



4장. 딥러닝 시작

2팀 안예은, 진웨이안, 신예나

목차

#01 인공지능, 머신 러닝, 딥러닝 / 딥러닝 구조(1)

#02 딥러닝 구조(2)

#03 딥러닝 알고리즘 / 우리는 무엇을 배워야 할까?



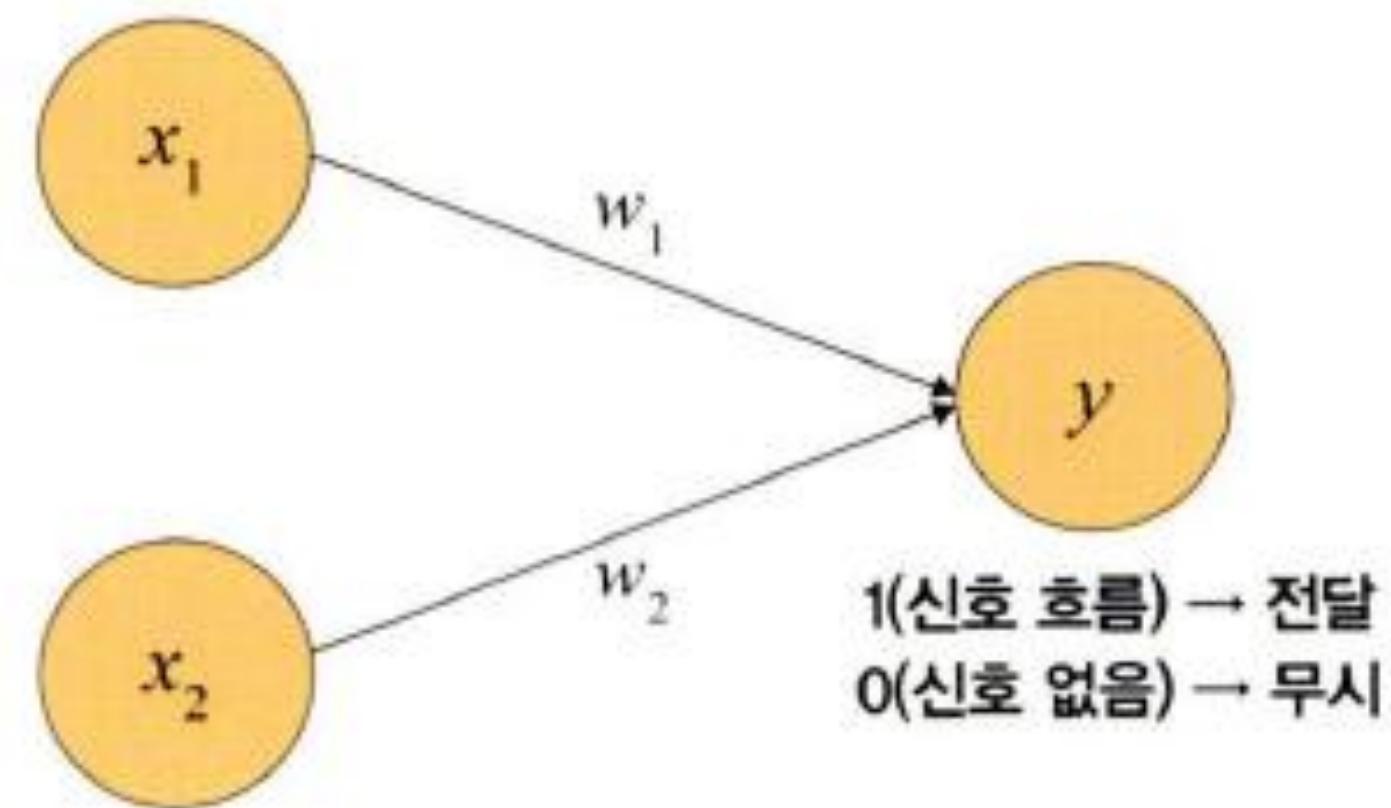
04-1 인공 신경망의 한계와 딥러닝 출현



#4-1 퍼셉트론

- 오늘날 딥러닝의 기원이 되는 알고리즘
- 입력신호를 받아 가중치를 곱해 출력
- 출력은 신호의 흐름(1)/없음(0)으로 표현 -> 선형 분류기

▼ 그림 4-1 퍼셉트론 원리



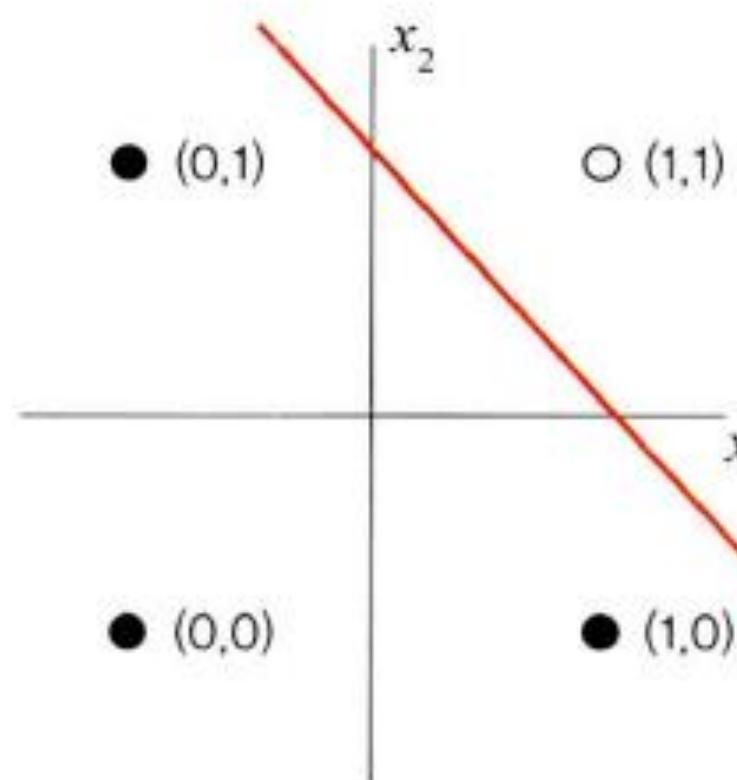
#4-1 AND 게이트

- 모든 입력이 1일 때만 1 출력
- 단층 퍼셉트론으로 구현 가능(선형적으로 구분 가능)

▼ 표 4-1 AND 게이트

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

▼ 그림 4-2 AND 게이트



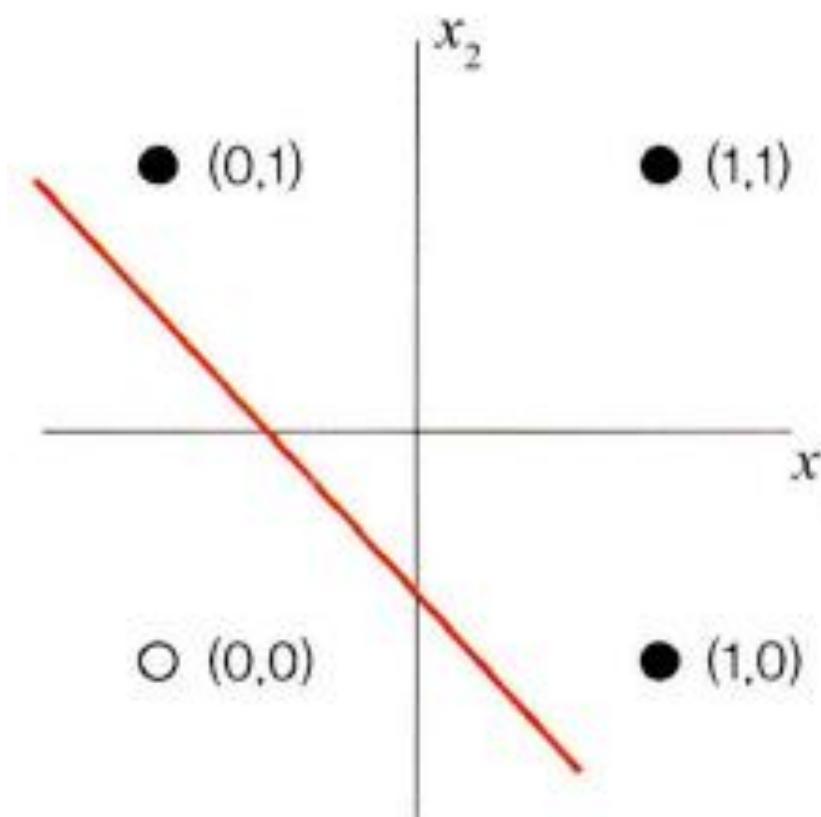
#4-1 OR 게이트

- 입력 값 중 하나라도 1이면 출력이 1
- 단층 퍼셉트론으로 구현 가능

▼ 표 4-2 OR 게이트

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

▼ 그림 4-3 OR 게이트



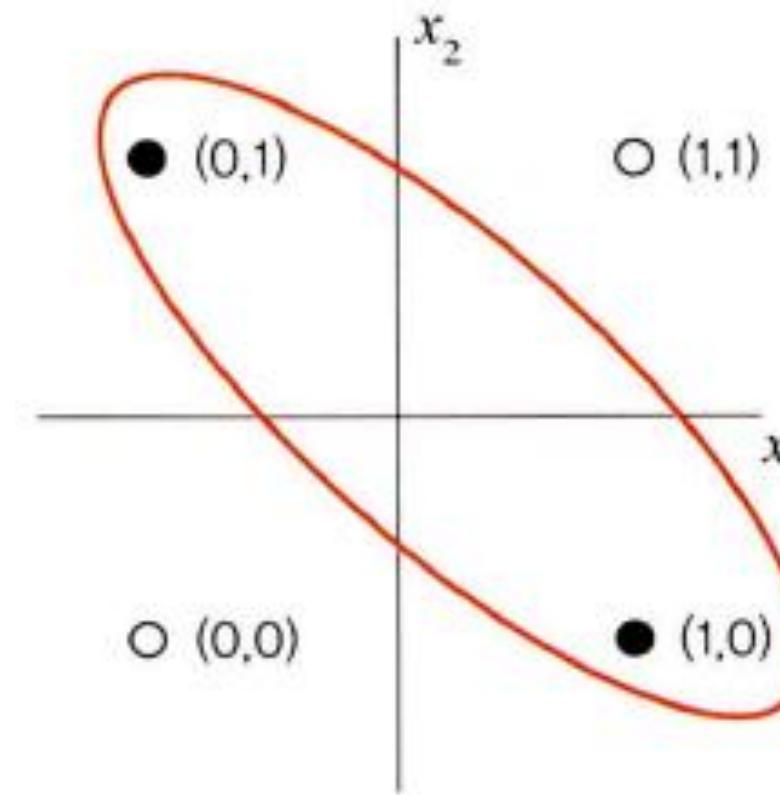
#4-1 XOR 문제 등장

- 입력이 다를 때만 1 출력
- 선형 분리 불가능 \rightarrow 단층 퍼셉트론으로 해결 불가
- 한계 원인: 선형 분류기 구조

▼ 표 4-3 XOR 게이트

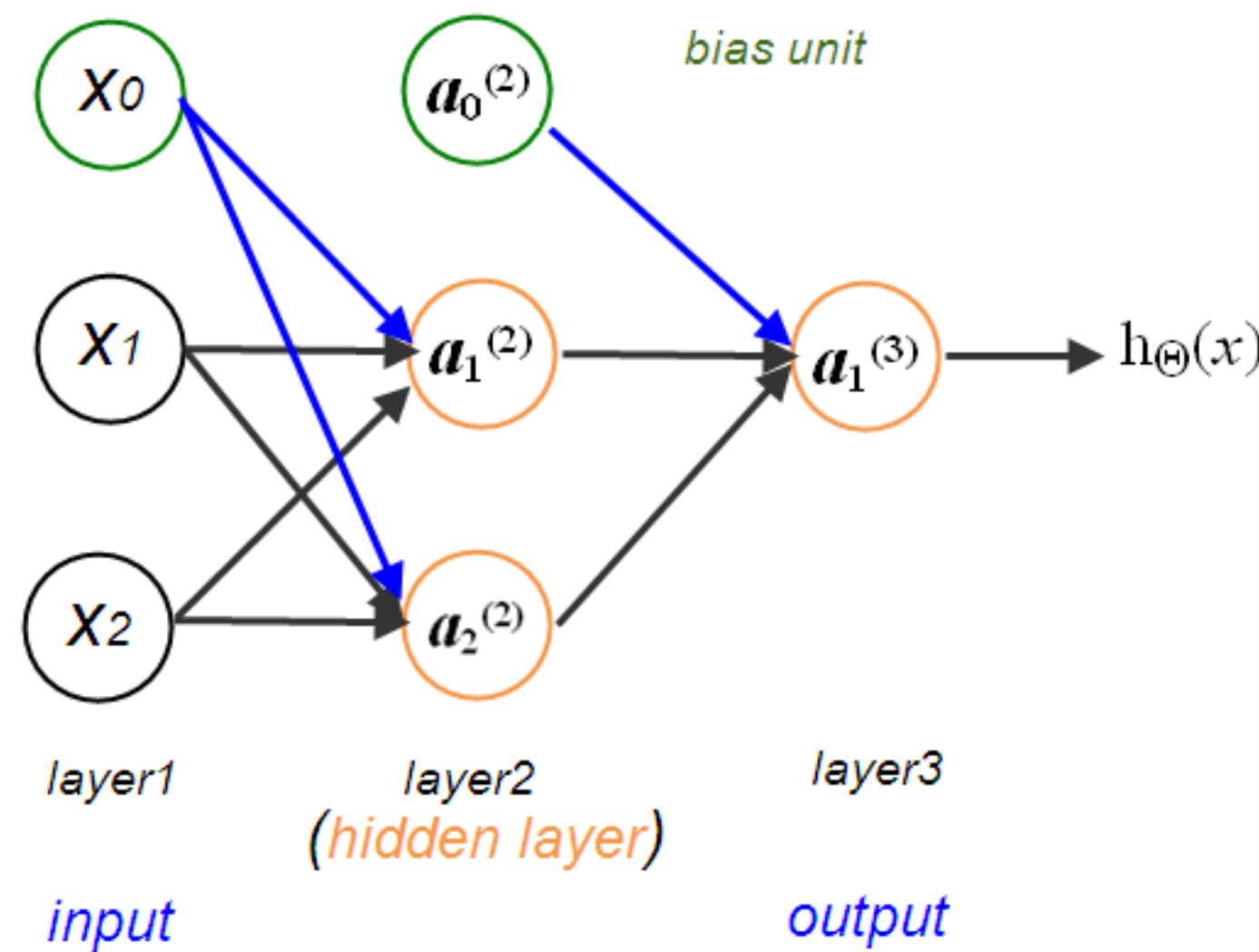
x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

