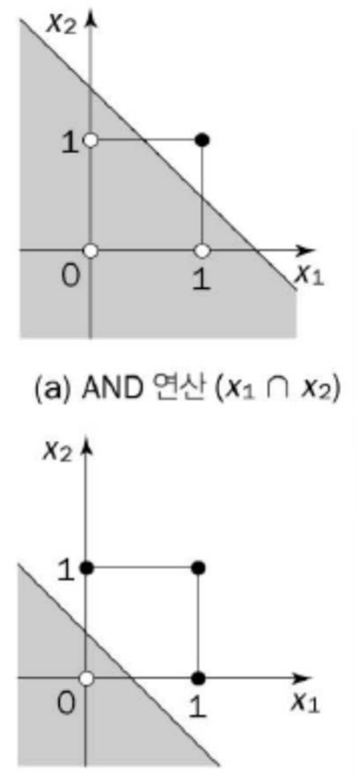
- AND와 OR 연산자 학습
  - 학습 순서

    1. 하나의 에폭(epoch)을 나타내는 네 개의 연속된 입력 패턴으로 퍼셉트론을 활성화
    2. 퍼셉트론의 가중치는 각각의 활성화 이후에 갱신
    3. 가중치가 일관된 수치 집합으로 수렴할 때까지 이 과정을 반복

퍼셉트론

- AND 논리 연산자의 퍼셉트론 학습 결과 (θ : 0.2, α : 0.1)

	입력		목표 출력	초기 가중치		실제 출력	오차	최종 가중치	
에폭	x_1	x_2	Y_d	w_1	w_2	Y	e	w_1	w_2
1	0	0	0	0.3	−0.1	0	0	0.3	−0.1
	0	1	0	0.3	−0.1	0	0	0.3	−0.1
	1	0	0	0.3	−0.1	1	−1	0.2	−0.1
	1	1	1	0.2	−0.1	0	1	0.3	0.0
2	0	0	0	0.3	0.0	0	0	0.3	0.0
	0	1	0	0.3	0.0	0	0	0.3	0.0
	1	0	0	0.3	0.0	1	−1	0.2	0.0
	1	1	1	0.2	0.0	1	0	0.2	0.0
3	0	0	0	0.2	0.0	0	0	0.2	0.0
	0	1	0	0.2	0.0	0	0	0.2	0.0
	1	0	0	0.2	0.0	1	−1	0.1	0.0
	1	1	1	0.1	0.0	0	1	0.2	0.1
4	0	0	0	0.2	0.1	0	0	0.2	0.1
	0	1	0	0.2	0.1	0	0	0.2	0.1
	1	0	0	0.2	0.1	1	−1	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1
5	0	0	0	0.1	0.1	0	0	0.1	0.1
	0	1	0	0.1	0.1	0	0	0.1	0.1
	1	0	0	0.1	0.1	0	0	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1



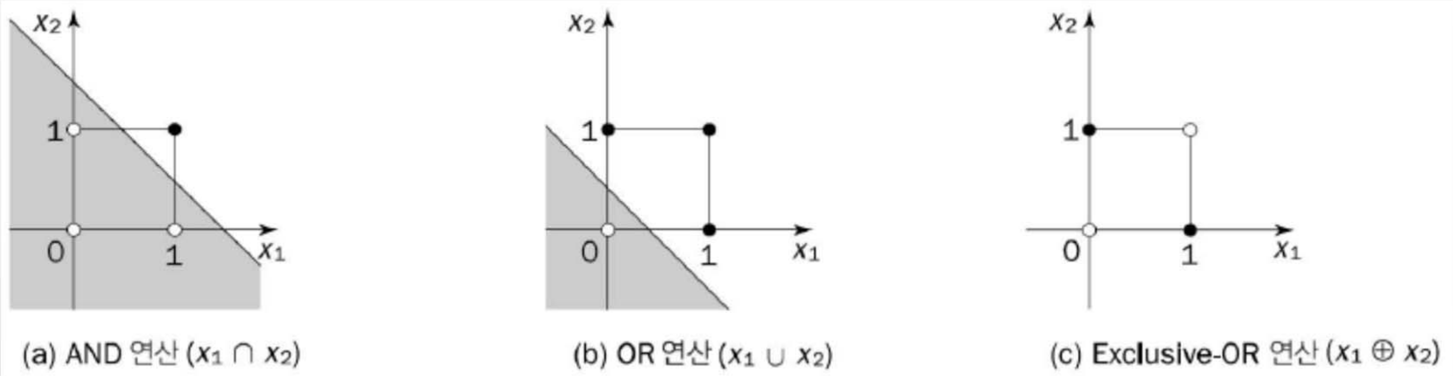
(a) AND 연산 ($x_1 \cap x_2$)

(b) OR 연산 ($x_1 \cup x_2$)

➢ 검정색 : 함수의 출력이1인 입력공간의 점; 흰색 : 출력이 0인 점

퍼셉트론

- Exclusive-OR 연산자 학습
  - 단층 퍼셉트론으로 학습 불가
    - ➢ 선형 분리가 불가능하기 때문

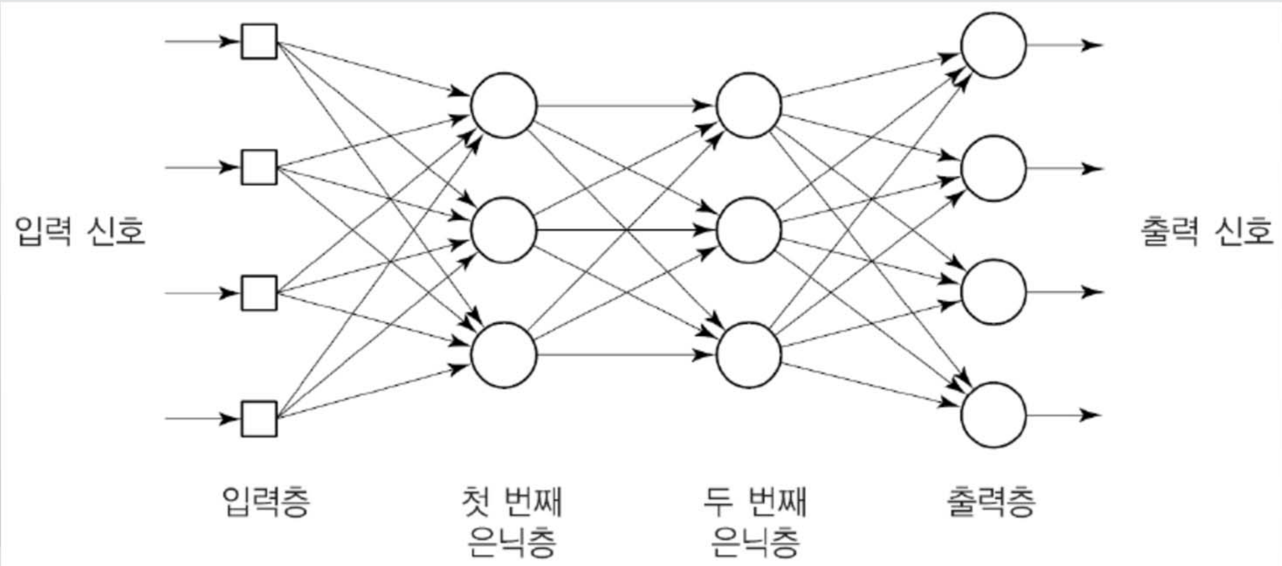


- ➢ (a)와 (b)에서는 검은 점과 흰 점을 구분하여 직선을 그릴 수 있지만, (c)의 점들은 직선으로 분리할 수 없음
- ➢ 퍼셉트론은 모든 검은점과 모든 흰점을 분리하는 직선이 있을 때만 함수로 표현가능

- 단층 퍼셉트론 학습의 한계와 극복
  - 단층 퍼셉트론은 선형 분리 가능한 함수만 학습 가능
    - ➢ 그러나 많은 경우 선형 분리 불가
  - 다층 신경망으로 단층 퍼셉트론의 한계 극복

다층 신경망(Multi-Layer Perceptron)

- 다층 신경망의 구조
  - 하나 혹은 그 이상의 은닉층이 있는 순방향 신경망(**feedforward neural network**)
    - 공급 뉴런으로 이루어진 입력층(**input layer**) 한 개
    - 계산 뉴런들로 이루어진 하나 이상의 은닉층(**hidden layer**)
    - 계산 뉴런들로 이루어진 출력층(**output layer**) 한개
  - 입력 신호는 한 층씩 순방향으로 전파
  - 다층 신경망의 예 : 두 개의 은닉층이 있는 다층 신경망



다층 신경망 (MLP)

- 다층 신경망의 구조
  - 각 층에는 각각 자신만의 특성 함수 존재 : non-liner activation function
  - 각 층간에는 방향성이 존재

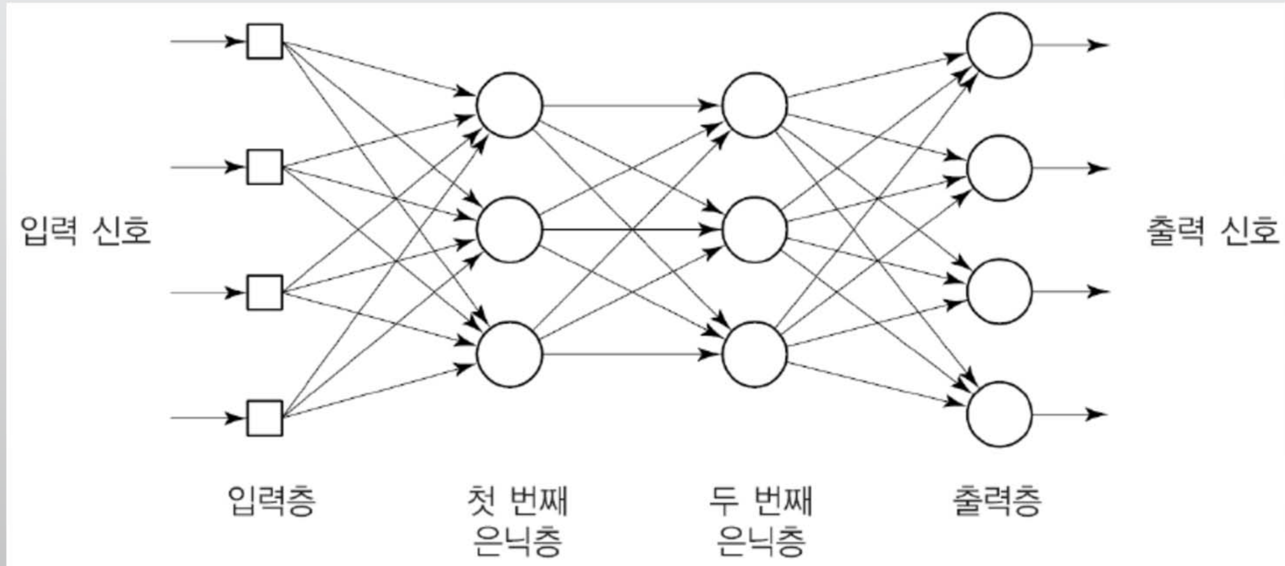
  - 입력층
    - 외부에서 받아들인 입력 신호를 은닉층의 모든 뉴런으로 보냄
    - 계산을 위한 뉴런은 거의 들어 있지 않음

  - 출력층
    - 은닉층에서 출력 신호, 즉 자극 패턴을 받아 들이고 전체 신경망의 출력 패턴을 결정

  - 은닉층
    - 은닉층 뉴런은 신경망의 입출력 동작을 통해 관찰되지 않음
    - 입력의 특성을 파악
    - 뉴런의 가중치는 입력 패턴에 숨겨져 있는 특성을 나타냄
      - 출력층이 출력 패턴을 정할 때, 이 특성을 사용
    - **은닉층의 목표 출력은 해당 층에서 자체적으로 결정** → 목표 출력이 '**은닉**'
    - 두 개 이상의 은닉층 가능 ➔ 보통 1개 은닉층 사용, 왜?

