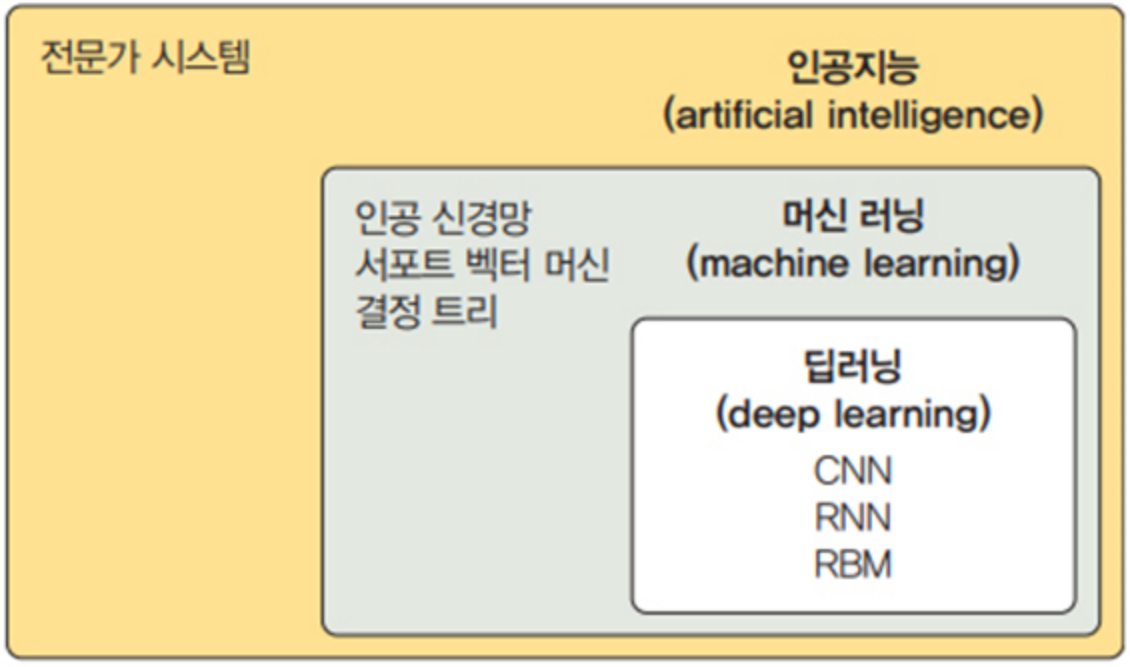