▼ 그림 4-4 XOR 게이트



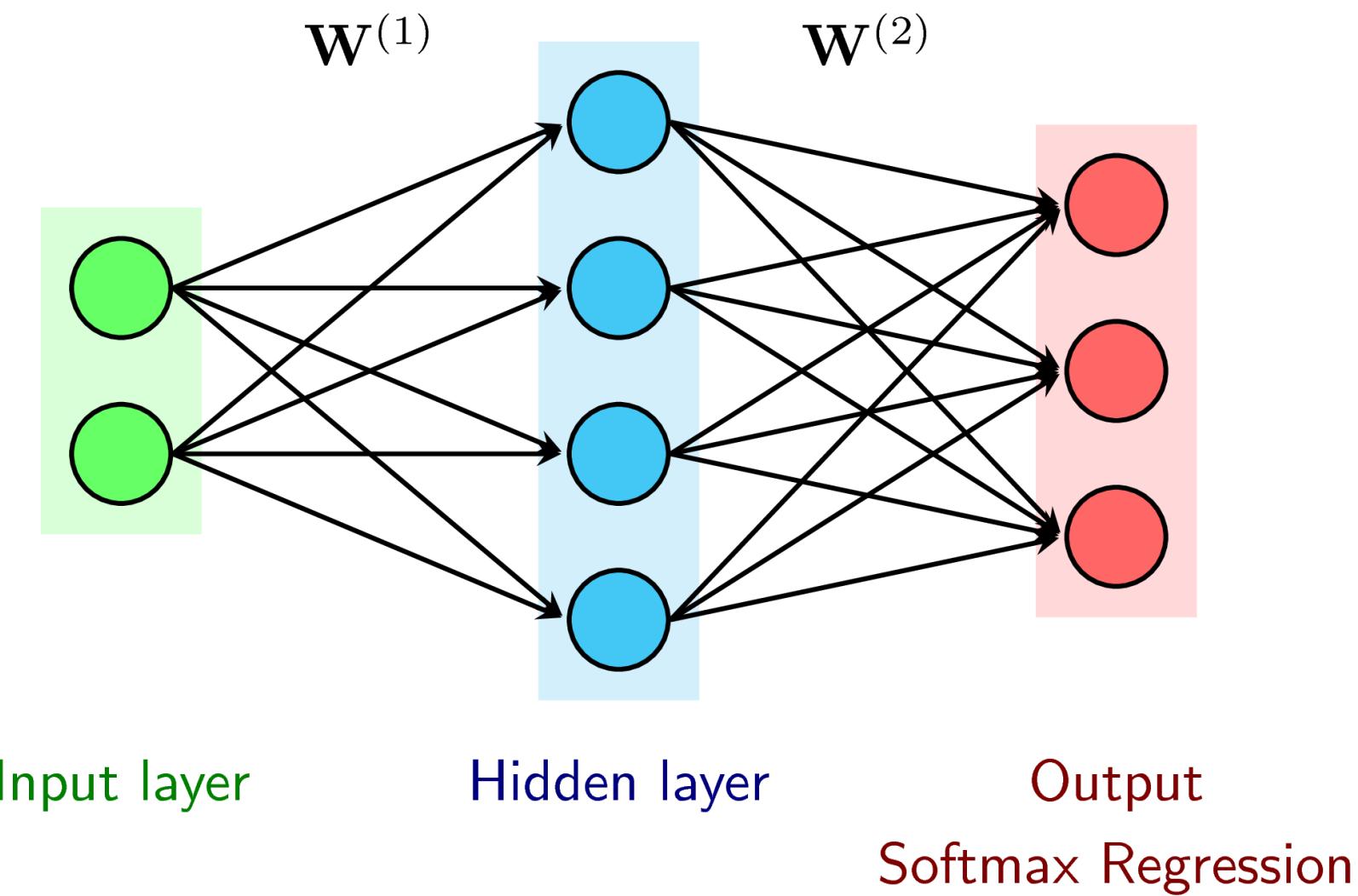
#4-1 다중 퍼셉트론(MLP)

- 은닉층 추가 -> 비선형 문제 해결
- AND/OR를 조합하여 XOR 표현 가능
- 핵심: 은닉층 도입으로 한계 극복



#4-1 심층 신경망(DNN)

- 은닉층이 여러 개인 구조 = Deep Neural Network
 - 다층 퍼셉트론의 발전된 형태
 - 오늘날의 딥러닝의 기반



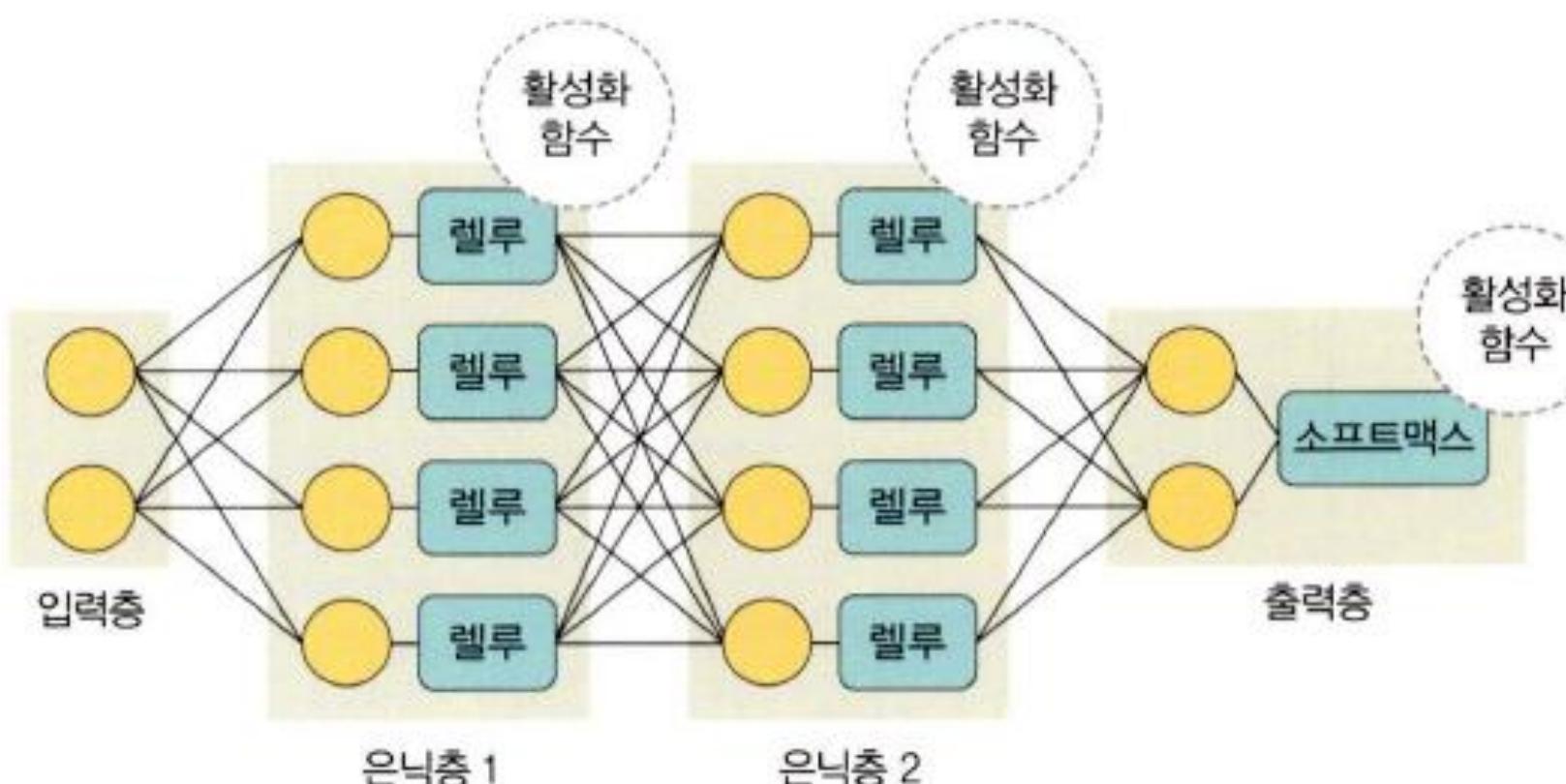
04-2 딥러닝 구조(1)