다층 신경망의 학습

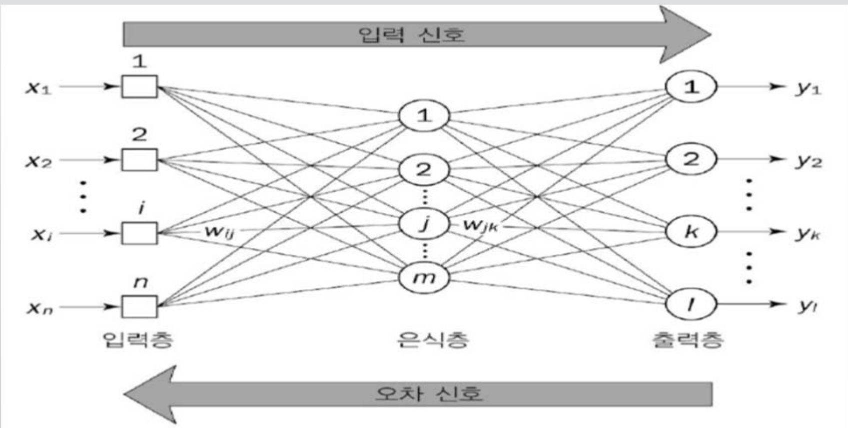
- 학습 데이터 셋에 대해 네트워크의 오차를 최소화하는 **parameter**들을 계산
  - 파라미터 : 가중치와 바이어스
- 다층 신경망의 학습은 퍼셉트론과 유사하게 진행
- 신경망은 출력 패턴을 계산하고 오차가 있다면(실제와 목표 출력 간에 차이가 있다면) 이 오차를 줄이도록 가중치를 조절
- 다층 신경망에서 각각의 가중치는 두 개 이상의 출력에 영향



역전파 학습 알고리즘 (Back Propagation Algorithm)

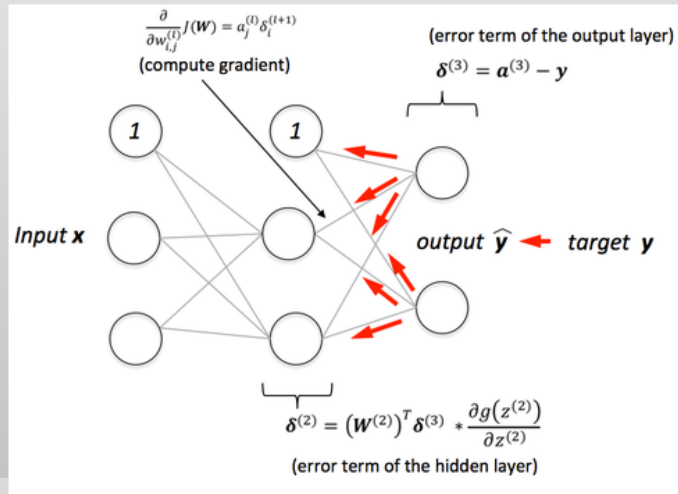
- 역전파 신경망
  - 지도 학습(supervised learning) 알고리즘
  - 세 개 또는 네 개의 층이 있는 다층 신경망의 오차 원인을 평가
  - 입력데이터에 대한 예측 연산인 forward 연산 후, 예측값과 정답과의 오차를 backward로 보내면서 weight와 bias 학습
  - 출력에 영향을 미치는 여러 가중치들 사이에서 **오차의 원인을 정하고 나누는 데 사용**
  - 역전파 학습은 널리 쓰이고 있지만, 문제에 대한 면역성이 없음
  - 계산이 방대하기 때문에 학습이 느림

→ 역전파 학습을 인간의 뇌와 같은 학습 방법을 흉내 낸 과정이라 보기 어려움



역전파 망 구조 : 층이 세 개인 신경망

i, j, k : 각각 입력층, 은닉층, 출력층 뉴런



다층 신경망 (MLP)

DANKOOK UNIVERSITY
Machine Learning & Pattern Analysis lab.

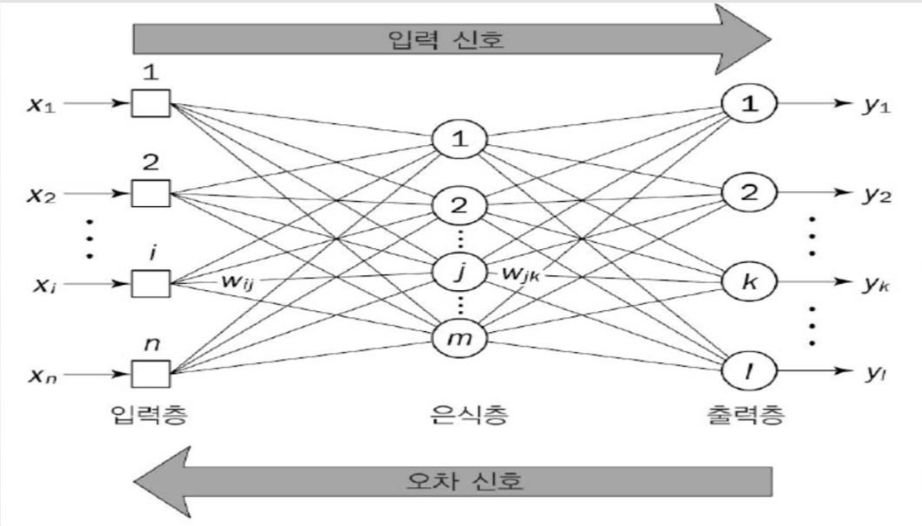
- 역전파 신경망의 학습 알고리즘
  - 학습 알고리즘의 두 단계
    - ➢ 순방향(feedforward)

      1. 학습 샘플의 패턴을 신경망의 입력층에 전달

      2. 신경망은 출력층에서 출력 패턴이 생성될 때까지 층에서 층으로 입력 패턴을 전파
    - ➢ 역방향(backward)

      1. 출력 패턴이 목표 패턴과 다르면 그 오차를 계산한 후 출력층에서 입력층까지 신경망을 따라 거꾸로 전파

      2. 오차가 전파되면서 가중치가 수정

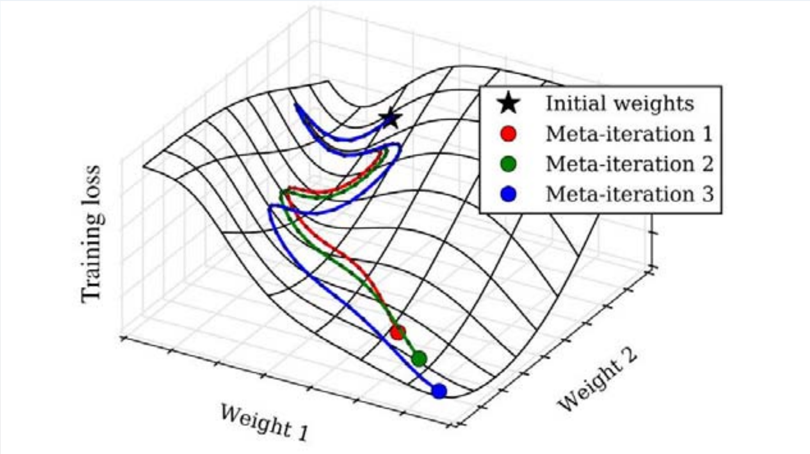


역전파 망 구조 : 층이 세 개인 신경망

i, j, k : 각각 입력층, 은닉층, 출력층 뉴런

- Gradient descent
  - 너무 많은 가중치 조합을 모두 계산하면 시간이 오래 걸리기 때문에 이를 효율적으로 하기 위해서 고안된 방법
  - 단계적으로 접근하는 것이기 때문에 만족스러운 정확도에 이를 때까지 계속해서 답을 찾아나가는 방식.
  - 정확한 답은 얻지 못할 수 있음.

- Gradient descent

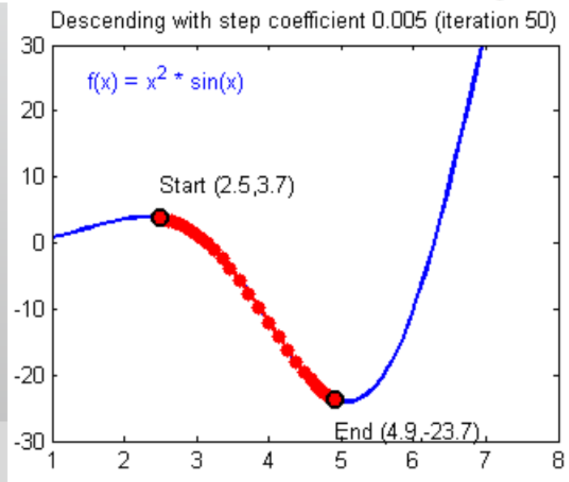


- Parameter θ 로 미분 가능한 Objective 값 $J(\theta)$를 최소화 하는 방법
- Parameter space를 한 눈에 볼 수 있다면 별도의 알고리즘 불필요
- 너무 많은 가중치 조합을 모두 계산하면 시간이 오래 걸리기 때문에 이를 효율적으로 하기 위해서 고안된 방법
  - 단계적으로 접근
  - 만족스러운 정확도에 이를 때까지 계속해서 답을 찾아나가는 방식
  - 정확한 답은 얻지 못할 수 있음
- 주변 값들에 대한 정보는 없음
  - 현재 위치에서의 미분값을 이용
  - 가장 가파르게 내려가는 방향을 선택하여 정해진 양 만큼 이동

다층 신경망 (MLP)

- Gradient descent
  - 신경망은 최종 단에서 틀린 정도(Loss Function)를 가지고 있음
  - 앞이 보이지 않는 상황에서 산의 정상에 올라가기 위해서는 발로 주변을 디뎠을 때 오르막 경사가 있는 곳으로 이동하다 보면 정상에 올라갈 수 있을 것이라는 생각에서 착안
    - ➢ 현재의 weight 세팅(산으로 올라가는 과정 중 현재 자리)에서 내가 가진 데이터를 넣으면 전체 에러가 계산됨
    - ➢ 그 자리에서의 미분이 에러를 줄이는 방향임
- 에러를 줄이는 방향으로 정해진 스텝량(learning rate)를 곱해서 weigh를 갱신하는 것을 반복한다.

weight의 업데이트 = 에러 낮추는 방향 (decent) X 한발자국 크기 (learning rate) X 현 지점의 기울기 (gradient)



- Stochastic Gradient Descent
  - Gradient $\nabla f(x,y)$
  - 입력 변수 각각에 대한 편미분으로 이루어진 vector

$$\nabla f(x,y) \triangleq [\nabla_x f(x,y) \quad \nabla_y f(x,y)] = \left[\frac{\partial f(x,y)}{\partial x} \quad \frac{\partial f(x,y)}{\partial y}\right]$$

➢ 편미분 (partial derivative)

» 함수 $f(x,y)$에 대한 변수 x의 편미분 : x에 의한 $f(x,y)$의 sensitivity

» Gradient : 함수 f의 변화량이 가장 큰 방향

» Ex) 함수$f(x,y,z)$가 다음과 같이 주어졌을 때, 이 함수의 gradient는?

$$f(x,y,z) = (x+y)z$$

$$\rightarrow \frac{\partial f(x,y,z)}{\partial x} = \frac{\partial}{\partial x}(x+y)z = z, \frac{\partial f(x,y,z)}{\partial y} = \frac{\partial}{\partial y}(x+y)z = z, \frac{\partial f(x,y,z)}{\partial z} = \frac{\partial}{\partial z}(x+y)z = x+y$$

$$\therefore \nabla f(x,y,z) = [z \quad z \quad x+y]$$

다층 신경망 (MLP)

- Gradient descent
  - 매 위치(특정한 θ 값)에서 J(θ)를 가장 급격하게 감소시키는 방향(gradient)으로 정해진 양(α, step size)만큼 이동

Step size (machine learning 분야에서는 learning rate으로 표기)

$$\theta_{t+1} = \theta_t - \alpha \times \nabla J(\theta_t)$$

» Note: θ는 vector

  - 더 이상 유의미한 이동이 없을 시 (gradient가 0에 가까울 시) iteration의 종료

$\|J(\theta_t) - J(\theta_{t+1})\| < \epsilon$ → iteration 종료

» $J(x_0) \geq J(x_1) \geq J(x_2)$ ...