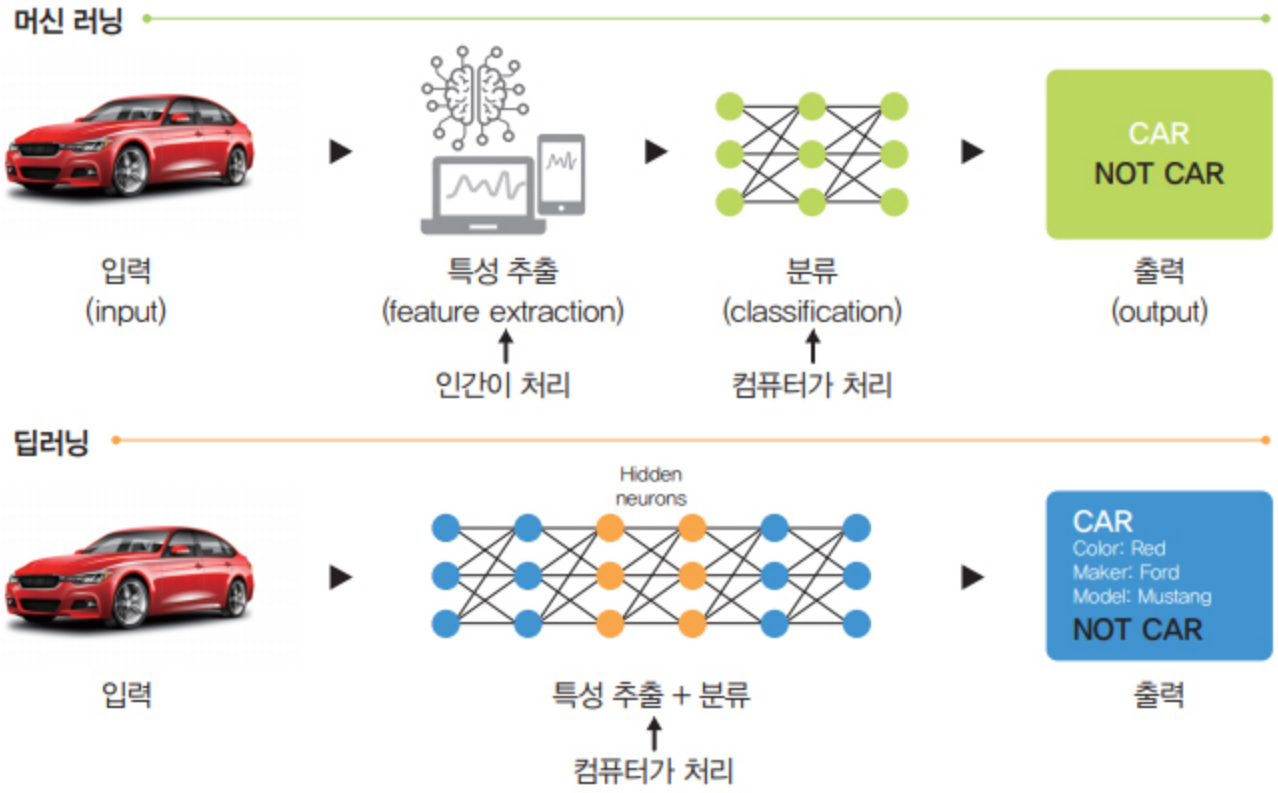
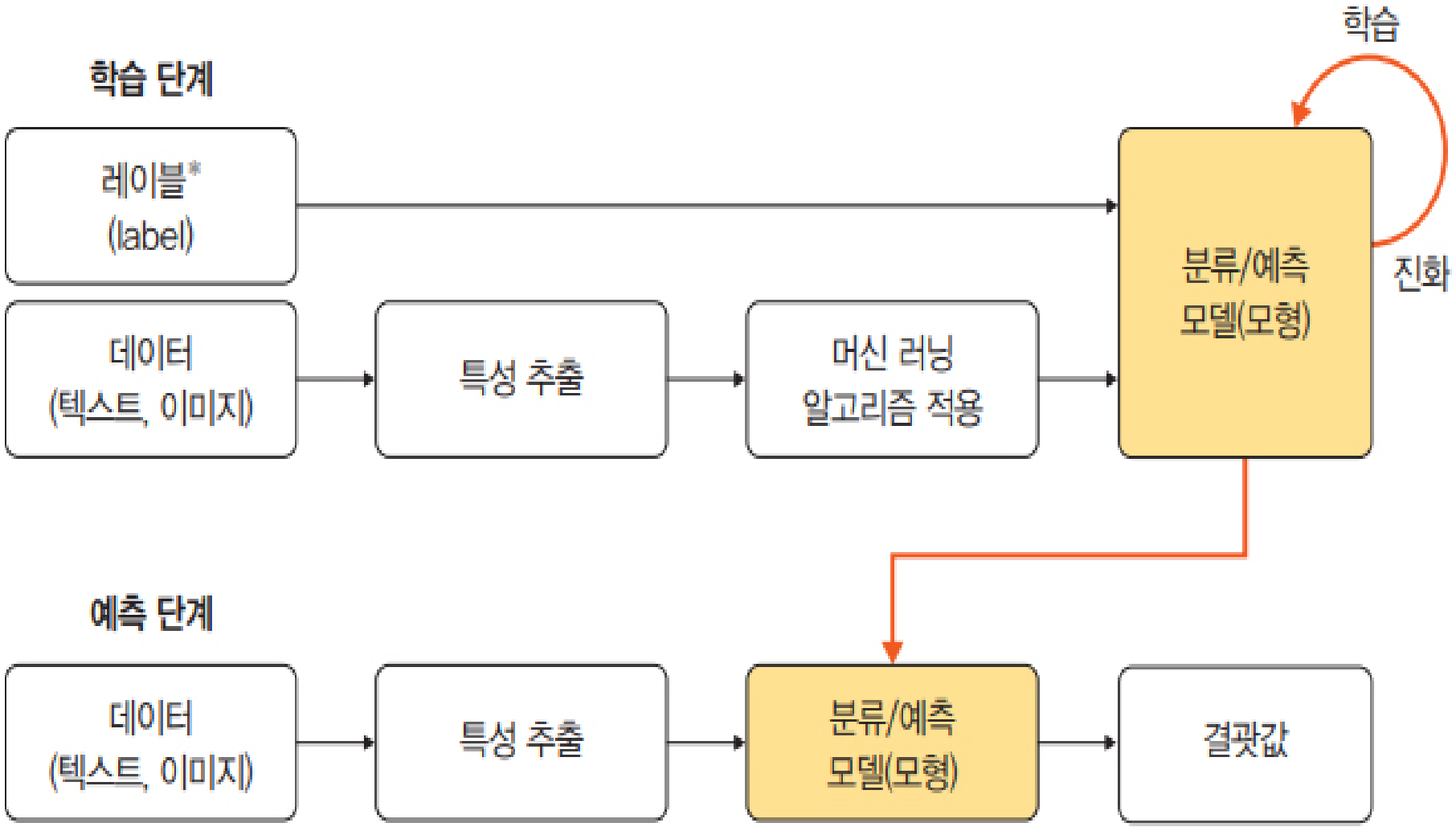