#4-2 신경망의 층 구조

- 입력층: 데이터를 받아들임
- 은닉층: 가중합 계산-> 활성화 함수 적용
- 출력층: 최종 결과 도출

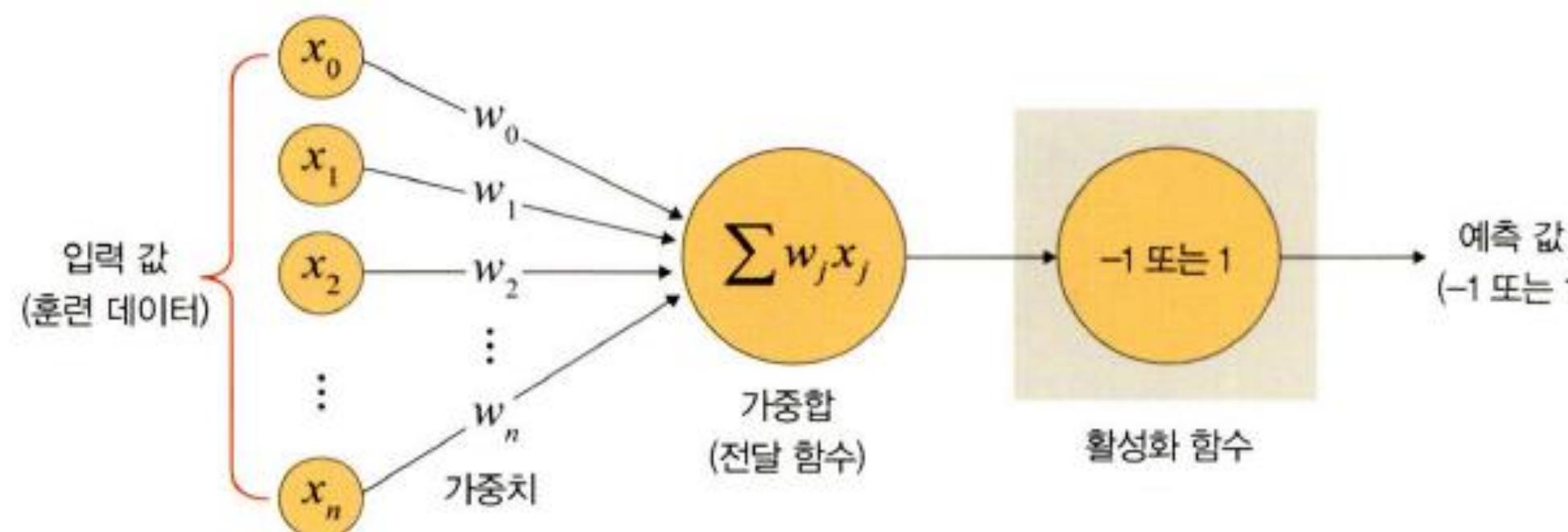
▼ 그림 4-5 딥러닝 구조



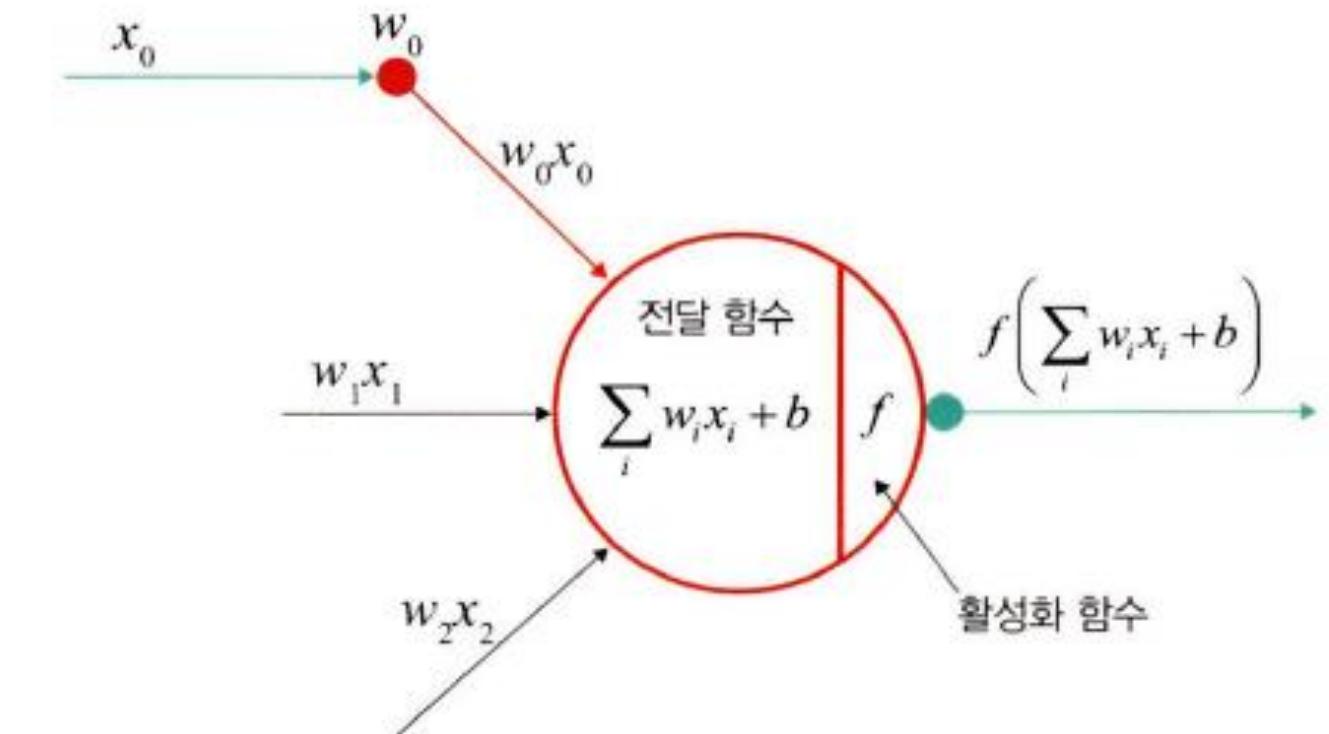
#4-2 가중치와 편향

- 가중치: 연결 강도, 입력이 결과에 미치는 영향 조절
- 편향: 기준선 조정, 출력 값의 유연성 확보
- 수식: $\sum(wx) + b$

▼ 그림 4-6 가중치



▼ 그림 4-7 전달 함수



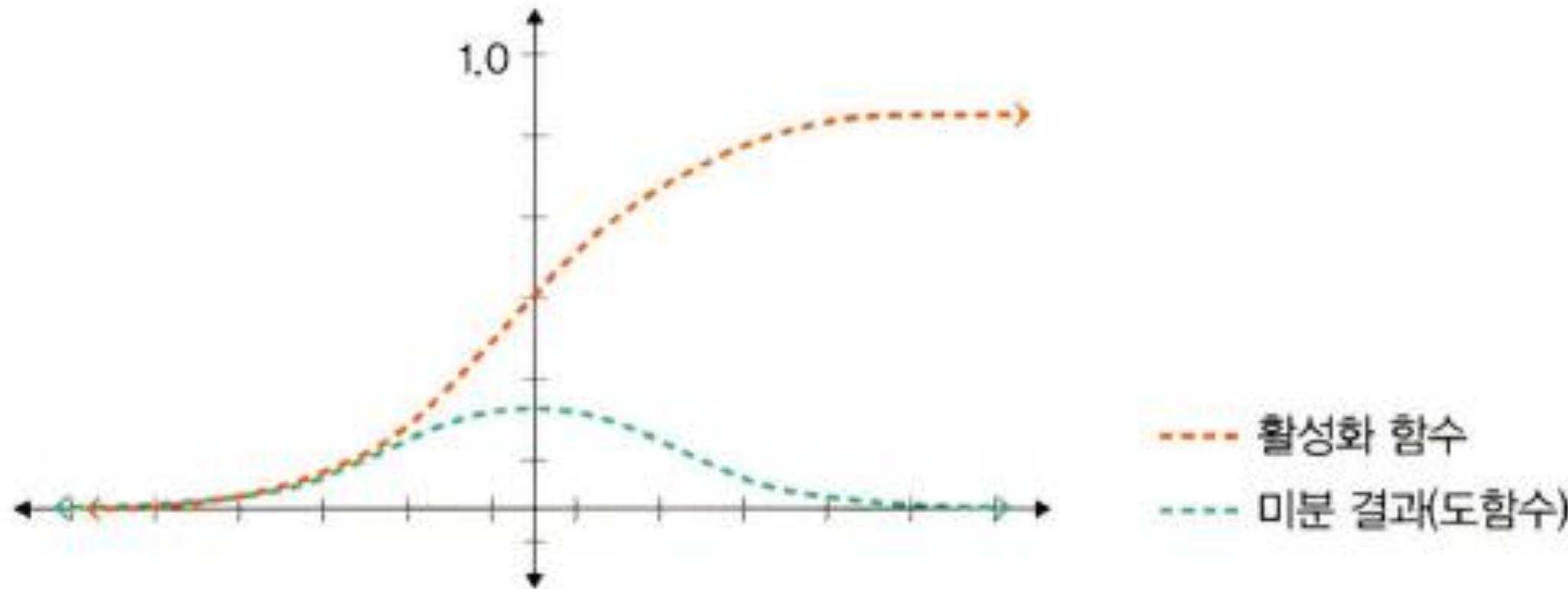
#4-2 활성화 함수

시그모이드 함수

- 출력: 0~1 범위
- 장점: 확률적 해석 가능
- 단점: 기울기 소멸 문제 발생 -> 딥러닝에는 잘 사용하지 않음

$$f(x) = \frac{1}{1 + e^{-x}}$$

▼ 그림 4-8 시그모이드 활성화 함수와 미분 결과

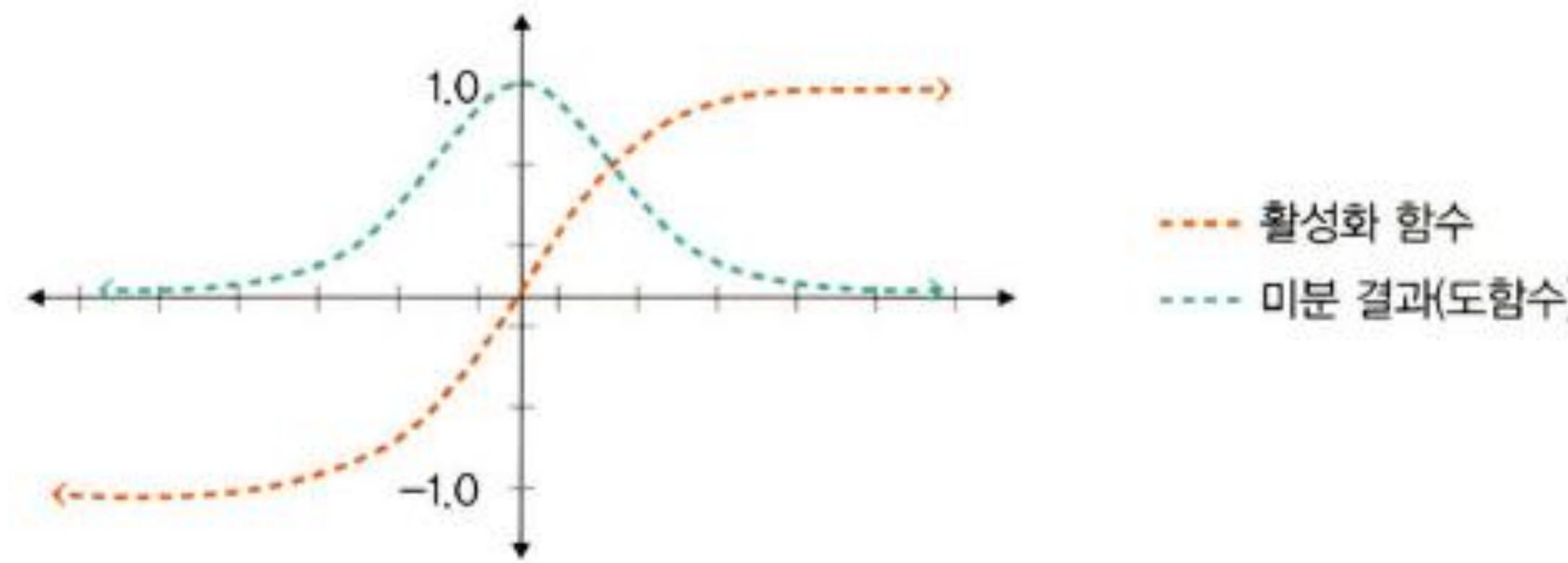


#4-2 활성화 함수

하이퍼볼릭 탄젠트 함수

- 출력: $-1 \sim 1$ 범위 \rightarrow 평균이 0이라 수렴 안정성 ↑
- 단점: 기울기 소멸 문제 여전히 존재

▼ 그림 4-9 하이퍼볼릭 탄젠트 활성화 함수와 미분 결과

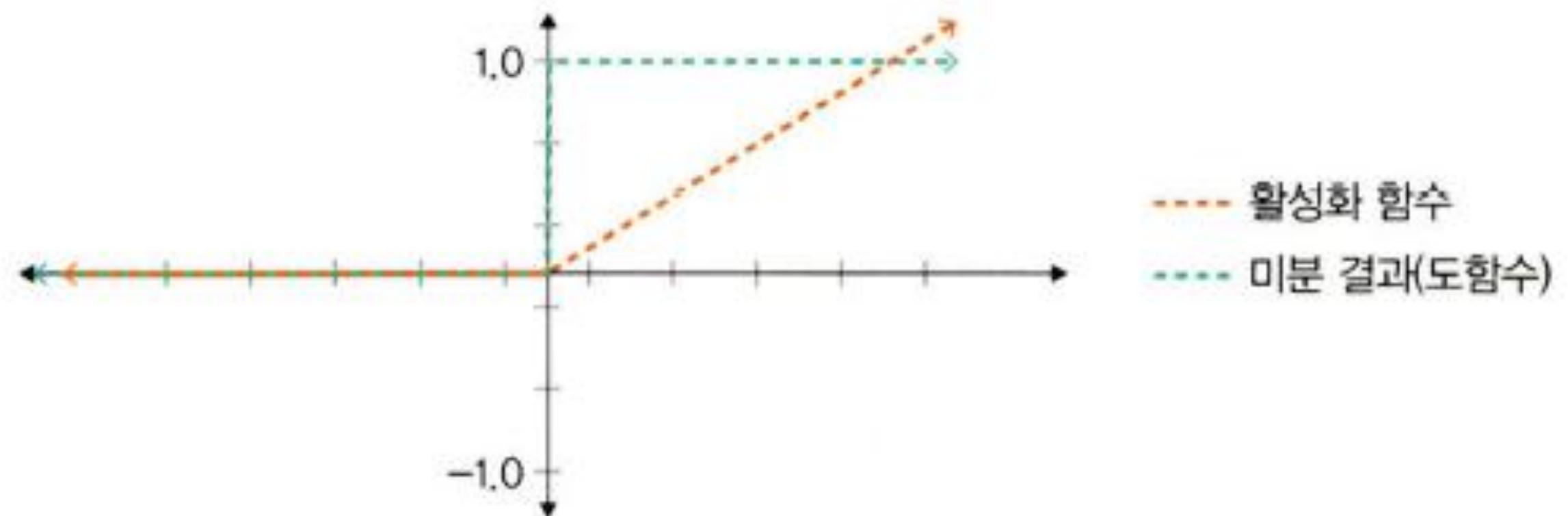


#4-2 활성화 함수

ReLU 함수

- $x > 0 \rightarrow x, x \leq 0 \rightarrow 0$
- 장점: 빠른 학습, 기울기 소멸 문제 완화
- 단점: 음수 입력 시 뉴런이 죽는 문제 발생

▼ 그림 4-10 렐루 활성화 함수와 미분 결과

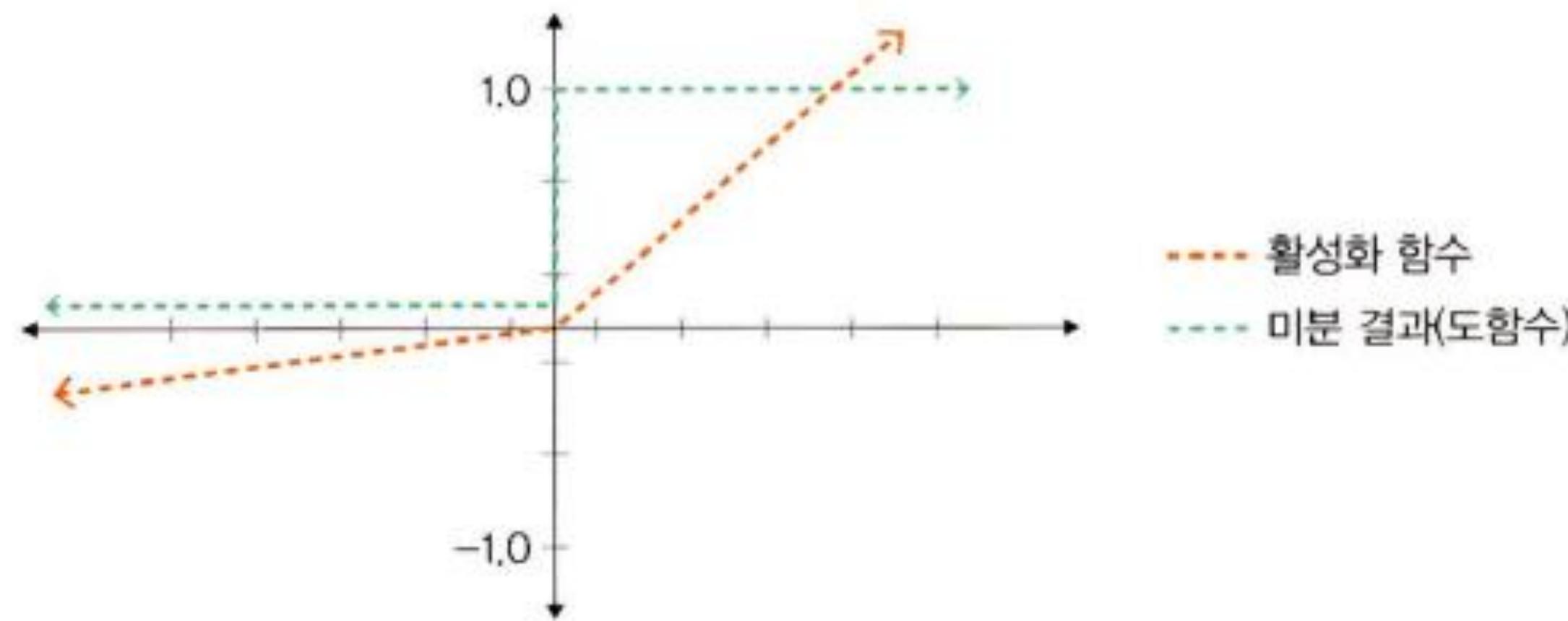


#4-2 활성화 함수

Leaky ReLU 함수

- 음수 입력일 때도 작은 기울기 부여 \rightarrow 죽은 뉴런 문제 완화

▼ 그림 4-11 리키 렐루 활성화 함수와 미분 결과



#4-2 활성화 함수

소프트 맥스 함수

- 다중 분류에서 출력층에 주로 사용
- 확률 분포로 변환(출력 합=1)