→ Sequence x_n converges to the desired local minimum

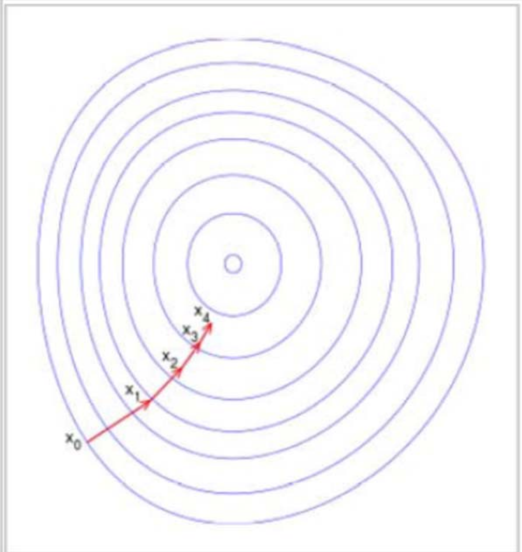


Illustration of gradient descent.

- Stochastic Gradient Descent
  - Gradient에 대해 stochastic approximation 한 것
    - Stochastic approximation
      - 입력 데이터에 noise가 존재할 때, 샘플들에 의한 함수 값의 평균으로 추정

$$f(x,\delta) = \mathbb{E}_{\delta}[f(x,\delta)]$$

      - Noise의 분포를 모를 시에는 uniform으로 가정
      - N개의 data에 대한 gradient의 stochastic approximation :

$$\nabla J(\theta) = \sum_{n=1}^{N} \nabla J_n(\theta)$$

    - Stochastic gradient descent의 매 iteration 식

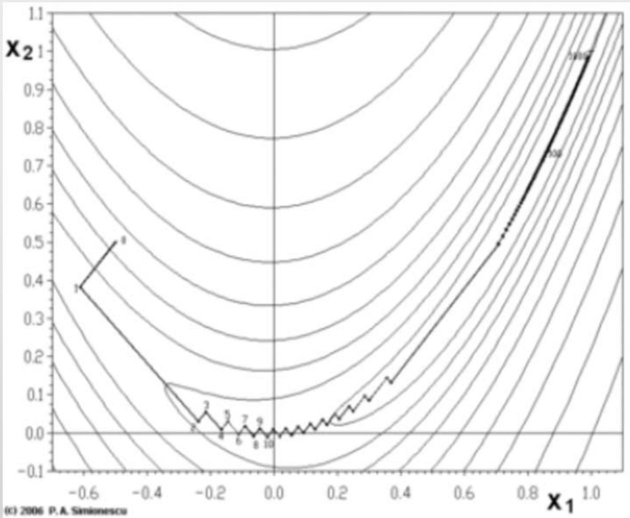
$$\theta_{t+1} = \theta_t - \alpha \times \nabla J(\theta_t) = \theta_t - \alpha \times \sum_{n=1}^{N} \nabla J_n(\theta)$$

다층 신경망 (MLP)

- Stochastic Gradient Descent
  - 한계
    - Minimum에 도달하는데 상대적으로 오래 걸림
      - » Ex. Rosenbrock function

$$f(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$$

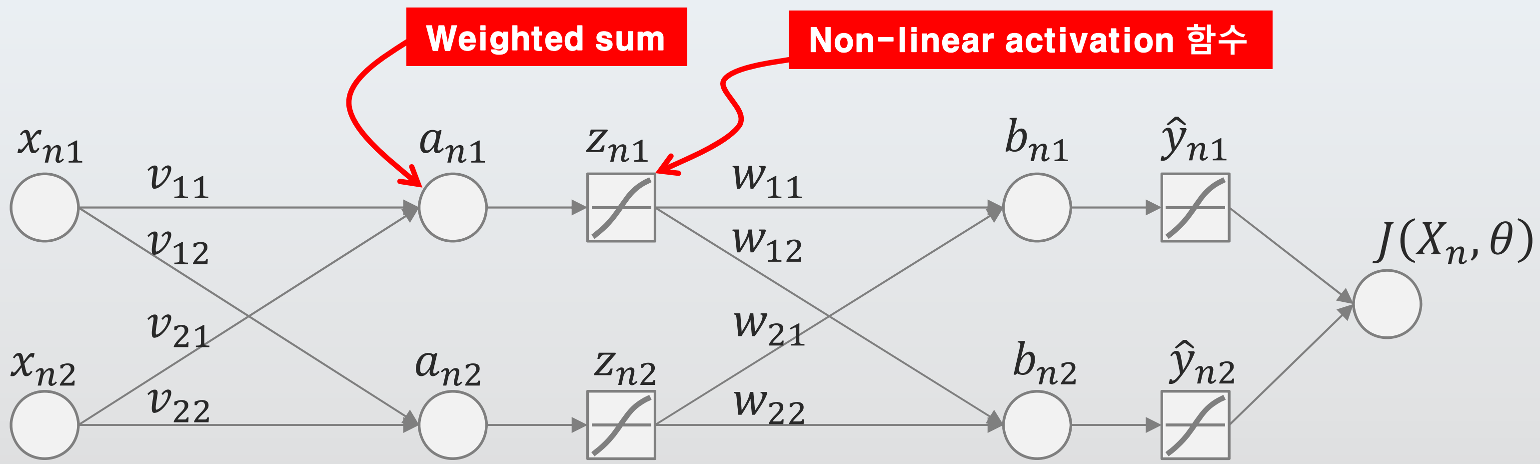
- Narrow curved valley which contains the minimum
- The bottom of the valley is very flat
- The optimization is zig-zagging slowly with small stepsizes towards the minimum



- 미분 불가능 함수들에 대해서는 gradient method가 ill-defined

다층 신경망 (MLP)

- Back Propagation Algorithm
  - Example network



$$X_n \xrightarrow{V} A_n \xrightarrow{g} Z_n \xrightarrow{W} B_n \xrightarrow{h} \hat{Y}_n$$

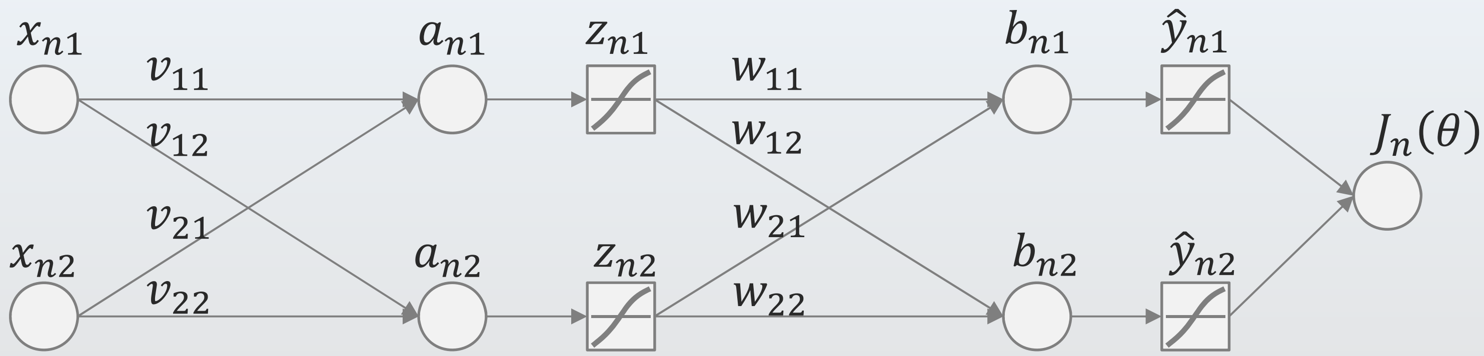
$$Z_n = g(A_n) \qquad \hat{Y}_n = h(B_n)$$

➢ n번째 입력 data : $X_n = [x_{n1} \quad x_{n2}]^T$

➢ Parameter θ=(V,W) : 입력 데이터는 고정 값으로 간주 $J(X_n, \theta) \rightarrow J_n(\theta)$

$$V = \begin{bmatrix} v_{11} & v_{12} \\ v_{21} & v_{22} \end{bmatrix} = [V_1 \quad V_2], \qquad W = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} = [W_1 \quad W_2]$$

- Example network



- Stochastic Gradient Descent
  - » Gradient의 stochastic approximation

$$\nabla J(\theta) = \sum_n \nabla J_n(\theta) = \left[\sum_n \nabla_{V_1} J_n(\theta);\ \sum_n \nabla_{V_2} J_n(\theta);\ \sum_n \nabla_{W_1} J_n(\theta);\ \sum_n \nabla_{W_2} J_n(\theta) \right]$$

$$\begin{bmatrix} \frac{\partial J_n}{\partial v_{11}} \\ \frac{\partial J_n}{\partial v_{12}} \end{bmatrix}$$

각 편미분을 어떻게 구할 수 있을까

다층 신경망 (MLP)

- 편미분의 chain rule
  - 함수 $f(x)=g(h(x))$에 대해

$$\frac{\partial f}{\partial x} = \frac{\partial g}{\partial h}\frac{\partial h}{\partial x}$$

  - Ex) 함수 $f(x, y, z)$가 다음과 같이 주어졌을 때, 이 함수의 gradient는?

$$\mathrm{q} = x + y, \qquad f(x, y, z) = qz$$

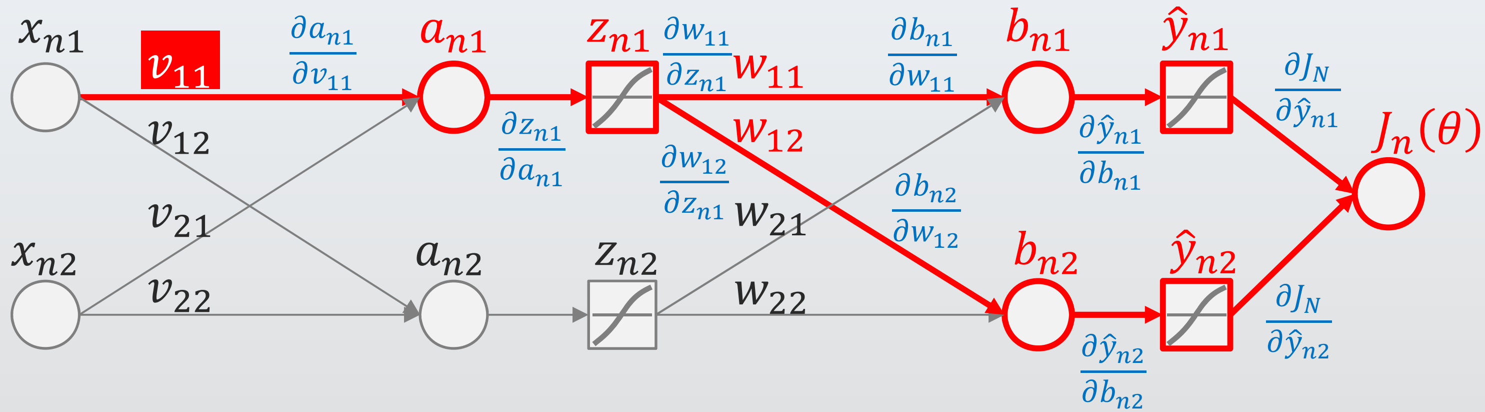
$$\rightarrow \frac{\partial f(x,y,z)}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x} = \frac{\partial (qz)}{\partial q}\frac{\partial (x+y)}{\partial x} = z \cdot 1 = z,$$

$$\frac{\partial f(x,y,z)}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y} = \frac{\partial (qz)}{\partial q}\frac{\partial (x+y)}{\partial y} = z \cdot 1 = z,$$

$$\frac{\partial f(x,y,z)}{\partial z} = \frac{\partial}{\partial z}(x + y)z = x + y$$

$$\therefore \nabla f(x, y, z) = [z \quad z \quad x + y]$$

- Example network
  - v_{11}에 의한 $J_n(\theta)$의 편미분 구하기



  - v_{11}으로부터 $J_n(\theta)$까지의 경로와 local gradient를 계산
    - » Local gradient 계산
      - → Ex. activation 함수가 sigmoid 함수라면