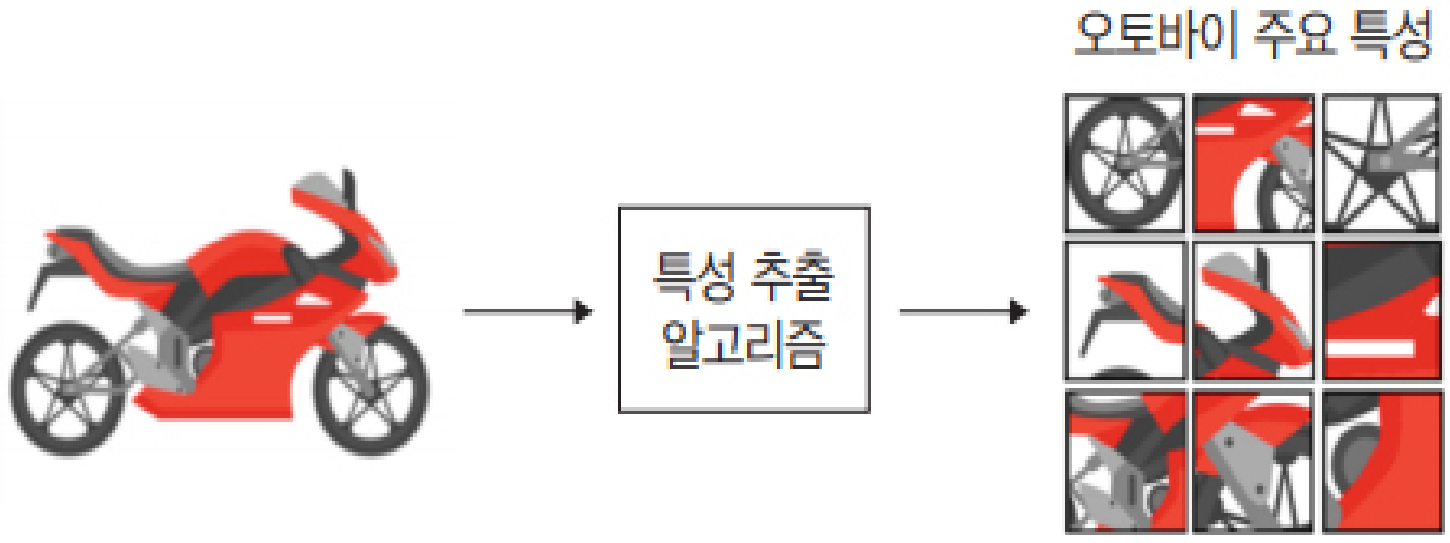