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^n \exp(a_i)}$$

#4-2 손실 함수

- 예측값 vs 실제값 차이 측정
- 학습의 기준이 되는 지표

평균 제곱 오차(MSE)

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

\hat{y}_i : 신경망의 출력(신경망이 추정한 값)
 y_i : 정답 레이블
 i : 데이터의 차원 개수

- 회귀 문제에 주로 사용
- 예측값과 실제값 차이를 제곱해 평균

크로스 엔트로피 오차(CEE)

$$\text{CrossEntropy} = - \sum_{i=1}^n y_i \log \hat{y}_i$$

\hat{y}_i : 신경망의 출력(신경망이 추정한 값)
 y_i : 정답 레이블
 i : 데이터의 차원 개수

- 분류 문제에 사용(원-핫 인코딩)
- 출력 확률과 실제 정답의 차이를 최소화

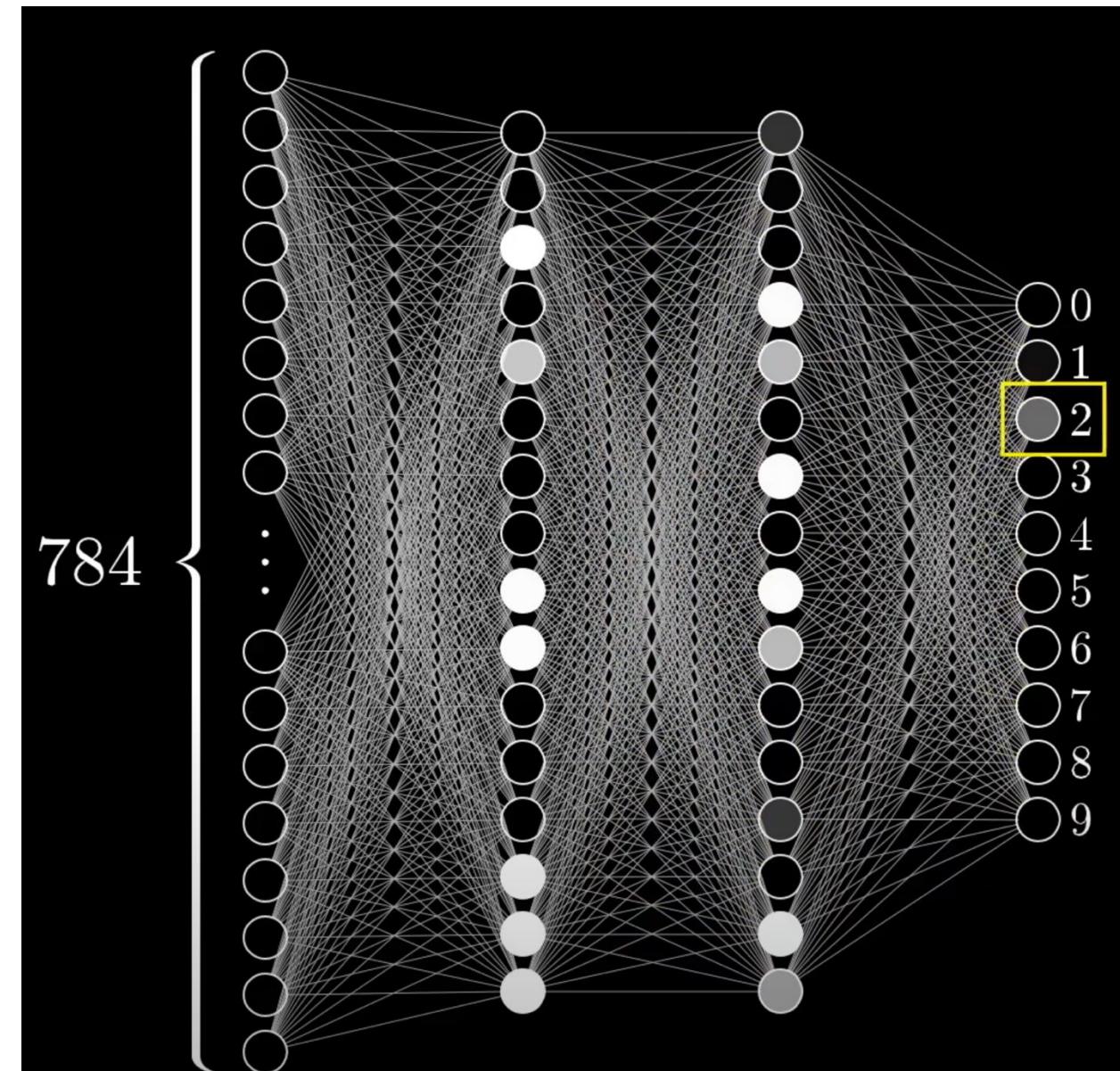
4-2 딥러닝 구조(2)



#4-2 딥러닝 학습

딥러닝 학습

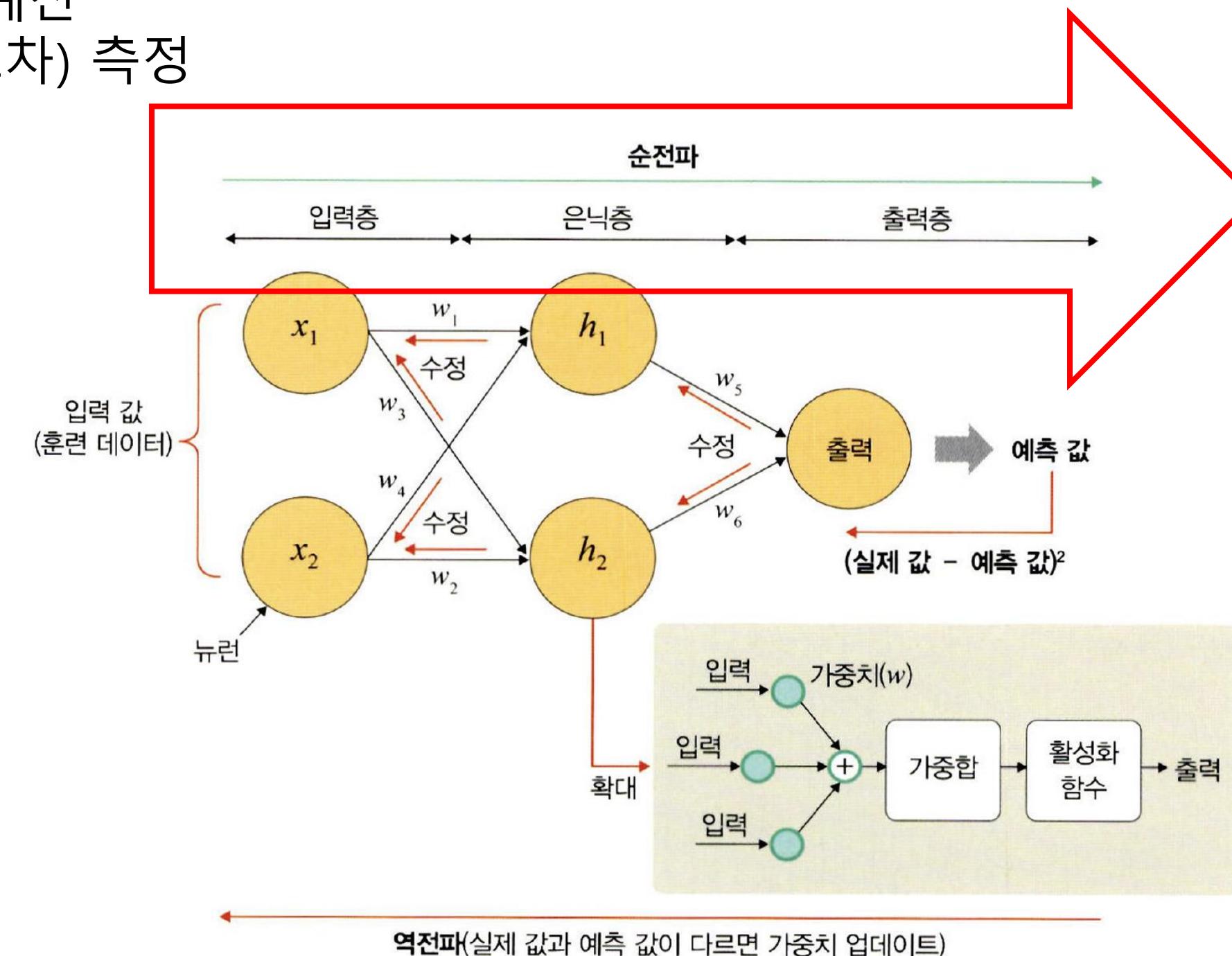
- 순전파(feedforward)와 역전파(backpropagation) 두 단계로 이루어짐
- 순전파: 입력 데이터를 신경망에 전달하여 예측 값 계산 →
- 역전파: 예측 값과 실제 값의 차이(손실, 오차)를 이용해 가중치 업데이트 ←



#4-2 딥러닝 학습

순전파 (Feedforward)

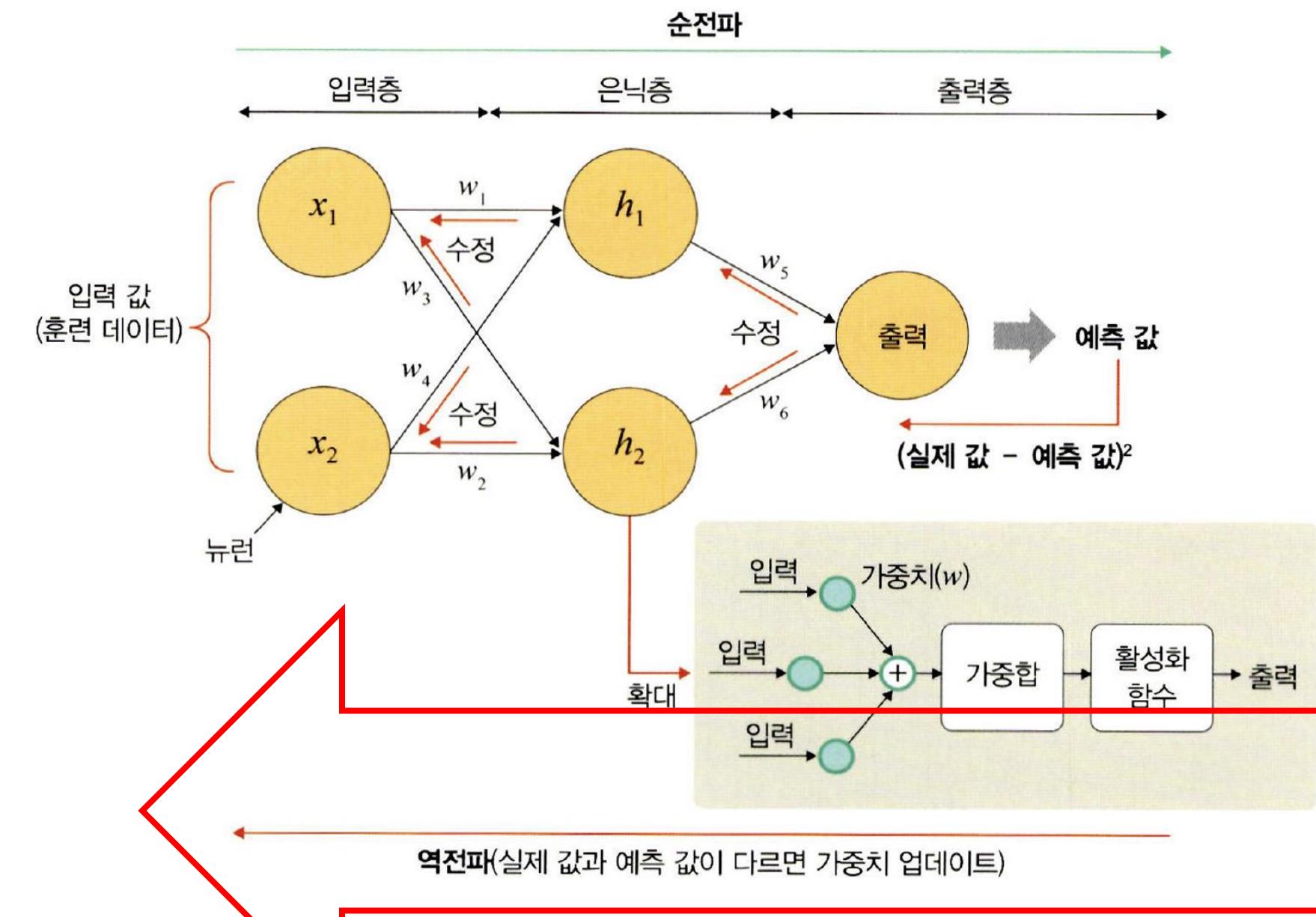
- 입력 데이터 \rightarrow 입력층 \rightarrow 은닉층(가중합 & 활성화 함수 적용) \rightarrow 출력층 전달
- 모든 뉴런을 통과하면서 최종 예측 값 계산
- 손실 함수: 실제 값과 예측 값의 차이(오차) 측정



#4-2 딥러닝 학습

역전파 (Backpropagation)