Feed-forward를 통해 구한 값을 대입

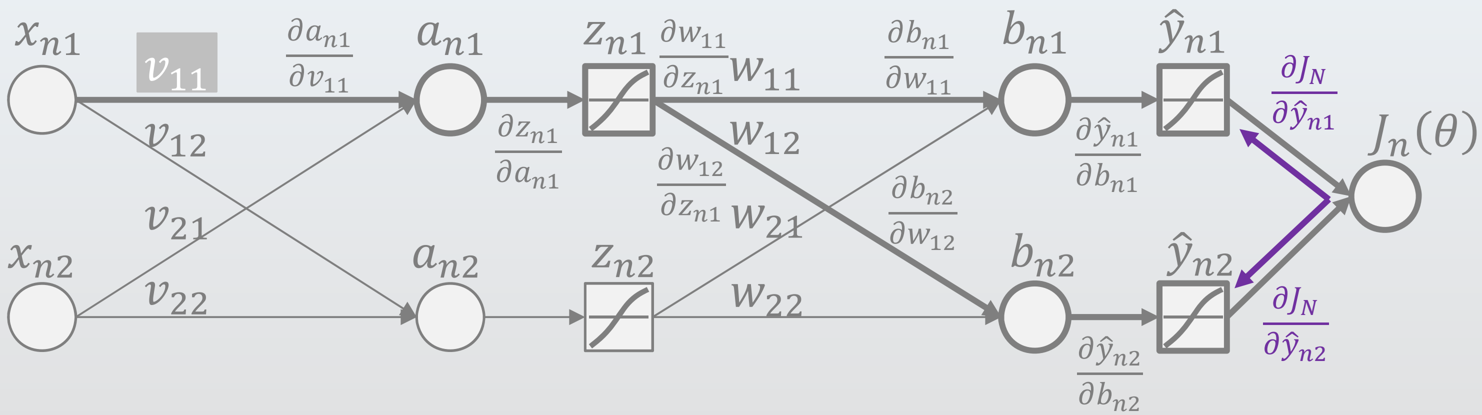
$$z_{n1}(a_{n1}) = sigm(a_{n1}) = \frac{1}{1+e^{-a_{n1}}}$$

$$\frac{\partial z_{n1}}{\partial a_{n1}} = \left(1 - \frac{1}{1+e^{-a_{n1}}}\right)\frac{1}{1+e^{-a_{n1}}} = \frac{e^{a_{n1}}}{(1+e^{a_{n1}})^2}$$

다층 신경망 (MLP)

DANKOOK UNIVERSITY
Machine Learning & Pattern Analysis lab.

- Example network
  - v_{11}에 의한 $J_n(\theta)$의 편미분 구하기

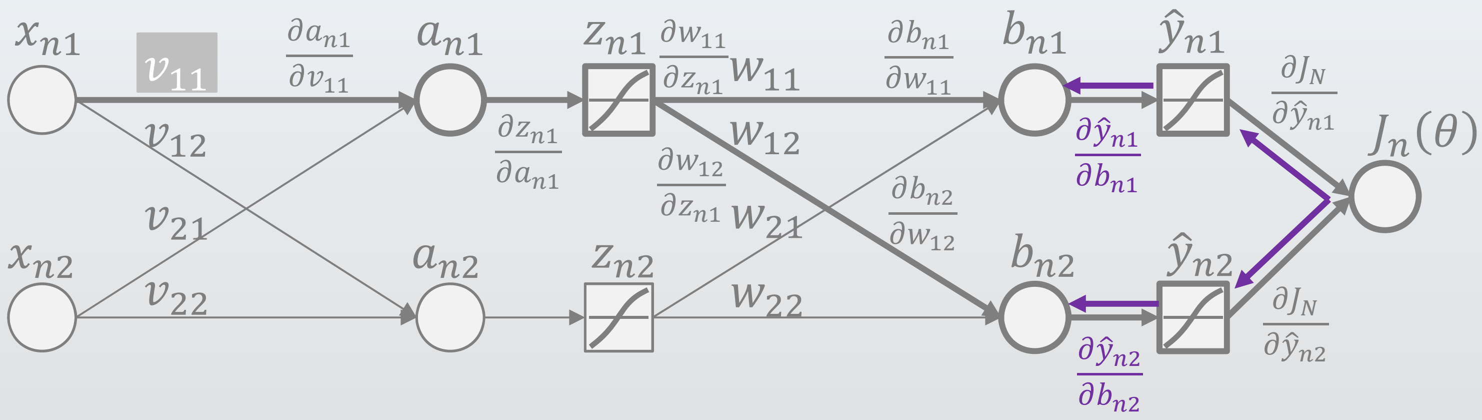


$$\frac{\partial J_N}{\partial v_{11}} = \frac{\partial J_N}{\partial \hat{y}_{n1}} \qquad + \frac{\partial J_N}{\partial \hat{y}_{n2}}$$

다층 신경망 (MLP)

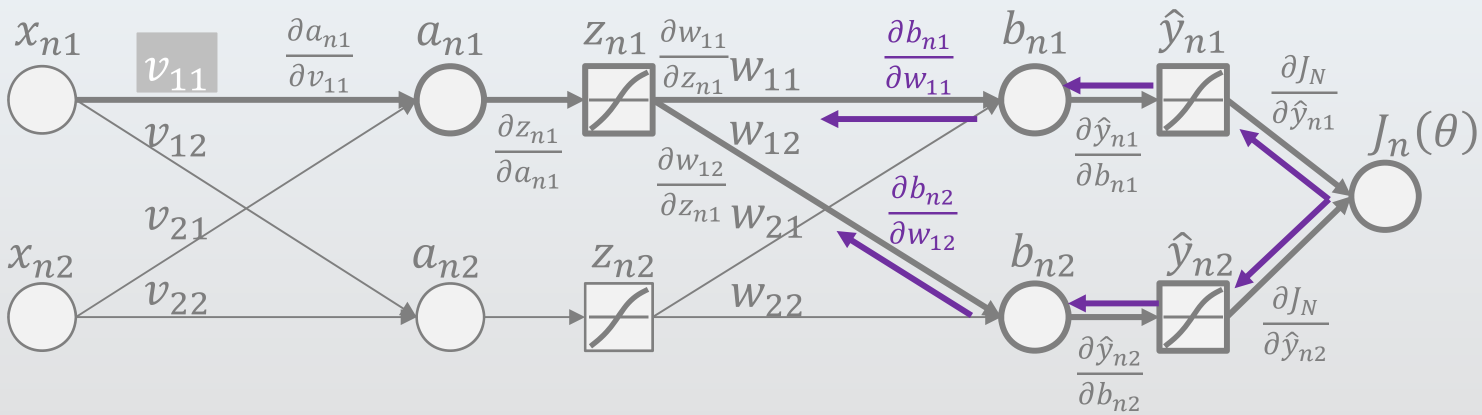
DANKOOK UNIVERSITY Machine Learning & Pattern Analysis lab.

- Example network
  - v_{11}에 의한 $J_n(\theta)$의 편미분 구하기



$$\frac{\partial J_N}{\partial v_{11}} = \frac{\partial J_N}{\partial \hat{y}_{n1}} \frac{\partial \hat{y}_{n1}}{\partial b_{n1}} + \frac{\partial J_N}{\partial \hat{y}_{n2}} \frac{\partial \hat{y}_{n2}}{\partial b_{n2}}$$

- Example network
  - v_{11}에 의한 $J_n(\theta)$의 편미분 구하기

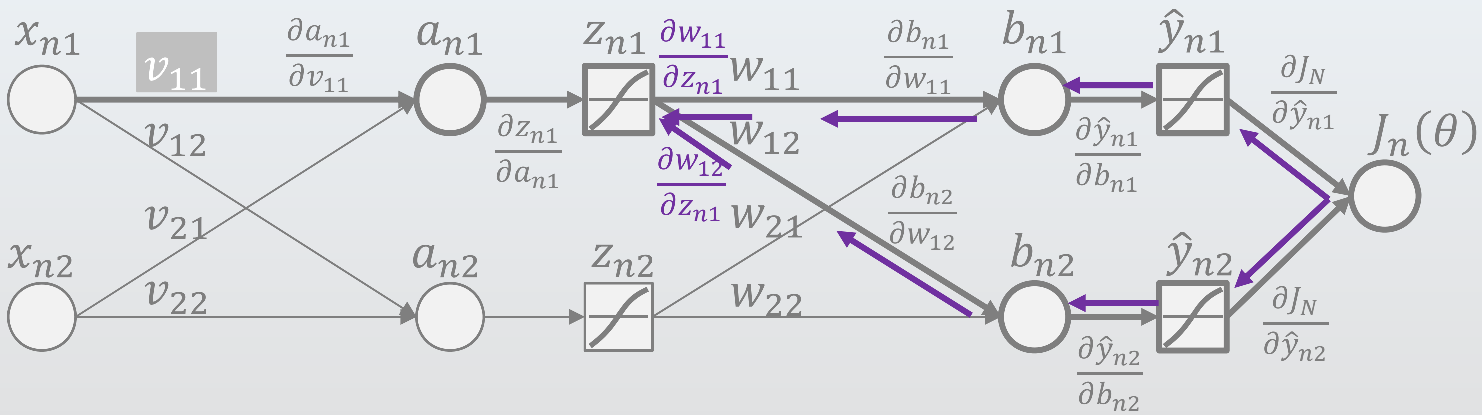


$$\frac{\partial J_N}{\partial v_{11}} = \frac{\partial J_N}{\partial \hat{y}_{n1}} \frac{\partial \hat{y}_{n1}}{\partial b_{n1}} \frac{\partial b_{n1}}{\partial w_{11}} + \frac{\partial J_N}{\partial \hat{y}_{n2}} \frac{\partial \hat{y}_{n2}}{\partial b_{n2}} \frac{\partial b_{n2}}{\partial w_{12}}$$

다층 신경망 (MLP)

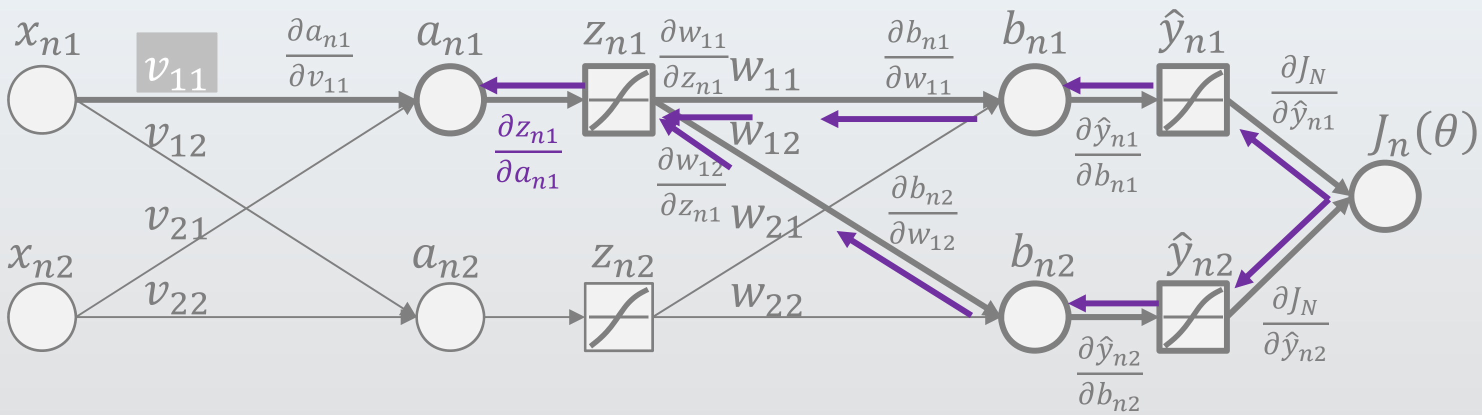
DANKOOK UNIVERSITY Machine Learning & Pattern Analysis lab.