전문가 시스템
인공지능
(artificial intelligence)
인공 신경망
서포트 벡터 머신
결정 트리
머신 러닝
(machine learning)
딥러닝
(deep learning)
CNN
RNN
RBM

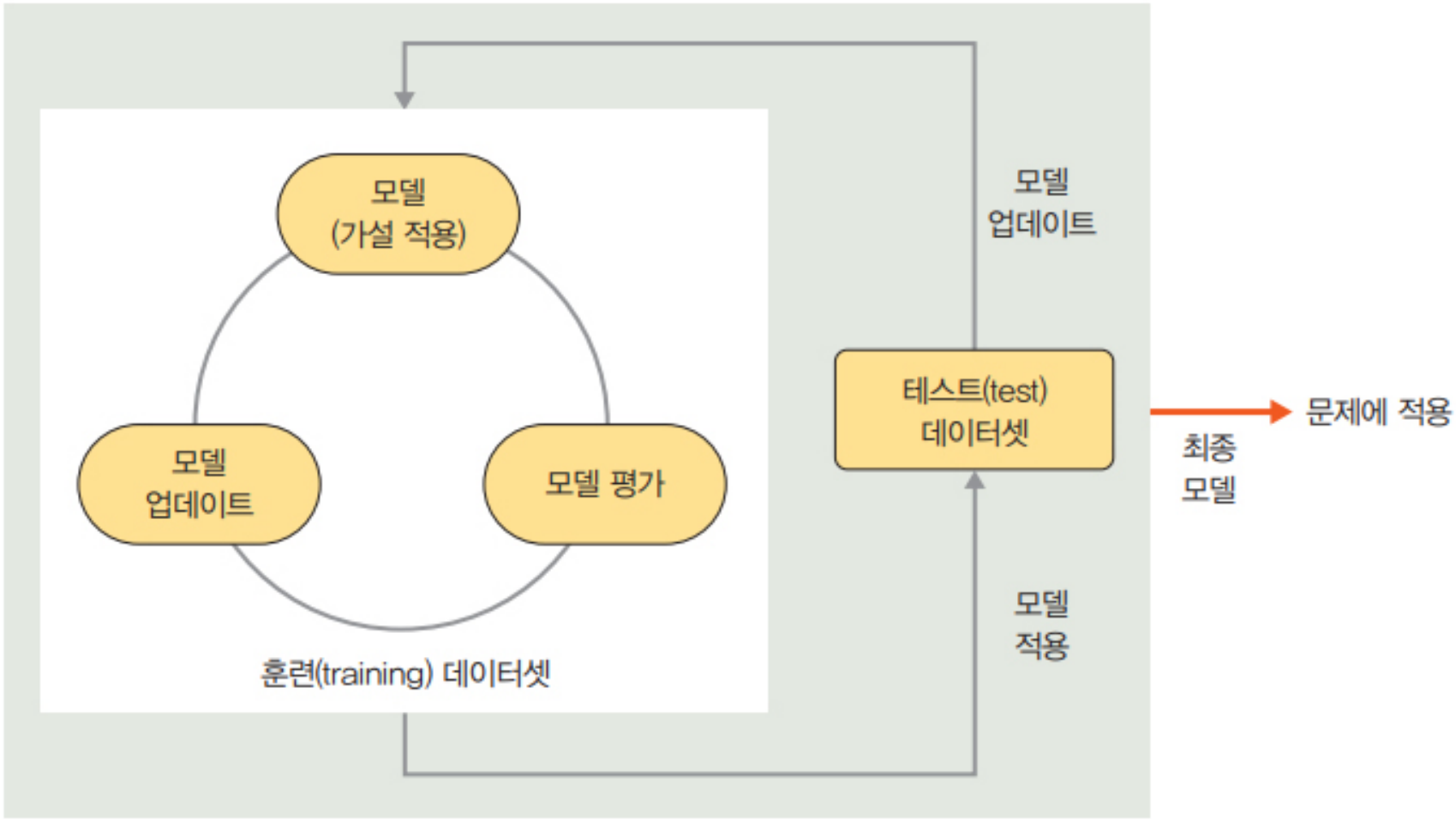

머신 러닝
입력
(input)
특성 추출
(feature extraction)
인간이 처리
분류
(classification)
컴퓨터가 처리
CAR
NOT CAR
출력
(output)
딥러닝
입력
Hidden
neurons
특성 추출 + 분류
컴퓨터가 처리
CAR
Color: Red
Maker: Ford
Model: Mustang
NOT CAR
출력

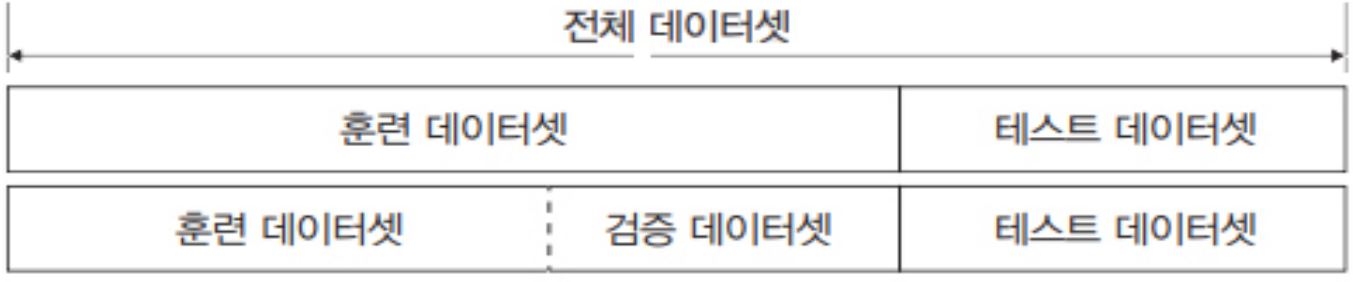
구분	머신 러닝	딥러닝
동작 원리	입력 데이터에 알고리즘을 적용하여 예측을 수행한다.	정보를 전달하는 신경망을 사용하여 데이터 특징 및 관계를 해석한다.
재사용	입력 데이터를 분석하기 위해 다양한 알고리즘을 사용하며, 동일한 유형의 데이터 분석을 위한 재사용은 불가능하다.	구현된 알고리즘은 동일한 유형의 데이터를 분석하는 데 재사용된다.
데이터	일반적으로 수천 개의 데이터가 필요하다.	수백만 개 이상의 데이터가 필요하다.
훈련 시간	단시간	장시간
결과	일반적으로 점수 또는 분류 등 숫자 값	출력은 점수, 텍스트, 소리 등 어떤 것이든 가능