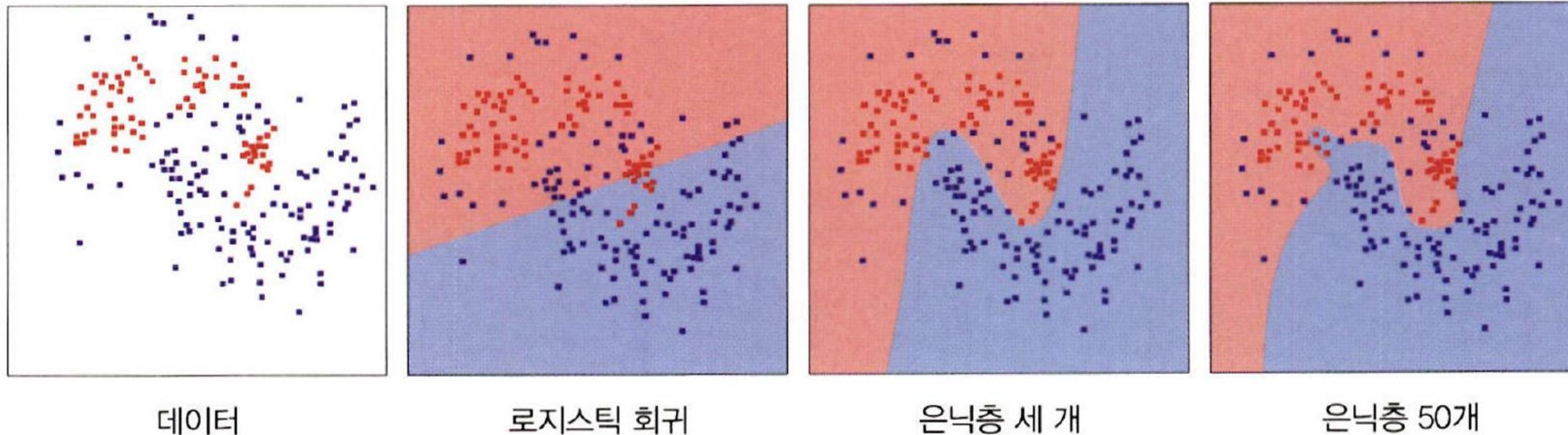
- 출력층에서 발생한 오차를 은닉층, 입력층으로 거꾸로 전달
- 각 뉴런의 기여도에 따라 가중치 조정 및 업데이트
- 반복 과정을 통해 모델이 점점 더 정확한 예측 수행



#4-2 딥러닝의 문제점과 해결 방안

딥러닝의 핵심

- 활성화 함수를 적용한 여러 은닉층을 결합해 비선형 영역을 표현하는 것
- 은닉층 개수가 많아질수록 데이터 분류 성능이 좋아짐

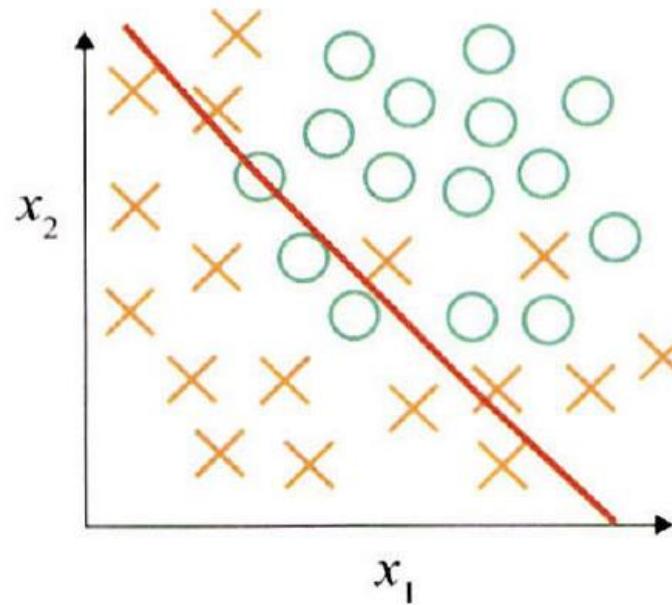


은닉층이 많아질수록 복잡한 문제를 해결할 수 있지만, 동시에 새로운 문제도 발생

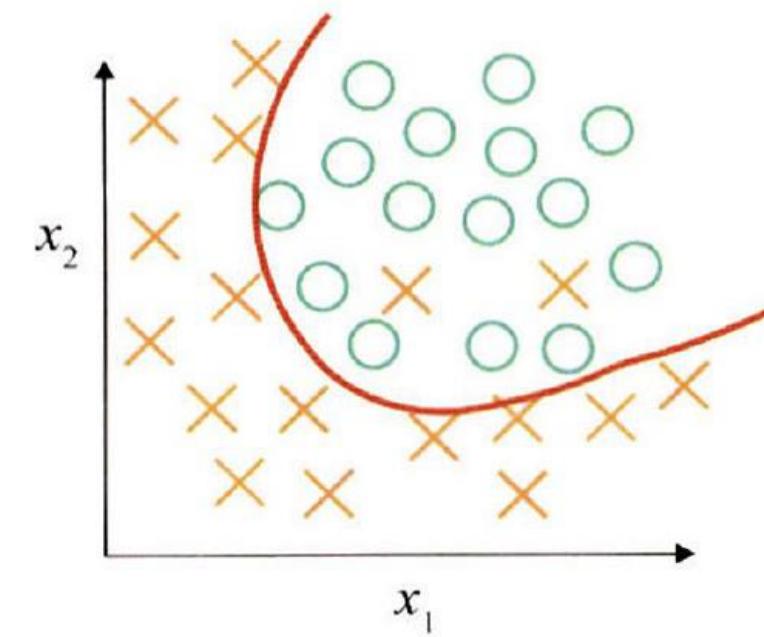
#4-2 딥러닝의 문제점과 해결 방안

과적합(Overfitting) 문제

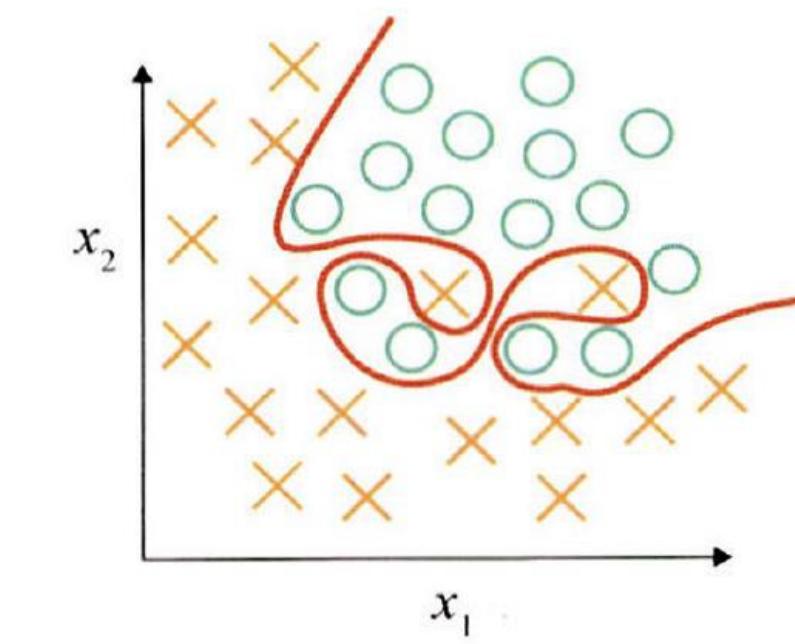
- 과적합: 훈련 데이터에 너무 과하게 맞춰 학습하는 현상
- 훈련 데이터 성능 ↑ → 검증/실제 데이터 성능 ↓
- 원인: 불필요하게 복잡한 모델, 너무 많은 파라미터
- 결과: 일반화 능력 저하



과소적합
(under-fitting)



적정적합
(generalized-fitting)



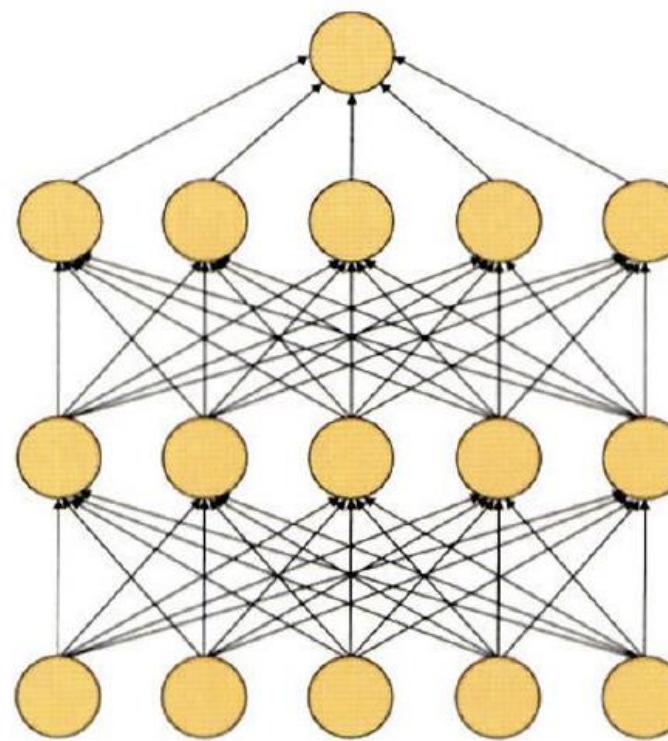
과적합
(over-fitting)

과적합 해결 방법 – 드롭아웃

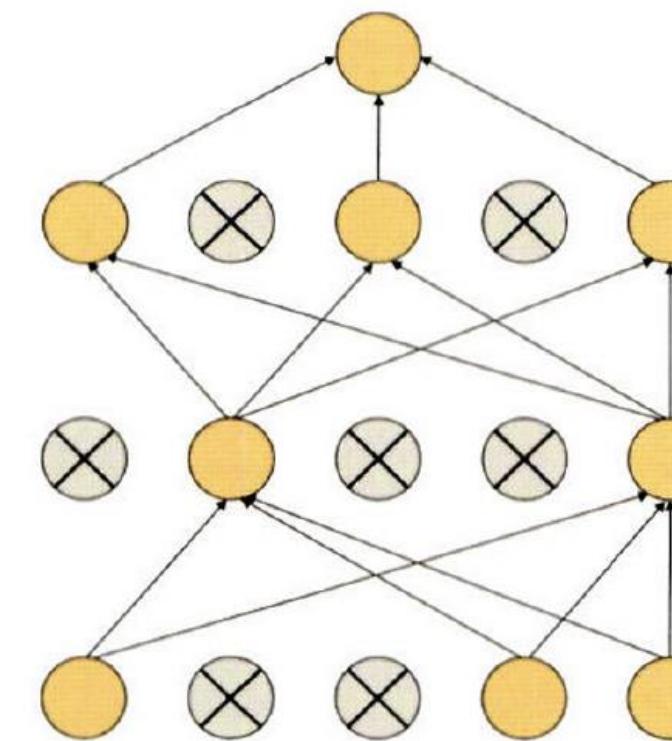
#4-2 딥러닝의 문제점과 해결 방안

드롭아웃(Dropout)

- 학습 중 일부 뉴런을 랜덤하게 제외
- 특정 뉴런/가중치에 의존하지 않도록 함
- 과적합 방지 → 일반화 성능 향상



일반적인 신경망



드롭아웃이 적용된 신경망

#4-2 딥러닝의 문제점과 해결 방안

드롭아웃 구현 (PyTorch)

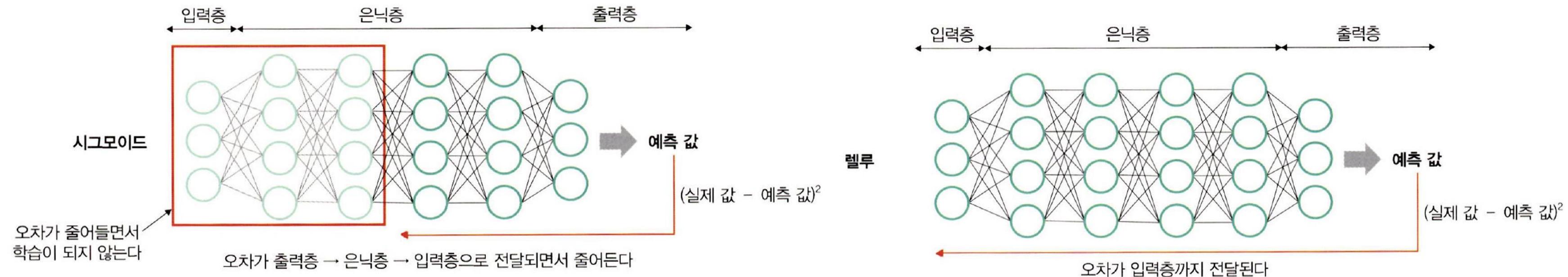
```
class DropoutModel(torch.nn.Module):
    def __init__(self):
        super(DropoutModel, self).__init__()
        self.layer1 = torch.nn.Linear(784, 1200)
        self.dropout1 = torch.nn.Dropout(0.5) ----- 50%의 노드를 무작위로 선택하여
        self.layer2 = torch.nn.Linear(1200, 1200)    사용하지 않겠다는 의미
        self.dropout2 = torch.nn.Dropout(0.5)
        self.layer3 = torch.nn.Linear(1200, 10)

    def forward(self, x):
        x = F.relu(self.layer1(x))
        x = self.dropout1(x)
        x = F.relu(self.layer2(x))
        x = self.dropout2(x)
        return self.layer3(x)
```

#4-2 딥러닝의 문제점과 해결 방안

기울기 소멸 문제 (Vanishing Gradient)