- Example network
  - v_{11}에 의한 $J_n(\theta)$의 편미분 구하기



$$\frac{\partial J_N}{\partial v_{11}} = \left(\frac{\partial J_N}{\partial \hat{y}_{n1}} \frac{\partial \hat{y}_{n1}}{\partial b_{n1}} \frac{\partial b_{n1}}{\partial w_{11}} \frac{\partial w_{11}}{\partial z_{n1}} + \frac{\partial J_N}{\partial \hat{y}_{n2}} \frac{\partial \hat{y}_{n2}}{\partial b_{n2}} \frac{\partial b_{n2}}{\partial w_{12}} \frac{\partial w_{12}}{\partial z_{n1}} \right)$$

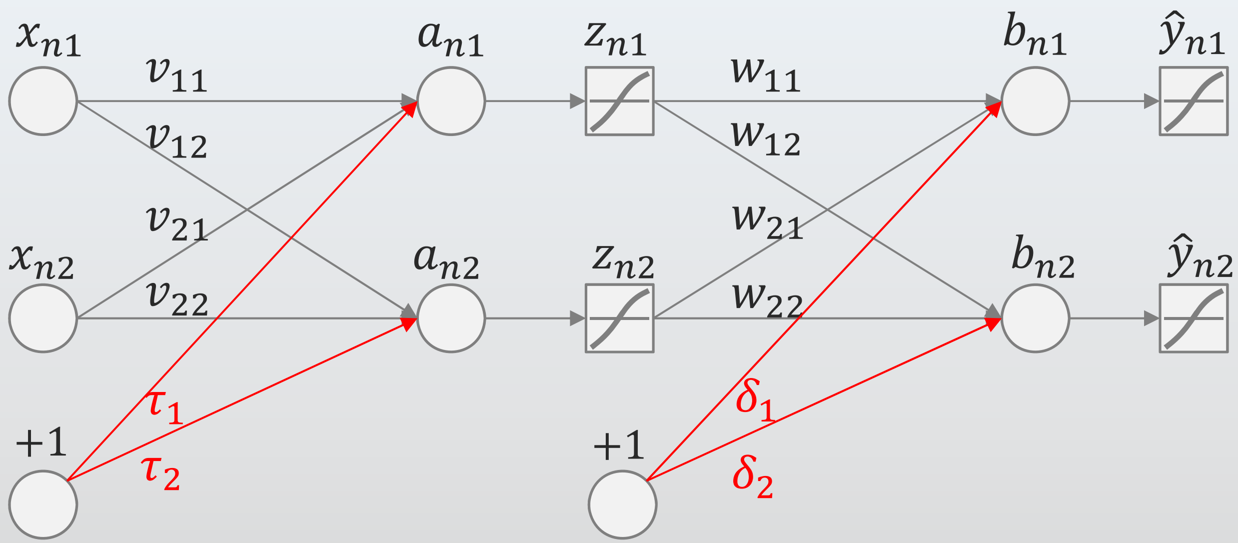
- Example network
  - v_{11}에 의한 $J_n(\theta)$의 편미분 구하기



$$\frac{\partial J_N}{\partial v_{11}} = \left(\frac{\partial J_N}{\partial \hat{y}_{n1}} \frac{\partial \hat{y}_{n1}}{\partial b_{n1}} \frac{\partial b_{n1}}{\partial w_{11}} \frac{\partial w_{11}}{\partial z_{n1}} + \frac{\partial J_N}{\partial \hat{y}_{n2}} \frac{\partial \hat{y}_{n2}}{\partial b_{n2}} \frac{\partial b_{n2}}{\partial w_{12}} \frac{\partial w_{12}}{\partial z_{n1}} \right) \frac{\partial z_{n1}}{\partial a_{n1}} \frac{\partial a_{n1}}{\partial v_{11}}$$

다층 신경망 (MLP)

- Example network
  - Bias (τ)는?



  - Parameter와 출력을 다음과 같이 수정 후, 전과 동일하게

$$X'_n \triangleq \begin{bmatrix} x_{n1} \\ x_{n2} \\ 1 \end{bmatrix} \qquad Z'_n \triangleq \begin{bmatrix} z_{n1} \\ z_{n2} \\ 1 \end{bmatrix}$$

Bias는 activation이 항상 1인 neuron과 같음

$$V'_k \triangleq \begin{bmatrix} V_k \\ \tau_k \end{bmatrix} \qquad W'_k \triangleq \begin{bmatrix} W_k \\ \delta_k \end{bmatrix}$$

- Example network
  - Pooling layer에서는?

$$X_n \xrightarrow{V} A_n \xrightarrow{g} Z_n \xrightarrow{W} B_n \xrightarrow{h} \hat{Y}_n$$

$$Z_n = g(A_n) \qquad \hat{Y}_n = h(B_n)$$

$$\therefore \delta_{nj}^{v} = \sum_k \delta_{nk}^{w} w_{kj} g'(a_{nj})$$

Non-linear 함수의 미분 값을 정의하면 어떤 layer도 back propagation 가능

» 함수의 미분을 다음과 같이 간주

Average pooling $\quad g'(x) = \frac{1}{m}$

Max pooling $\quad \frac{\partial g(x)}{\partial x_i} = \begin{cases} 1 & if \ \ x_i = \max(x) \\ 0 & otherwise \end{cases}$

다층 신경망 (MLP)

- Implementation 초기화
  - Feed-forward network 정의
  - Network의 parameter에 대한 초기화
    - » Weight에 0이 아닌 값을 주어야 정상 작동
  - Local gradient를 구하기 위한 식 기입

- 구동
  - 데이터 x_n에 대해 Feed-forward 과정을 거치면서 local gradient와 최종 error 계산
  - Local gradient를 chain rule에 대입하여 각 파라미터에 대한 편미분 값 계산
  - 하나의 epoch (mini-batch, $\{x_1, \ldots, x_N\}$)에 대해 편미분 값 합산으로 gradient 계산
  - Gradient를 통해 파라미터 조절

- Derivation
  - 한 개의 출력 뉴런에 대한 에러 함수를 정의
  - $E = \frac{1}{2}(y_d - y)^2 \quad cf.\ e = y_d - y$
    - E : squared error
      y_d : target output for a training sample
      y : actual output of the output neuron
  - 각 뉴런 k에 대한 출력 : $y_j = \varphi(X_k) = \varphi(\sum_{t=1}^{n} w_{tk} y_t)$
    - y_t: previous neuron
    - X_k: weighted sum of outputs of previous neurons
    - $\varphi(\cdot)$: activation function → general non-linear and differentiable function
      - » Ex. $\varphi(z) = \frac{1}{1+e^{-z}} \rightarrow \frac{d\varphi}{dz}(z) = \varphi(z)(1-\varphi(z))$

- Derivation
  - 에러에 대한 가중치의 partial derivative 계산
    - Using chain rule, $\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial net_k}\frac{\partial net_k}{\partial w_{jk}}$

» $\frac{\partial X_k}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}}\left(\Sigma_{t=1}^{n} w_{tj} y_t\right) = y_i$

→ 만약 o_k가 입력층 이후 첫번째 은닉층의 뉴런이라면 $y_k = x_k$

» $\frac{\partial y_k}{\partial X_k} = \frac{\partial}{\partial X_k}\varphi(X_k) = \varphi(X_k)(1-\varphi(X_k))$

1) y_k가 출력 뉴런일 때,

$$\frac{\partial E}{\partial y_k} = \frac{\partial E}{\partial y} = \frac{\partial}{\partial y}\frac{1}{2}(y_d - y)^2 = y_d - y$$

2) y_k가 임의의 은닉층의 뉴런일 때,

$\frac{\partial E(y_k)}{\partial y_k} = \frac{\partial E(X_u, X_v, \ldots, X_w)}{\partial y_k}$; $L = u, v, \ldots, w$: 뉴런 k로부터 입력을 받는 뉴런들

→ $\frac{\partial E}{\partial y_k} = \Sigma_{l \in L}\left(\frac{\partial E}{\partial X_l}\frac{\partial X_l}{\partial y_k}\right) = \Sigma_{l \in L}\left(\frac{\partial E}{\partial y_l}\frac{\partial y_l}{\partial X_l} w_{kl}\right)$

- Derivation
  - 에러에 대한 가중치의 partial derivative 계산
    - $\frac{\partial E}{\partial w_{jk}} = \delta_k y_j$
      - » $\delta_k = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial X_k} = \begin{cases} (y_k - y_d)y_k(1-y_k) = e_k y_k(1-y_k) & if\ k\ is\ an\ output\ neuron \\ \Sigma_{l\in L}\delta_l w_{jl} y_k(1-y_k) & if\ k\ is\ an\ inner\ neuron \end{cases}$
  - 가중치 업데이트
    - $w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p),\qquad (\Delta w_{jk}(p) = -\alpha\frac{\partial E(p)}{\partial w_{jk}(p)})$

은닉층 뉴런에 대한 가중치 보정값

- 역전파 학습 알고리즘
  - 총 4단계로 구성

- 1단계 : 초기화
  - 가중치 초기화는 각 뉴런별로 설정
    - ➢ 좁은 범위 안에서 균등 분포를 따라 임의의 수로 설정

$$\left(-\frac{2.4}{F_i},\ +\frac{2.4}{F_i}\right)$$

F_i는 신경망에 있는 뉴런 i의 총 입력 개수

- 2단계 : 활성화
  - 입력과 목표 출력을 적용하여 역전파 신경망을 활성화
  - 은닉층에 있는 뉴런의 실제 출력을 계산

$$y_j = \varphi(X_k) = \varphi(\Sigma_{t=1}^{n} w_{tk} y_t)$$

- ➢ n : 은닉층에 있는 뉴런 j의 입력 개수

- 3단계 : 가중치 학습
  - 출력 뉴런과 연관된 오차를 역방향으로 전파시키면서 역전파 신경망의 가중치를 갱신
  - 출력층에 있는 뉴런에 대해 오차 기울기를 계산

$$\delta_k(p) = y_k(p)\big(1 - y_k(p)\big)e_k(p)$$

  - 뉴런의 출력 가중치 갱신

$$w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p)$$

  - 은닉층에 있는 뉴런의 오차 기울기를 계산

$$\delta_j(p) = y_j(p)\left(1 - y_j(p)\right)\Sigma_{k=1}^{l}\delta_k(p)w_{jk}(p)$$