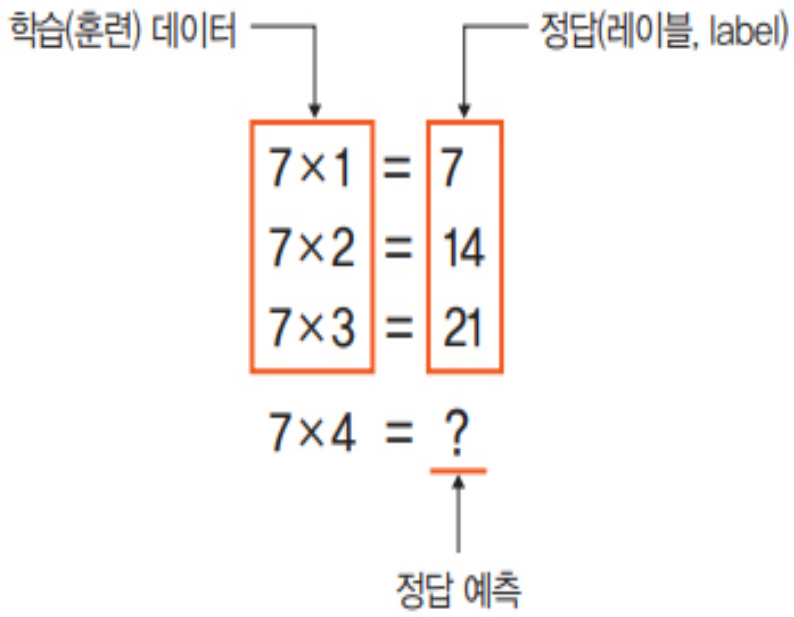


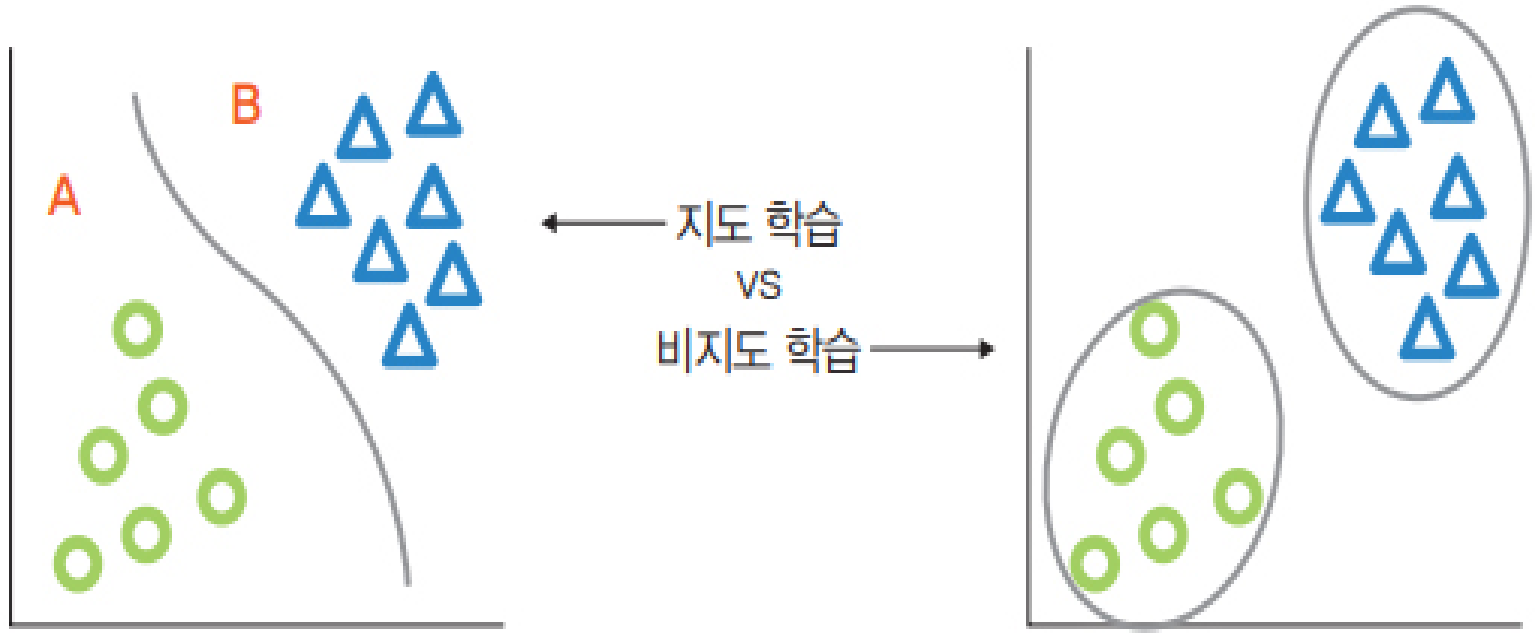
* 레이블은 지도 학습에서 정답을 의미


오토바이 주요 특성
특성 추출
알고리즘


모델
(가설 적용)
모델
업데이트
모델 평가
훈련(training) 데이터셋
모델
업데이트
테스트(test)
데이터셋
문제에 적용
최종
모델
모델
적용