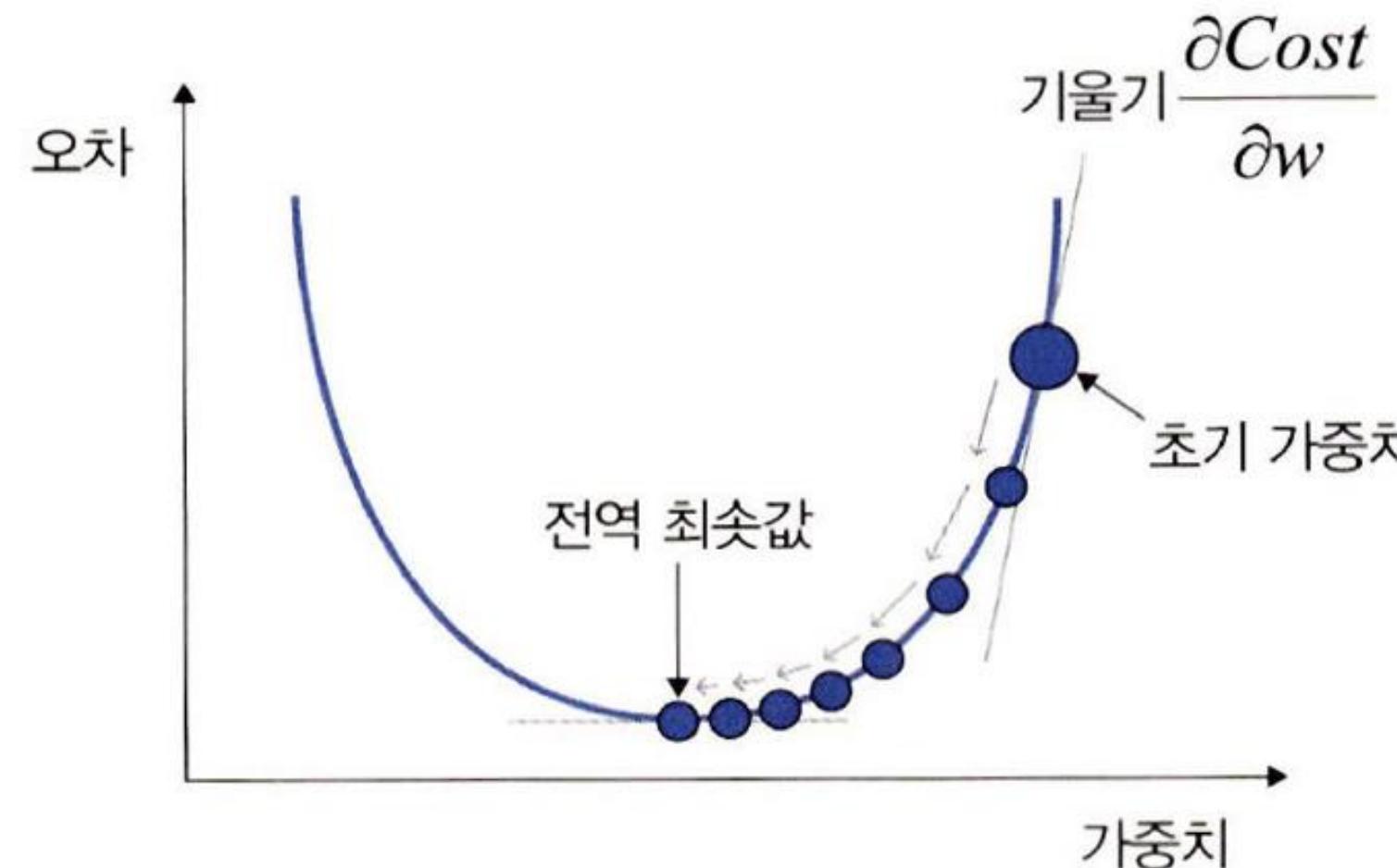
- 깊은 신경망에서 주로 발생하는 문제
- 역전파 과정에서 오차(gradient)가 은닉층을 거치며 점점 작아짐
- 결국 입력층 근처까지 전달되지 못해 학습이 멈추는 현상
- 시그모이드, 탄젠트 함수 사용 시 발생 확률
- 해결책: ReLU 같은 활성화 함수 사용



#4-2 딥러닝의 문제점과 해결 방안

성능이 나빠지는 문제 발생

- 경사 하강법: 손실 함수의 비용을 최소화하기 위해 기울기가 낮은 쪽으로 반복 이동
- 한계 : 경우에 따라 성능이 나빠지거나, 최적점에 도달하지 못하는 경우 발생

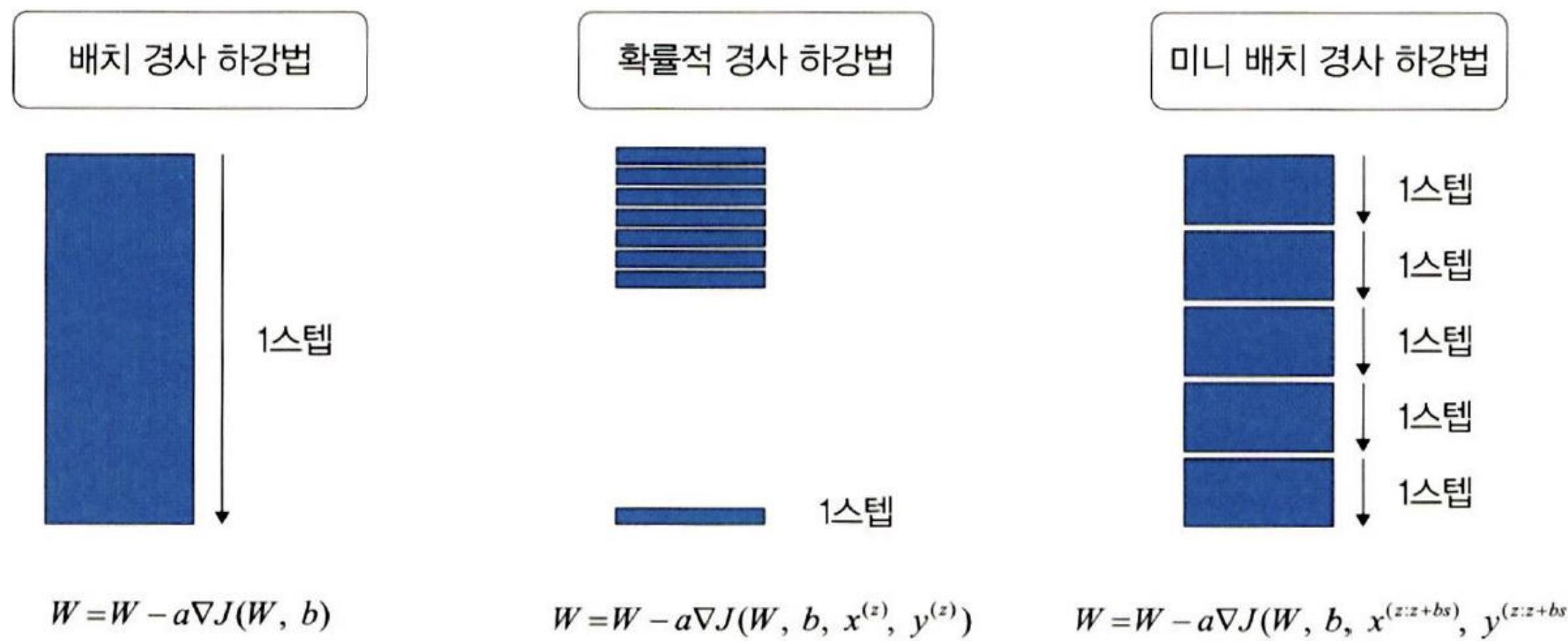


해결 방법 – 확률적 경사 하강법, 미니 배치 경사 하강법

#4-2 딥러닝의 문제점과 해결 방안

경사 하강법의 종류

- 배치 경사 하강법(BGD): 전체 데이터로 학습 → 안정적이지만 느림
- 확률적 경사 하강법(SGD): 데이터 1개씩 학습 → 빠르지만 불안정
- 미니 배치 경사 하강법: 일부 데이터(batch) 단위 학습 → 속도와 안정성 균형



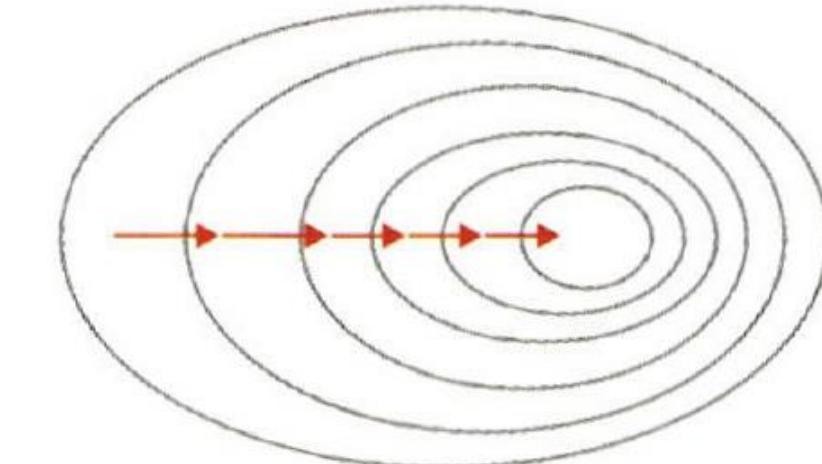
해결 방법 – 확률적 경사 하강법, 미니 배치 경사 하강법

#4-2 딥러닝의 문제점과 해결 방안

배치 경사 하강법 (Batch Gradient Descent, BGD)

- 전체 데이터셋을 이용해 기울기를 계산하고 한 번에 가중치를 업데이트
- 안정적이고 정확하지만 속도가 느림
- 수식:

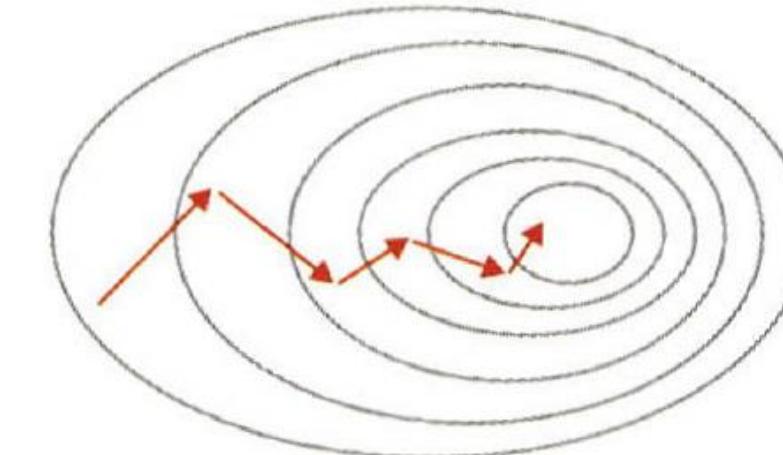
$$W = W - \alpha \nabla J(W, b)$$



배치 경사 하강법

확률적 경사 하강법 (Stochastic Gradient Descent, SGD)

- 데이터셋에서 무작위로 선택한 하나의 샘플을 사용해 기울기를 계산하고 업데이트
- 속도는 빠르지만 정확도는 낮을 수 있음

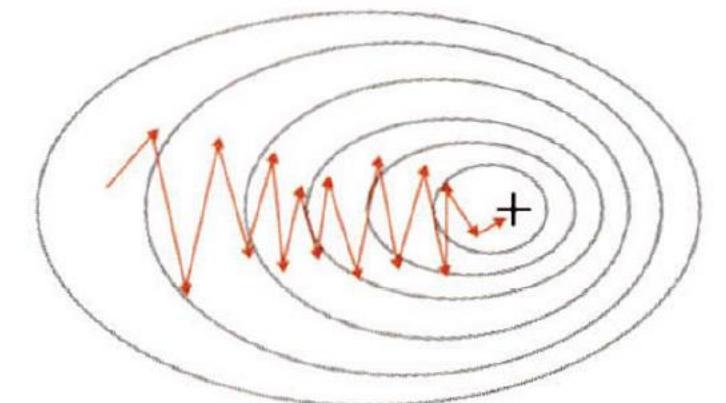
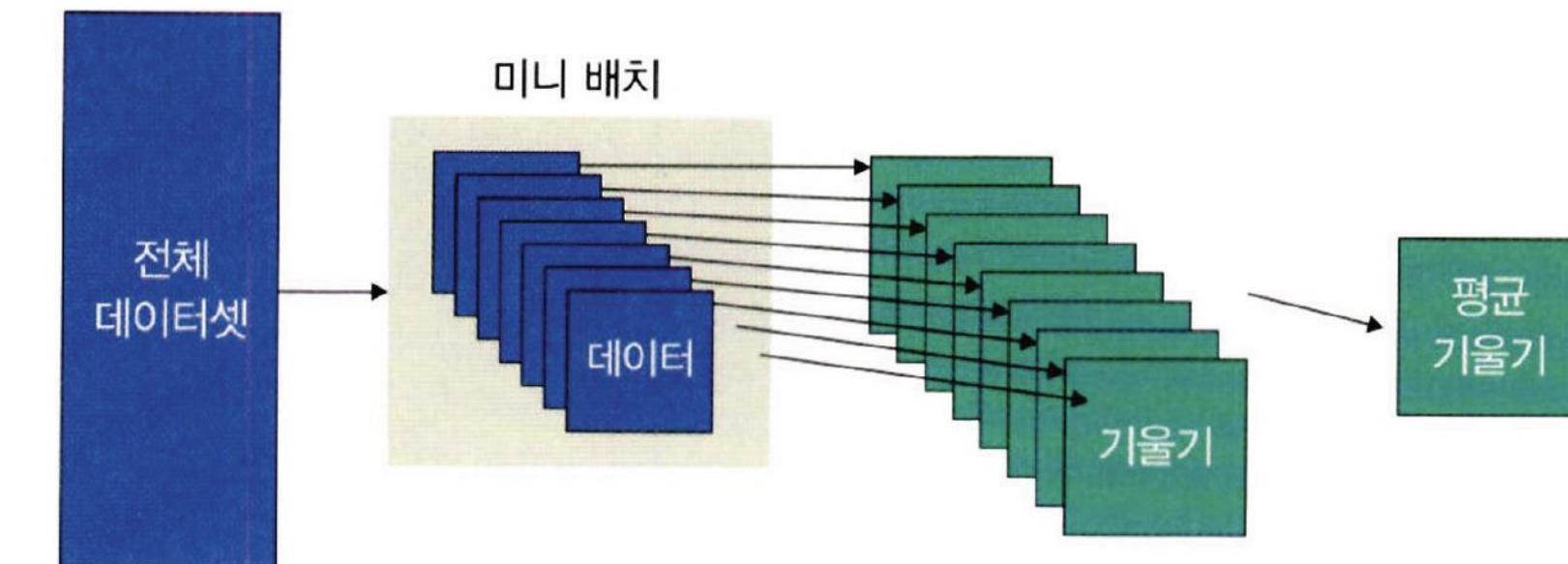