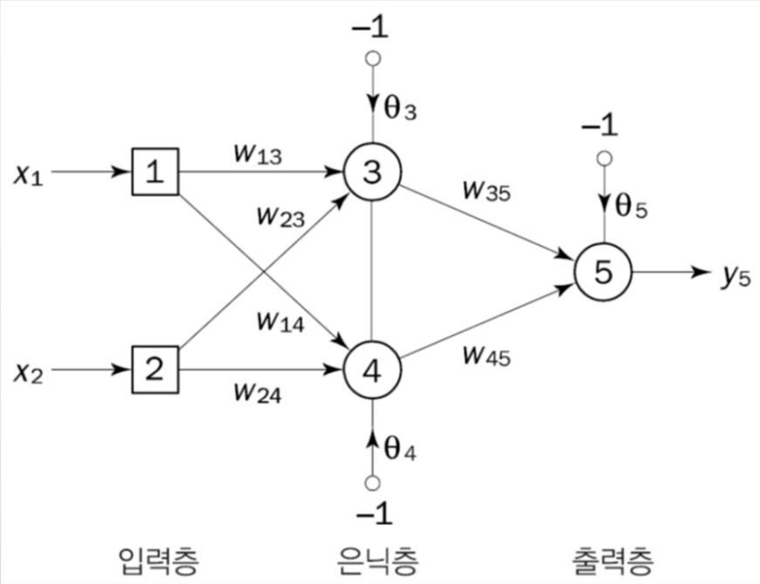
  - 은닉층에서의 가중치를 갱신

$$w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p)$$

- 4단계 : 반복
  - 반복 횟수 p값을 1 증가시키고, 2단계로 돌아가서 선택한 오차 기준을 만족할 때까지 과정을 반복한다.

다층 신경망

- Exclusive-OR 연산을 수행하기 위한 신경망
  - 층이 세 개인 역전파 신경망
  - Exclusive-OR 연산을 수행



➤ 각 가중치들과 임계값의 초기값 설정

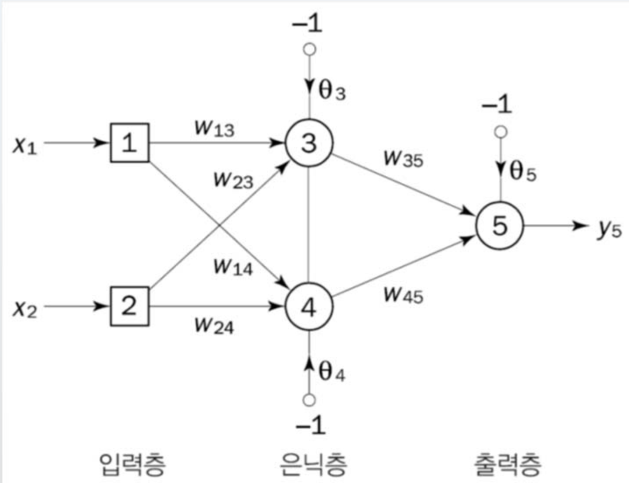
» 임의로 설정될 수 있으나, 초기 조건이 다르더라도 솔루션 도출 가능

$w_{13} = 0.5$, $w_{14} = 0.9$, $w_{23} = 0.4$, $w_{24} = 1.0$, $w_{35} = -1.2$, $w_{45} = 1.1$,
$\theta_3 = 0.8$, $\theta_4 = -0.1$ and $\theta_5 = 0.3$.

다층 신경망

DANKOOK UNIVERSITY Machine Learning & Pattern Analysis lab.

- Exclusive-OR 연산을 수행



- 순방향
  - 뉴런 1과 2에서 입력 신호를 받아 그대로 은닉층으로 전달

$$x_{13} = x_{14} = x_1 \text{ and } x_{23} = x_{24} = x_2$$

  - 뉴런 3과 4의 출력 값 계산

$$y_3 = sigmoid\,(x_1 w_{13} + x_2 w_{23} - \theta_3) = 1/[1 + e^{-(1\times0.5+1\times0.4-1\times0.8)}] = 0.5250$$

$$y_4 = sigmoid\,(x_1 w_{14} + x_2 w_{24} - \theta_4) = 1/[1 + e^{-(1\times0.9+1\times1.0+1\times0.1)}] = 0.8808$$

  - 뉴런 5에서의 출력 값 계산

$$y_5 = sigmoid\,(y_3 w_{35} + y_4 w_{45} - \theta_5) = 1/[1 + e^{-(-0.5250\times1.2+0.8808\times1.1-1\times0.3)}] = 0.5097$$

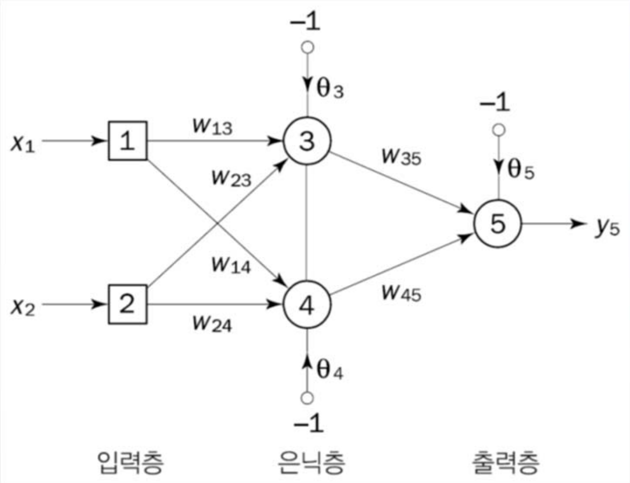
  - 에러 계산

$$e = y_{d,5} - y_5 = 0 - 0.5097 = -0.5097$$

다층 신경망

DANKOOK UNIVERSITY
Machine Learning & Pattern Analysis lab.

- Exclusive-OR 연산을 수행



- 역방향
  - 뉴런 5에서의 오차 기울기 계산

$$\delta_5 = y_5(1 - y_5)e = 0.5097 \times (1 - 0.5097) \times (-0.5097) = -0.1274$$

  - 가중치 보정값 계산 (α = 0.1)

$$\Delta w_{35} = \alpha \times y_3 \times \delta_5 = 0.1 \times 0.5250 \times (-0.1274) = -0.0067$$

$$\Delta w_{45} = \alpha \times y_4 \times \delta_5 = 0.1 \times 0.8808 \times (-0.1274) = -0.0112$$

$$\Delta \theta_5 = \alpha \times (-1) \times \delta_5 = 0.1 \times (-1) \times (-0.1274) = 0.0127$$

  - 뉴런 3과 4에서의 오차 기울기 계산

$$\delta_3 = y_3(1 - y_3) \times \delta_5 \times w_{35} = 0.5250 \times (1 - 0.5250) \times (-0.1274) \times (-1.2) = 0.0381$$

$$\delta_4 = y_4(1 - y_4) \times \delta_5 \times w_{45} = 0.8808 \times (1 - 0.8808) \times (-0.1274) \times 1.1 = -0.0147$$

다층 신경망

- 역방향
  - 가중치 보정값 계산 (α = 0.1)

$$\Delta w_{13} = \alpha \times x_1 \times \delta_3 = 0.1 \times 1 \times 0.0381 = 0.0038$$

$$\Delta w_{23} = \alpha \times x_2 \times \delta_3 = 0.1 \times 1 \times 0.0381 = 0.0038$$

$$\Delta \theta_3 = \alpha \times (-1) \times \delta_3 = 0.1 \times (-1) \times 0.0381 = -0.0038$$

$$\Delta w_{14} = \alpha \times x_1 \times \delta_4 = 0.1 \times 1 \times (-0.0147) = -0.0015$$

$$\Delta w_{24} = \alpha \times x_2 \times \delta_4 = 0.1 \times 1 \times (-0.0147) = -0.0015$$

$$\Delta \theta_4 = \alpha \times (-1) \times \delta_4 = 0.1 \times (-1) \times (-0.0147) = 0.0015$$

  - 가중치 갱신

$$w_{13} = w_{13} + \Delta w_{13} = 0.5 + 0.0038 = 0.5038$$

$$w_{14} = w_{14} + \Delta w_{14} = 0.9 - 0.0015 = 0.8985$$

$$w_{23} = w_{23} + \Delta w_{23} = 0.4 + 0.0038 = 0.4038$$

$$w_{24} = w_{24} + \Delta w_{24} = 1.0 - 0.0015 = 0.9985$$

$$w_{35} = w_{35} + \Delta w_{35} = -1.2 - 0.0067 = -1.2067$$

$$w_{45} = w_{45} + \Delta w_{45} = 1.1 - 0.0112 = 1.0888$$

$$\theta_3 = \theta_3 + \Delta \theta_3 = 0.8 - 0.0038 = 0.7962$$

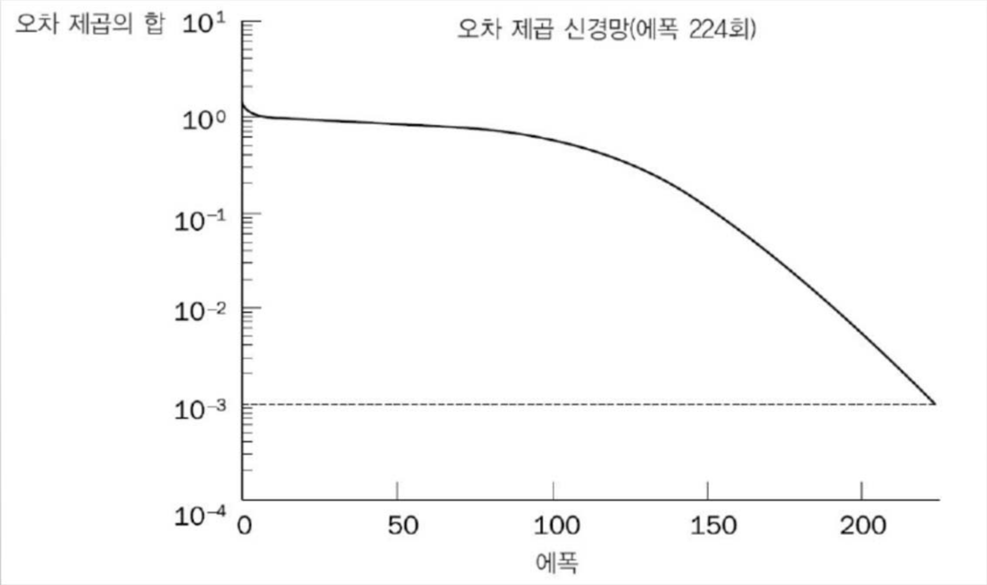
$$\theta_4 = \theta_4 + \Delta \theta_4 = -0.1 + 0.0015 = -0.0985$$

$$\theta_5 = \theta_5 + \Delta \theta_5 = 0.3 + 0.0127 = 0.3127$$

  - 오차 제곱의 합이 0.001 이하가 될 때까지 반복

다층 신경망

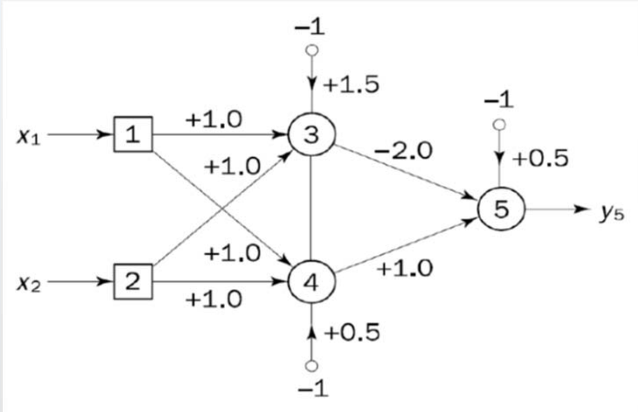
- 오차 제곱의 합(sum of the squared errors)
  - 신경망의 성능을 보여주는 유용한 지표
  - 한 패스 동안의 모든 훈련 집합 또는 에폭에 대한 오차 제곱의 합이 충분히 작으면 신경망이 수렴했다고 간주
  - 충분히 작은 오차 제곱의 합을 0.001로 설정했을 때의 학습 곡선