전체 데이터셋
훈련 데이터셋
테스트 데이터셋
훈련 데이터셋
검증 데이터셋
테스트 데이터셋

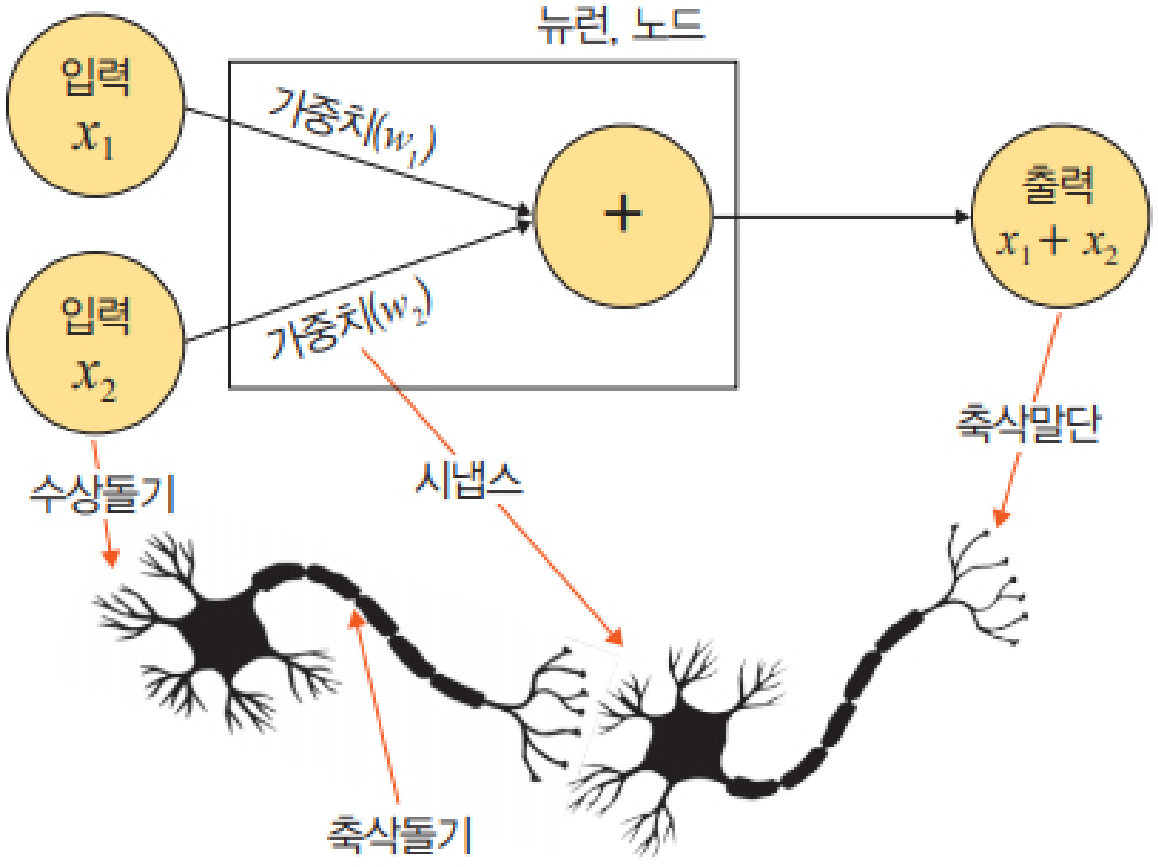

학습(훈련) 데이터
정답(레이블, label)
7×1 = 7
7×2 = 14
7×3 = 21
7×4 = ?
정답 예측


A
B
지도 학습
VS
비지도 학습

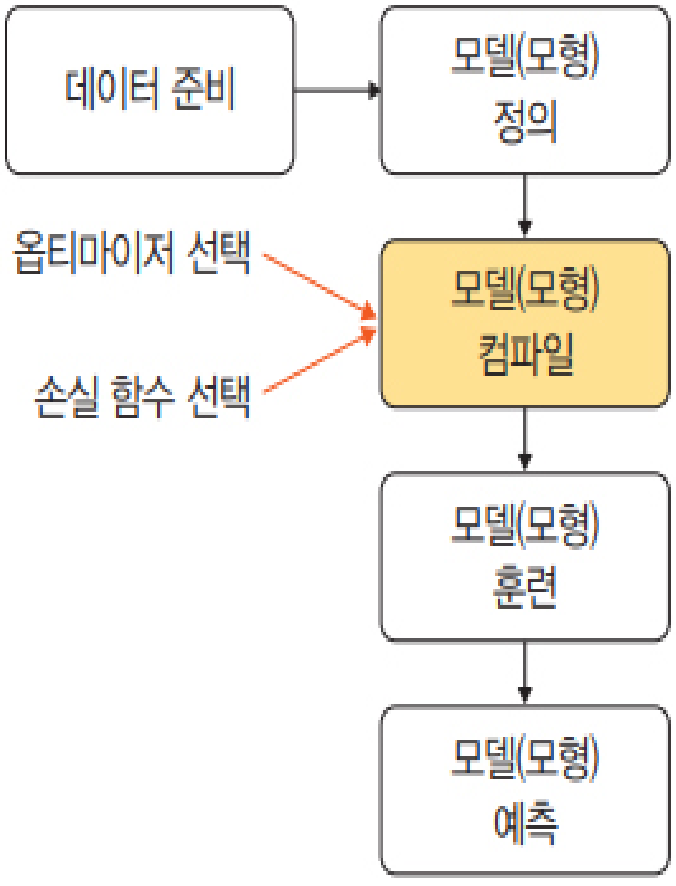
지도 학습, 비지도 학습, 강화 학습

구분	유형	알고리즘
지도 학습 (supervised learning)	분류(classification)	• K-최근접 이웃(K-Nearest Neighbor, KNN) • 서포트 벡터 머신(Support Vector Machine, SVM) • 결정 트리(decision tree) • 로지스틱 회귀(logistic regression)
	회귀(regression)	선형 회귀(linear regression)
비지도 학습 (unsupervised learning)	군집(clustering)	• K-평균 군집화(K-means clustering) • 밀도 기반 군집 분석(DBSCAN)
	차원 축소 (dimensionality reduction)	주성분 분석 (Principal Component Analysis, PCA)
강화 학습 (reinforcement learning)	-	마르코프 결정 과정 (Markov Decision Process, MDP)