확률적 경사 하강법

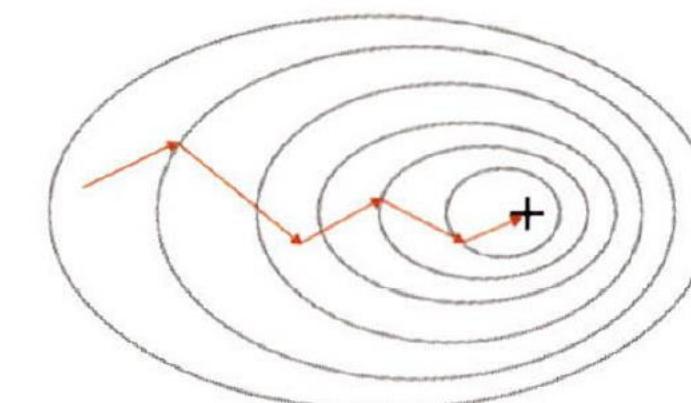
#4-2 딥러닝의 문제점과 해결 방안

미니 배치 경사 하강법 (Mini-Batch Gradient Descent)

- 전체 데이터셋을 여러 개의 미니 배치로 나눠 학습
- 각 미니 배치에서 기울기를 구하고, 평균 기울기를 이용해 모델 업데이트
- 장점:
 - 배치 경사 하강법보다 빠름
 - 확률적 경사 하강법보다 안정적
 - 실제로 가장 많이 사용되는 방식



확률적 경사 하강법



미니 배치 경사 하강법

#4-2 딥러닝의 문제점과 해결 방안

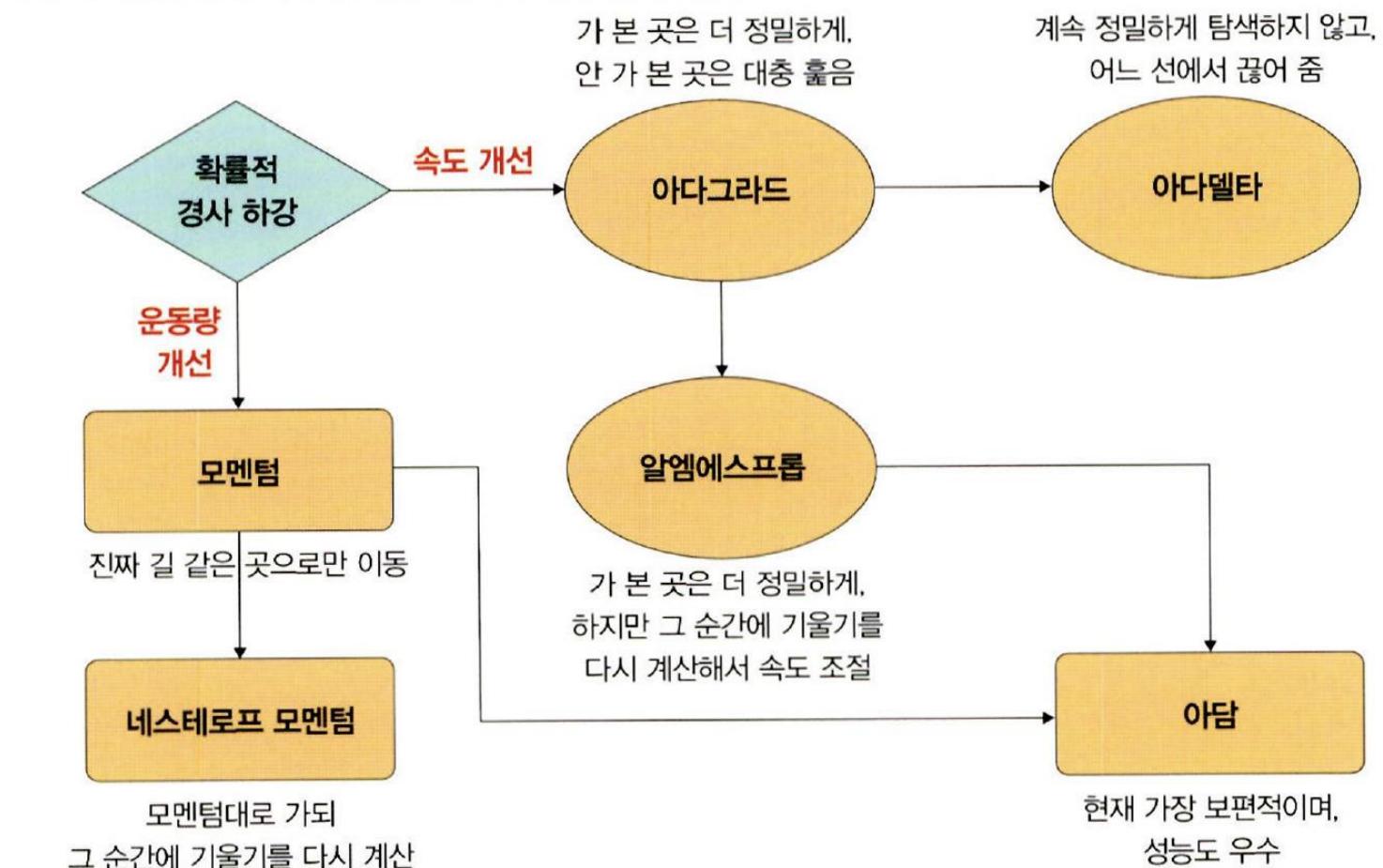
미니 배치 구현 (PyTorch)

```
class CustomDataset(Dataset):
    def __init__(self):
        self.x_data = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
        self.y_data = [[12], [18], [11]]
    def __len__(self):
        return len(self.x_data)
    def __getitem__(self, idx):
        x = torch.FloatTensor(self.x_data[idx])
        y = torch.FloatTensor(self.y_data[idx])
        return x, y
dataset = CustomDataset()
dataloader = DataLoader(
    dataset, ..... 데이터셋
    batch_size=2, ..... 미니 배치 크기로 2의 제곱수를 사용하겠다는 의미입니다.
    shuffle=True, ..... 데이터를 불러올 때마다 랜덤으로 섞어서 가져옵니다.
)
```

#4-2 딥러닝의 문제점과 해결 방안

옵티마이저(Optimizer)

- 경사 하강법의 한계를 보완하기 위해 속도(learning rate) 와 운동량(momentum) 조절
- 다양한 옵티마이저는 학습 속도, 안정성, 성능을 향상시킴
- 대표 종류:
 - Adagrad
 - Adadelta
 - RMSProp
 - Momentum / Nesterov Momentum
 - Adam