➢ 학습 곡선은 신경망이 얼마나 빨리 학습하고 있는지를 보여줌

다층 신경망

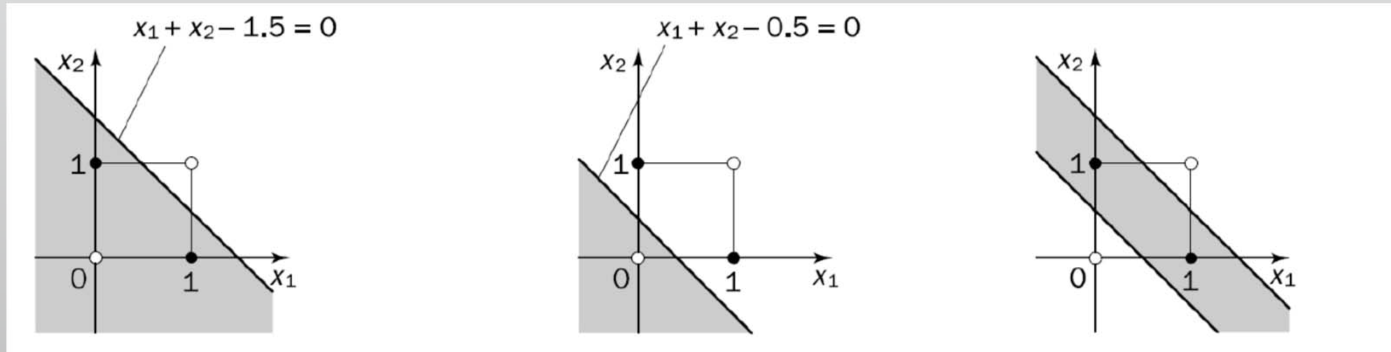
- Exclusive-OR 연산을 풀기 위한 신경망 (솔루션)



➢ 부호함수를 사용하는 맥클록과 피츠의 모델로 은닉층과 출력층에 있는 뉴런을 표현

➢ 시그모이드 활성화 함수를 사용하는 뉴런들이 만들어낸 결정 경계를 그리는 일은 조금 어려울 수 있음

➢ 신경망을 검은 점과 흰점으로 분리하여 Exclusive-OR 문제를 해결



(a) [그림 6-12]의 신경망에 있는 은닉 뉴런 3에 의해 형성된 결정 경계

(b) 은닉 뉴런 4에 의해 형성된 결정 경계

(c) 층이 3개인 완전한 신경망에 의해 형성된 결정 경계

다층 신경망에서의 가속 학습

가속 학습

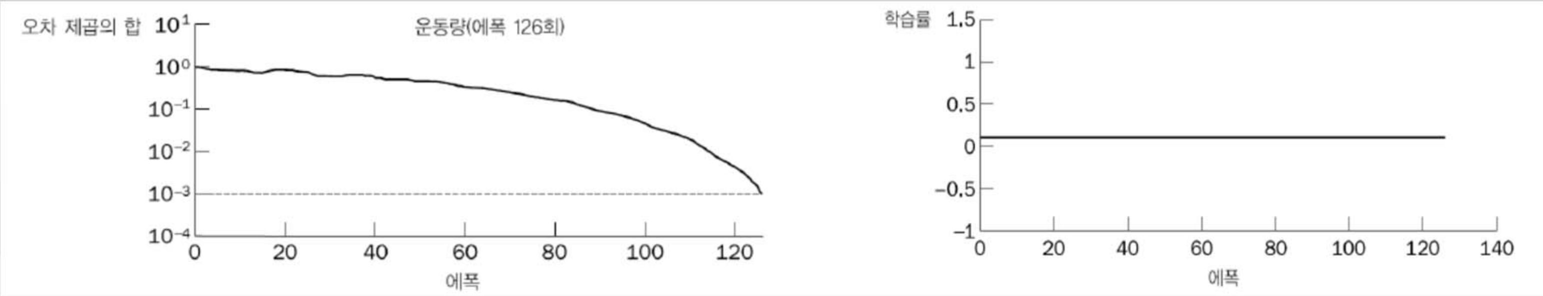
- 역전파 알고리즘의 계산 효율을 높이는 방법
  - 시그모이드 활성화 함수가 쌍곡 탄젠트로 표현될 때 조금 더 빠르게 학습됨

$$Y^{tan\,h} = \frac{2a}{1 + e^{-bX}} - a$$

    - a와 b는 상수로 a = 1.716, b = 0.667가 적당 (Guyon, 1991)
    - 델타규칙에 운동량 항(보통 0.95로 설정)을 포함시킴으로써 학습을 가속
  - 일반화된 델타 규칙

$$\Delta w_{jk}(p) = \beta \times \Delta w_{jk}(p-1) + \alpha \times y_j(p) \times \delta_k(p)$$

- 운동량 상수의 필요성
  - 역전파 알고리즘에 운동량을 포함시키면 학습하는 데 **안정화 효과**
    - 내리막 방향일 때는 하강하는 데 가속
    - 학습 표면이 봉우리와 골짜기 상태일 때는 학습 과정이 감속되는 경향
  - 운동량을 포함한 Exclusive-OR 연산 학습 결과 : 에폭의 횟수가 224에서 126으로 감소



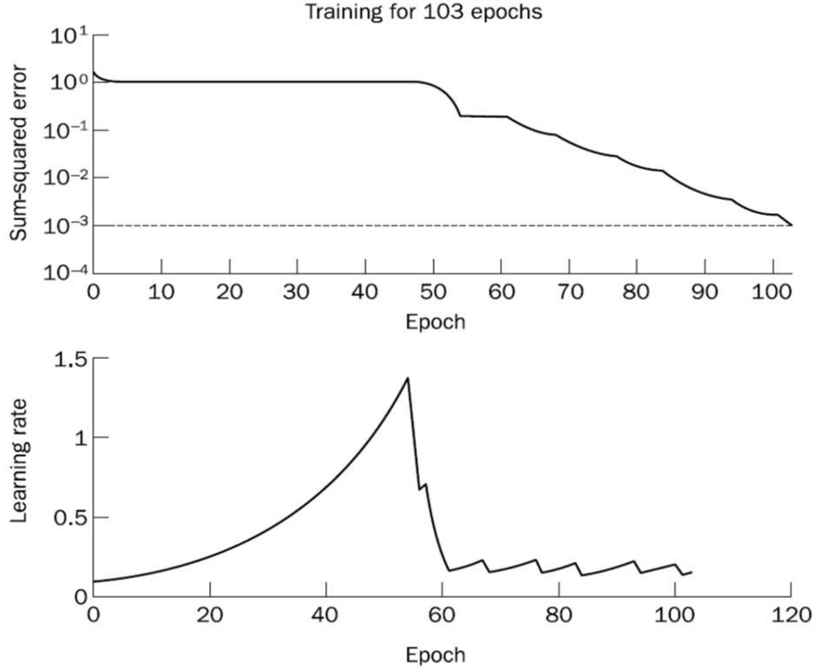
역전파 학습의 수렴 가속

- 훈련 중에 학습률 매개변수를 조절
  - 가장 효과적인 방법 중 하나임
  - 작은 학습률 α
    - 한 반복에서 그 다음으로 신경망의 가중치에 작은 변화
    - 매끄러운 학습 곡선
  - 큰 학습률 α
    - 가중치가 크게 변하므로 불안정한 상태
    - 학습 곡선의 빠른 수렴

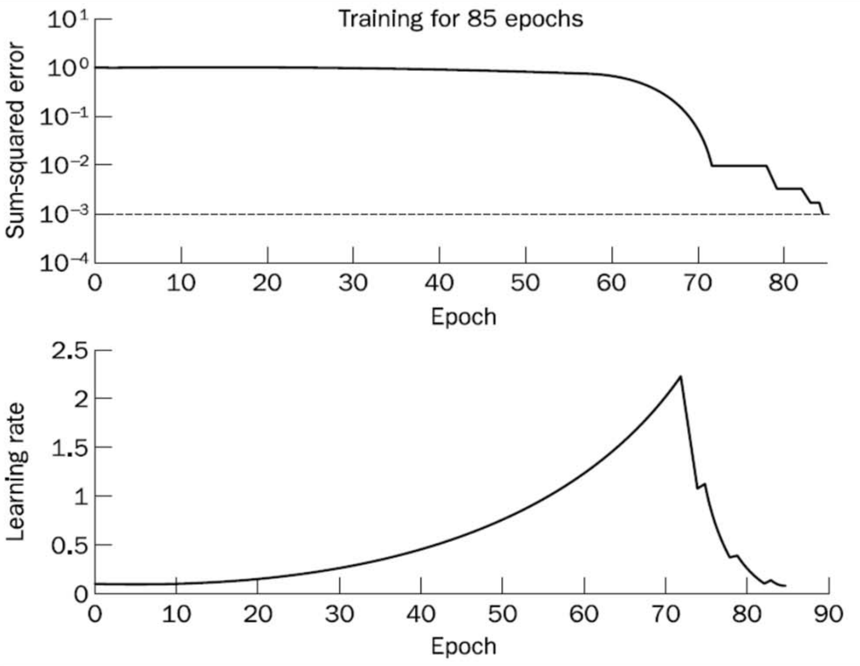
- 휴리스틱을 이용한 수렴 가속
  - 수렴을 가속시키면서도 불안정한 상태를 피할 수 있음
  - 휴리스틱을 적용 가능
    - 휴리스틱1:
      - 오차 제곱의 합의 변화량이 몇 번의 연속적인 에폭에 대해 계속 같은 부호로 나타나면 학습률 매개변수 α값을 증가
    - 휴리스틱2:
      - 오차 제곱의 합의 변화량의 부호가 몇 번의 연속적인 에폭에 대해 계속 엇갈리면 학습률 매개변수 α값을 감소

다층 신경망에서의 가속 학습

- 적응 학습률
  - 초기 학습률에서, 현재 에폭에서 미리 정의된 비율 이상으로 오차 제곱의 합이 이전 값을 초과하면 학습률 매개변수 감소 (보통 **0.7**곱함)
  - 초기 학습률에서, 현재 에폭에서 미리 정의된 비율 이상으로 오차 제곱의 합이 이전 값보다 작으면 학습률 매개변수 증가 (보통 **1.05**곱함)



적응학습률 이용한 학습



운동량과 적응학습률 이용한 학습

배운 것들을 적용했는데 잘 안되는 원인은?!

덜하거나
Underfitting 학습이 잘 안돼!

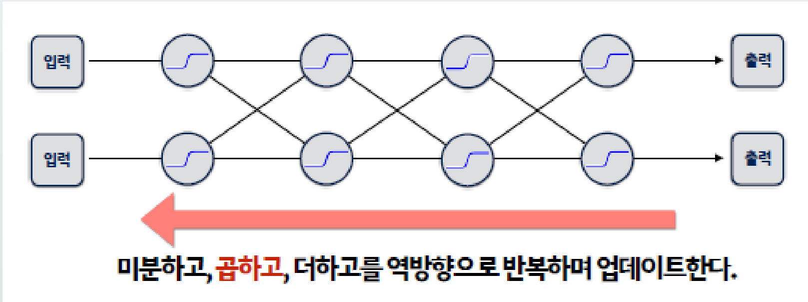
느리거나
Slow 학습은 언제 끝나는건가?