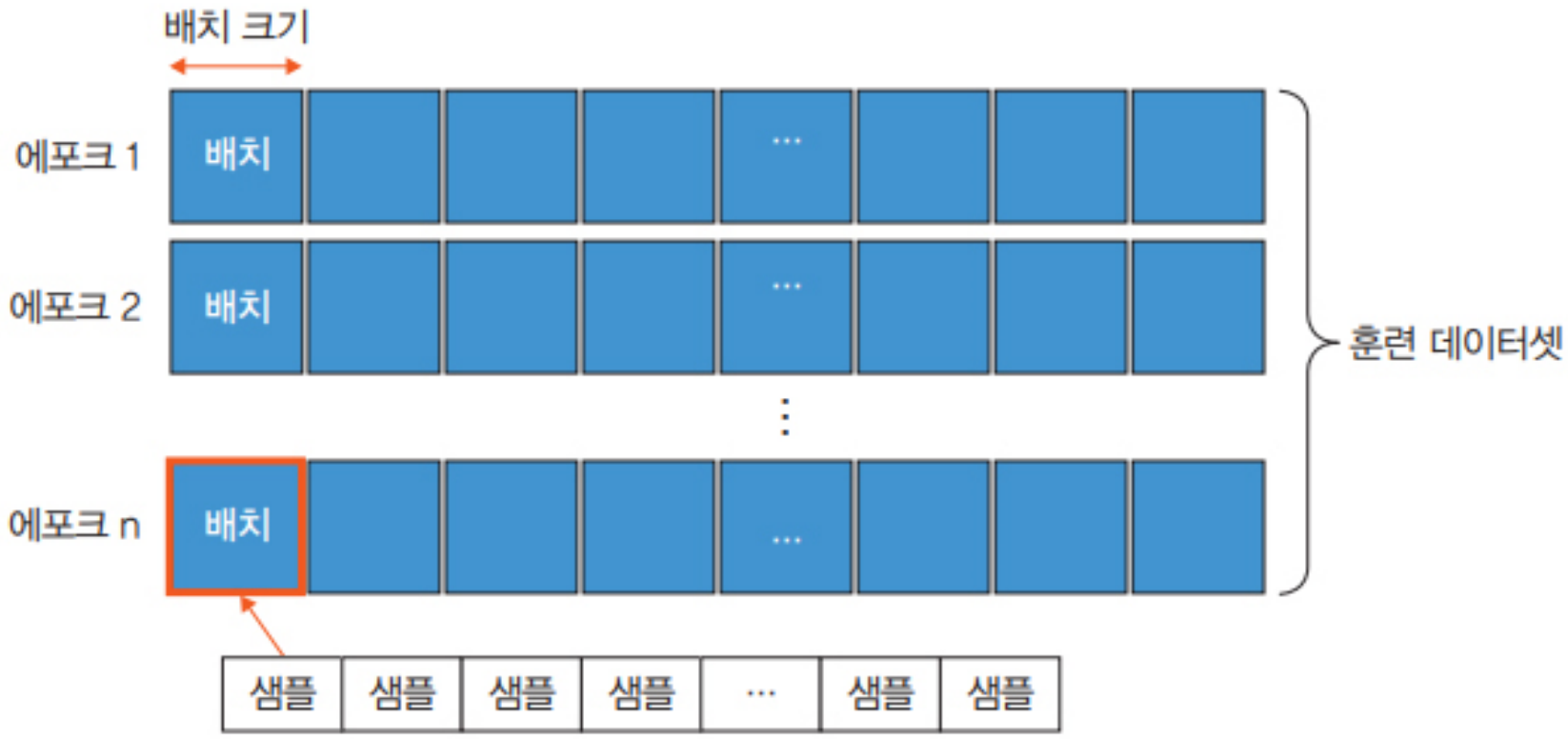
인간의 신경망 원리를 모방한 심층 신경망



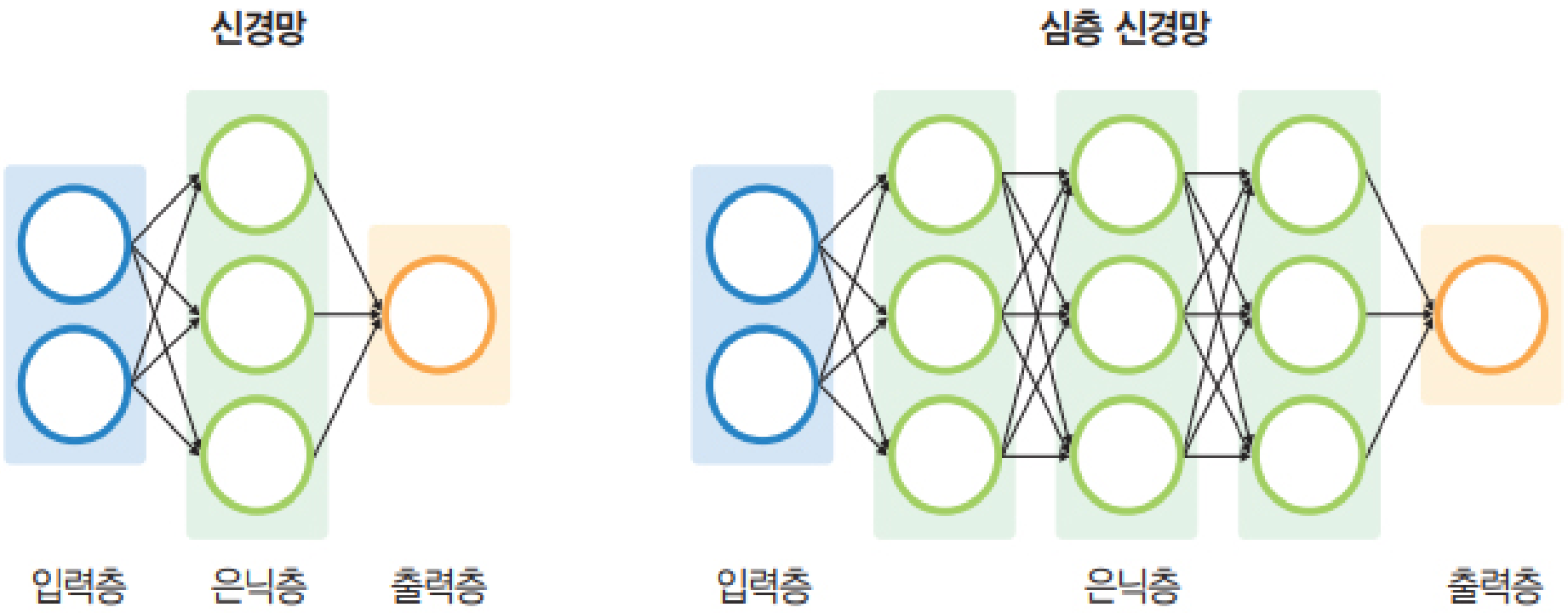