#4-2 딥러닝의 문제점과 해결 방안

주요 옵티마이저 특징

- Adagrad: 자주 업데이트 되는 변수 → 작은 학습률, 드문 변수 → 큰 학습률
- Adadelta: Adagrad 단점 보완, 학습률 조정 불필요
- RMSProp: 기울기 폭발 문제 방지, 안정적 학습
- Momentum: 이전 기울기 방향을 고려해 더 빠른 수렴
- Nesterov Momentum: 미래 위치를 고려한 개선된 Momentum
- Adam: Momentum + RMSProp 장점 결합, 가장 많이 쓰임

```
import torch.optim as optim

# Adagrad
optimizer = optim.Adagrad(model.parameters(), lr=0.01)

# RMSProp
optimizer = optim.RMSprop(model.parameters(), lr=0.001)

# SGD + Momentum
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)

# Adam
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

Optimizer 구현 (PyTorch)

#4-2 딥러닝을 사용할 때 이점

특성 추출

- 기존 **머신러닝** : 데이터에서 직접 특징(feature)을 뽑아야 했음
 - SVM, 나이브 베이즈, 로지스틱 회귀 등은 전문가 지식 필요
- **딥러닝** : 자동으로 특성 추출 수행
 - 은닉층을 깊게 쌓아 데이터 특성을 잘 학습
 - 복잡한 데이터(제조, 의료 등)에서도 효과적

빅데이터 활용

- 데이터가 많을수록 성능 향상
- 딥러닝의 자동 특성 추출 기능은 많은 사례 데이터와 결합할 때 더 강력
- 데이터가 부족하면 성능 향상이 어려움 → 이런 경우엔 머신러닝 고려 필요

03-1 딥러닝 알고리즘



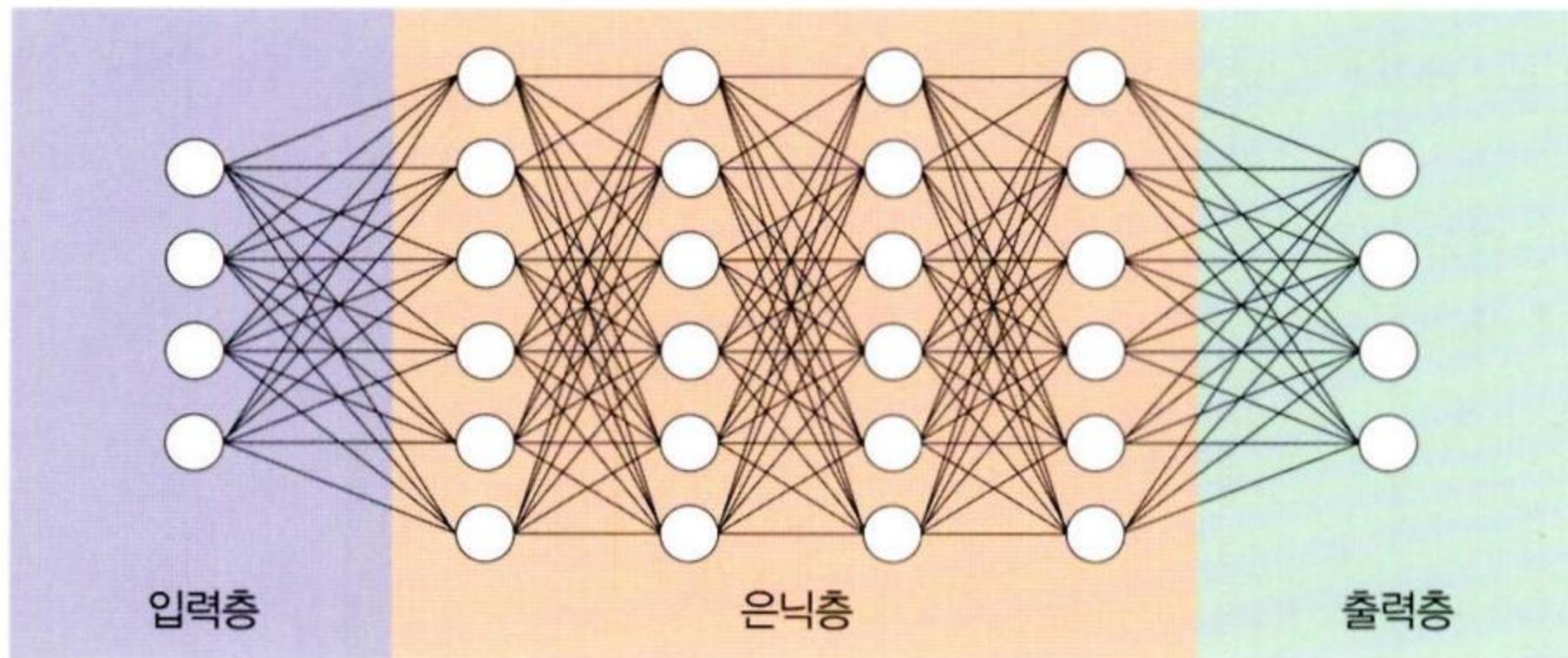
#03-1 심층 신경망 (DNN)

입력층과 출력층 사이에 다수의 은닉층을 포함하는
인공 신경망.

장. 별도의 트릭 없이 다수의 은닉층을 활용해 비선형
분류 가능

▼ 그림 4-24 심층 신경망

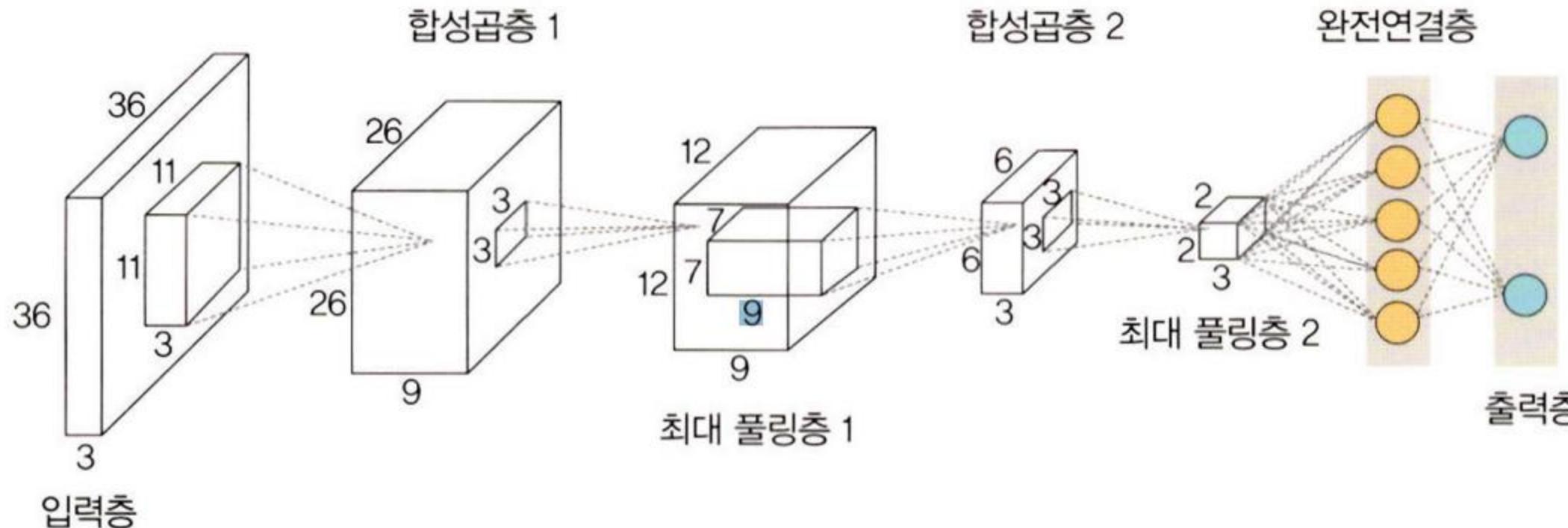
발생



#03-1 합성곱 신경망 (CNN)

- > 특징
- 각 층 입출력 형상 유지
- 이미지 공간 정보 유지하면서 인접 이미지 차이 인식
- 복수 필터로 이미지 특징 추출 및 학습
- 추출한 이미지 특징 모으고 강화하는 풀링층
- 공유 파라미터로 필터 사용, 학습 파라미터 적음

=> 합성곱층과 풀링층을 포함하는, 이미지 처리 성능이 좋은 인공 신경망 알고리즘

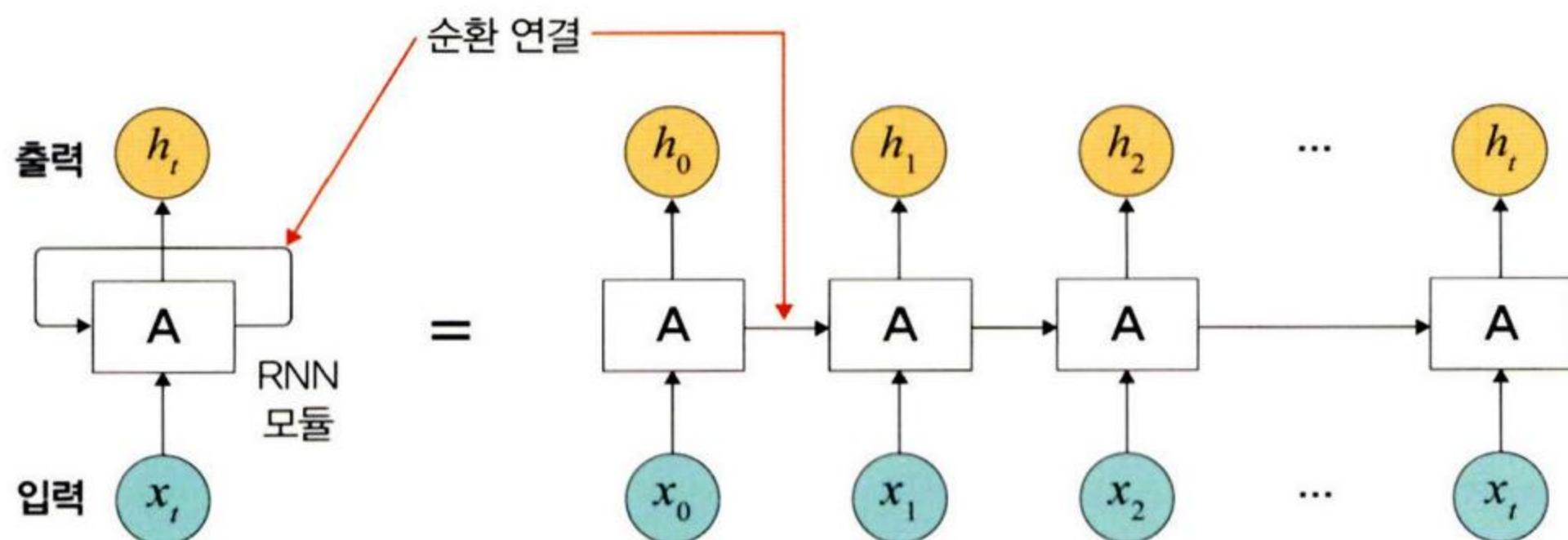


#03-1 순환 신경망 (RNN)

- > 특징
 - 시간성 정보를 활용하여 데이터 특징을 다룸
 - 사용되는 데이터가 동적이고, 가변적
- 단. 기울기 소멸 문제 발생 -> 메모리 개념을 도입한 LSTM이 사용되는 중

=> 시간 흐름에 따라 변화하는 데이터를 학습하기 위한 인공 신경망, 자연어 처리 분야에서 유용함

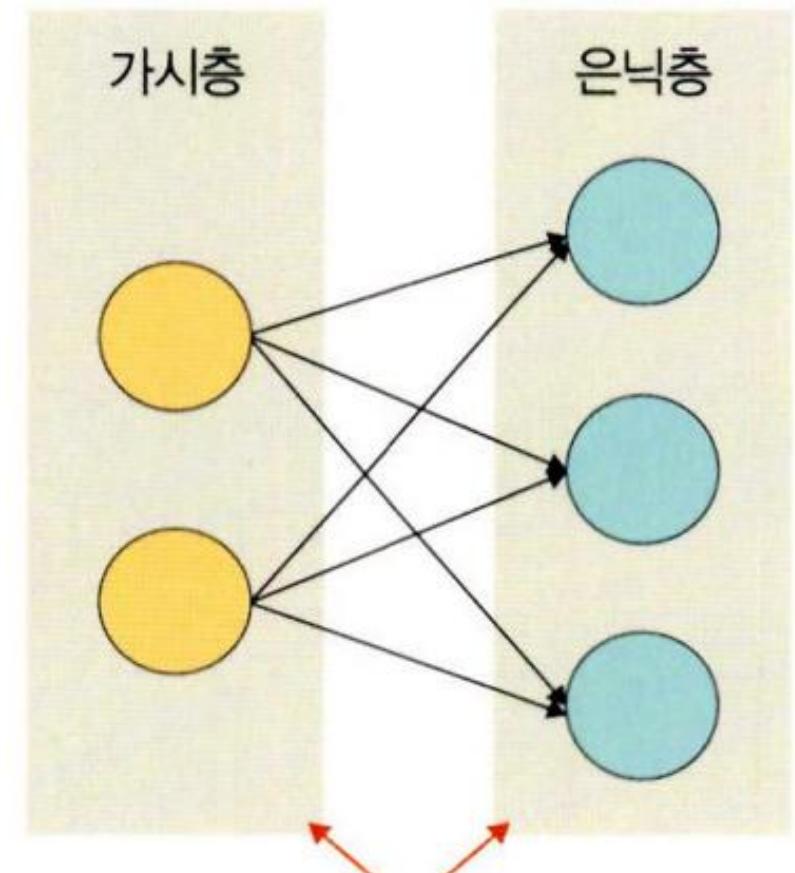
▼ 그림 4-26 순환 신경망



#03-1 제한된 볼츠만 머신(RBM)

- > 특징
- > 차원 감소, 분류, 선형 회귀 분석, 협업 필터링, 특성 값 학습, 주제 모델링에 사용됨
- > 기울기 소멸 문제 해결 위해 사전 학습 용도로 사용
- > 심층 신경망의 요소로 활용됨

=> 가시층과 은닉층으로 구성된 볼츠만 머신에서 가시층/은닉층 간 연결이 없는 모델.

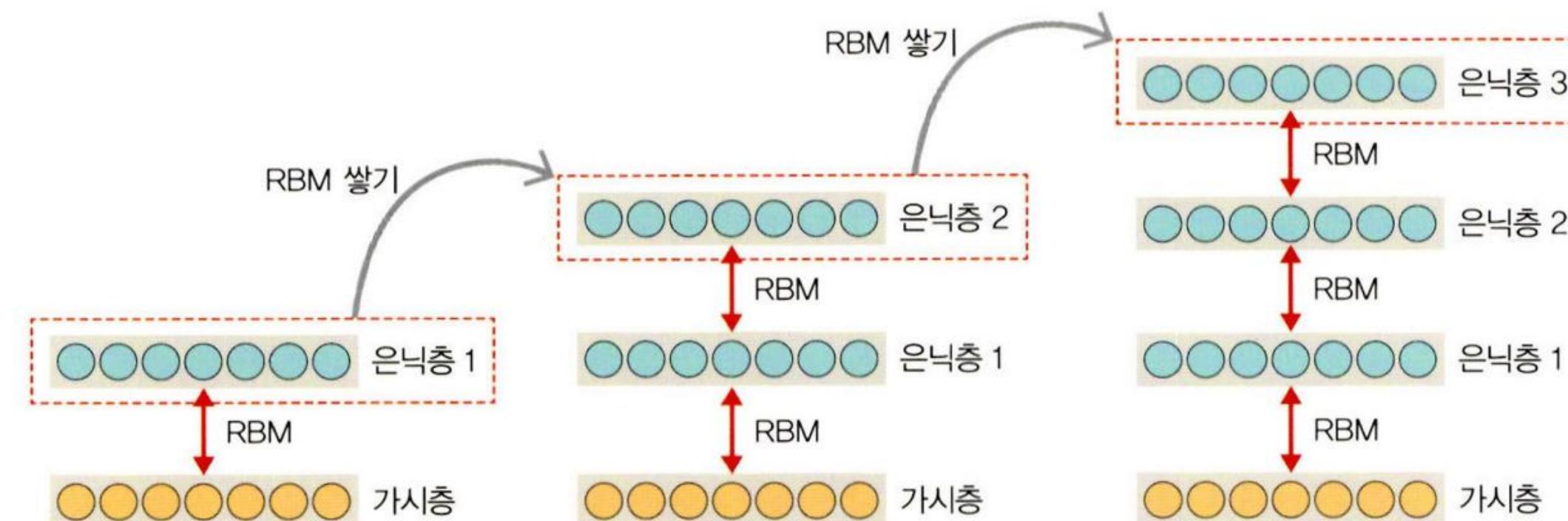


가시층 간, 은닉층 간에 연결은 없다

#03-1 심층 신경망(DBN)

> 학습 절차

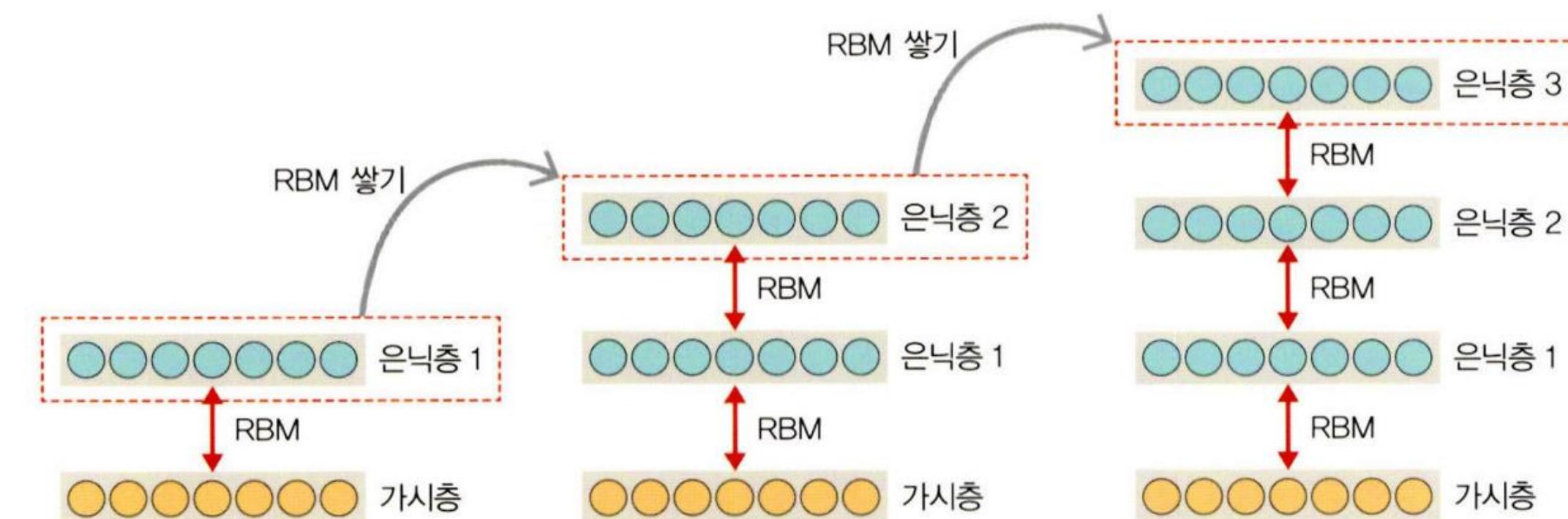
1. 가시층과 은닉층 1에 제한된 볼츠만 머신을 사전 훈련
2. 첫 번째 층 입력 데이터와 파라미터를 고정하여 두 번째 층 제한된 볼츠만 머신을 사전 훈련
3. 원하는 층 개수만큼 제한된 볼츠만 머신을 쌓아 올려 전체 DBN을 완성



#03-1 심층 신경망(DBN)

- > 특징
- 순차적으로 학습시켜 가며 계층적 구조 형성
- 비지도 학습으로 학습
- 위로 올라갈수록 추상적 특성 추출
- 학습된 가중치를 다중 퍼셉트론의 가중치 초기값으로 사용

=> 사전 훈련된 제한된 볼츠만 머신을 여러 층으로 쌓은 형태로 연결된 신경망, 일반화와 추상화 과정 구현에 유용.



03-2 우리는 무엇을 배워야 할까?



#03-2 우리는 무엇을 배워야 할까?

간단한 선형 회귀 분류 -> 머신러닝

복잡한 비선형 데이터 분류 및 예측 -> 딥러닝

=> 얻고 싶은 결과 도출을 위한 도구의 측면에서 이해

THANK YOU