과하거나
Overfitting 학습 되어도 분류가 잘 안된다?!

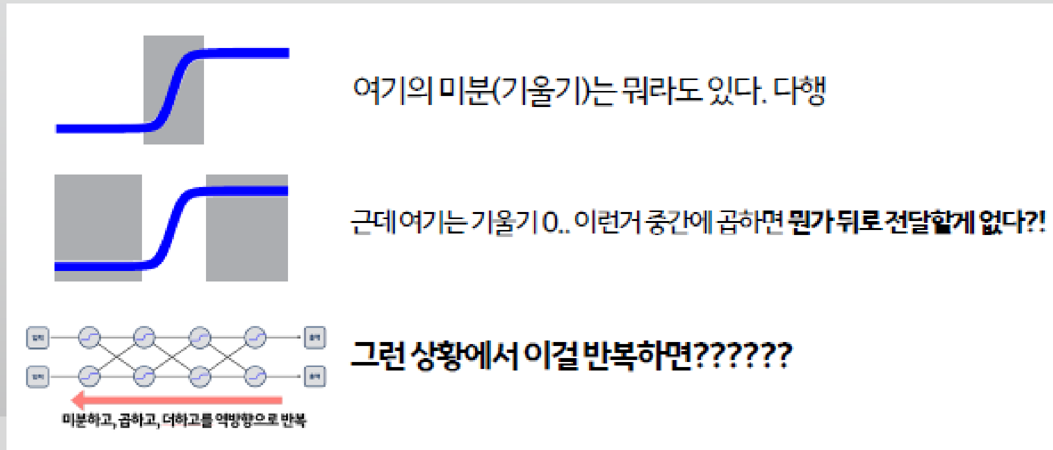
인공신경망의 어려움 – Underfitting

❖ Underfitting

- **Back propagation :** 현재 틀린 정도를 '미분'을 이용하면서 역방향으로 전달하고 가중치를 갱신

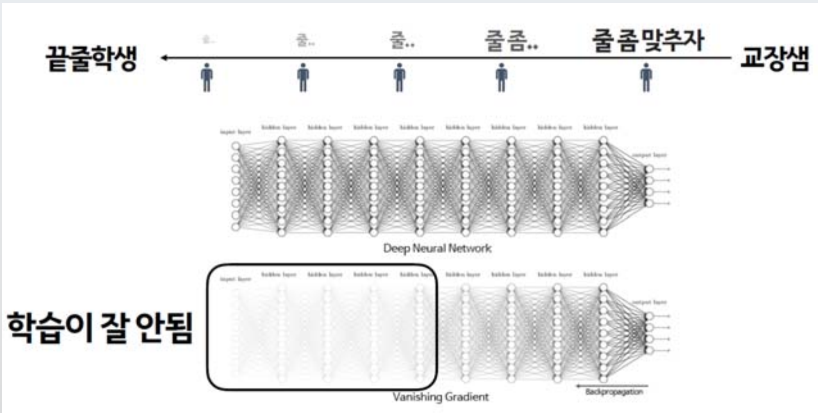


- **Activation** 함수로서 **sigmoid**를 사용할 때 문제 발생

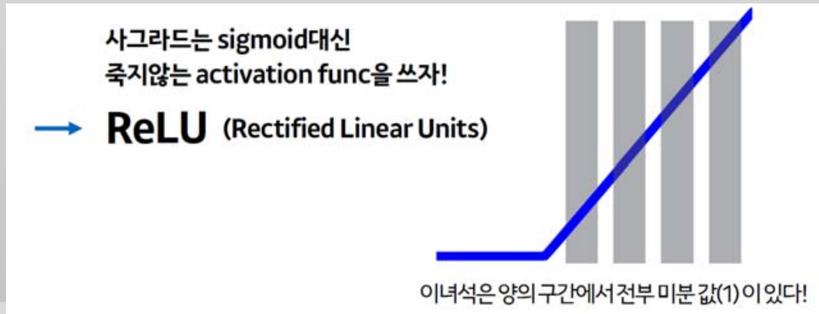


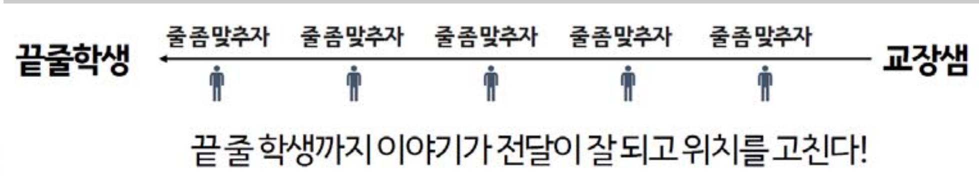
인공신경망의 어려움 – Underfitting

- Gradient Vanishing
  - Back propagation 과정에서, 레이어들을 지나는 동안 전달되는 값이 최초의 값보다 현저하게 작아짐
    - ➢ 값을 전달해도 의미 있는 변화가 일어나지 않음



  - 해결책





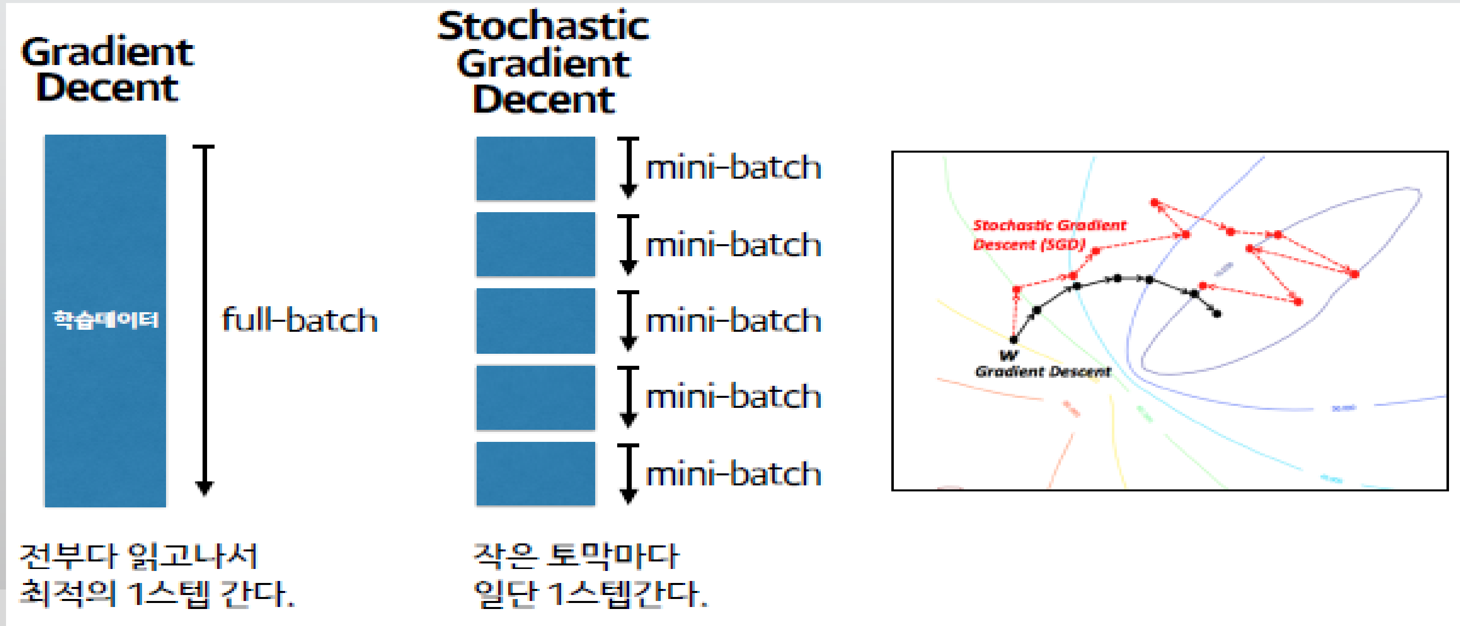
인공신경망의 어려움 – Slow

- Gradient Descent 사용시, 전체 에러를 계산하기 위해서 전체 데이터를 넣을 경우,
  - Weight를 한번 갱신하기 위해서 가지고 있는 큰 데이터를 모두 넣기 때문에 학습이 느려짐

이를 해결할 수 있는 방법은?!

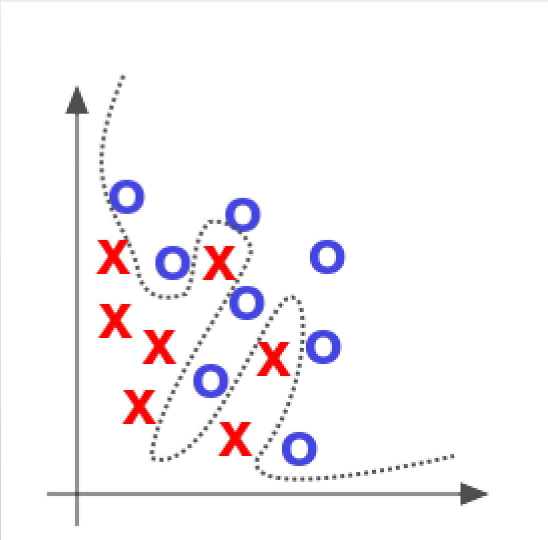
Stochastic Gradient Descent!

- 해결책 SGD
  - 느린 완벽보다 조금만 훑어보고 일단 빨리 가보는 컨셉



인공신경망의 어려움 - Overfitting

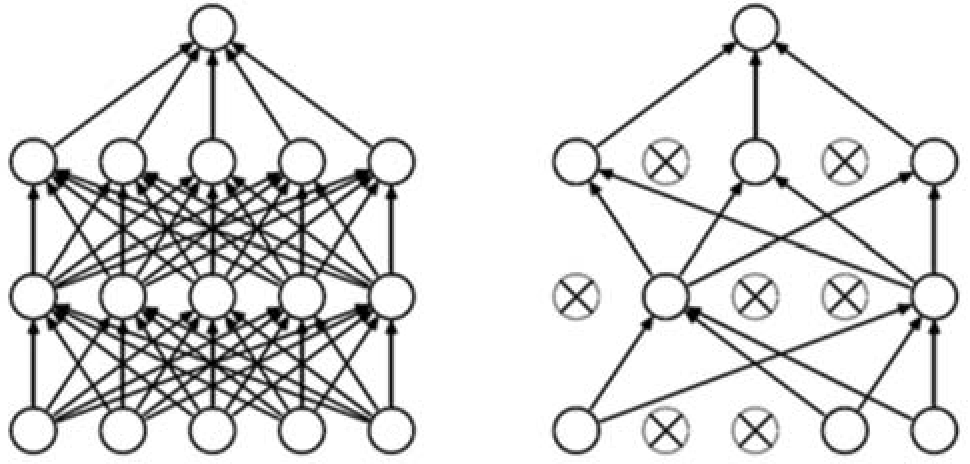
- 현재의 학습 데이터에 대해서 너무 과도하게 학습
  - 이 경우 우리가 가지고 있지 않은 데이터에 대해서는 잘 분류해내지 못하는 문제가 발생
  - 범용성 저하



Overfitting 상황

인공신경망의 어려움 - Overfitting

- 해결책 : Dropout
  - 학습을 일부 뉴런을 생략하고 수행하는 방법
  - 일정한 mini-batch 구간 동안 생략된 망에 대한 학습을 끝내면, 다시 무작위로 다른 뉴런들을 생략하면서 반복적으로 학습을 수행
  - 무작위로 망을 생략한 후 학습하므로, Voting에 의한 평균 효과를 얻을 수 있음


(a) Standard Neural Net (b) After applying dropout.