딥러닝 모델의 학습 과정



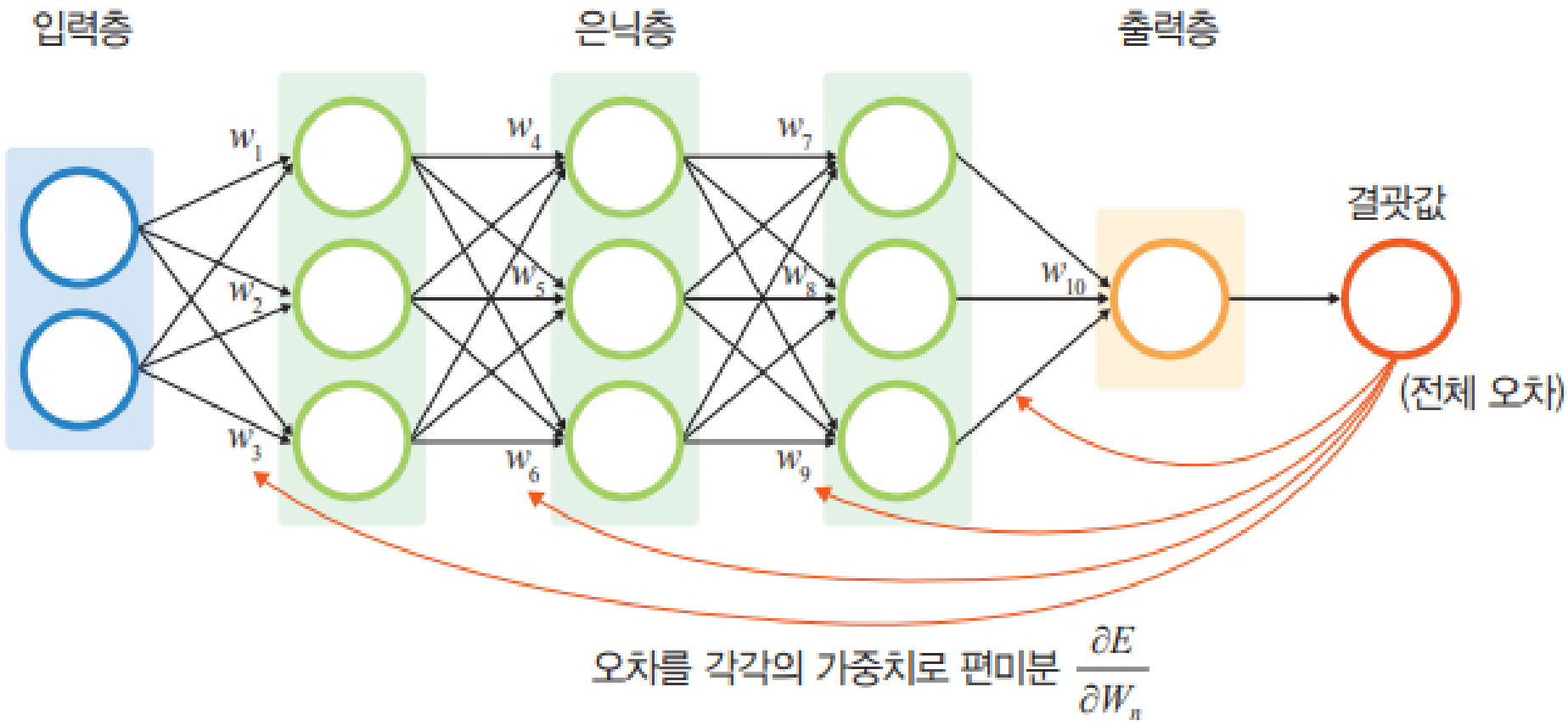
모델 훈련에 필요한 하이퍼파라미터



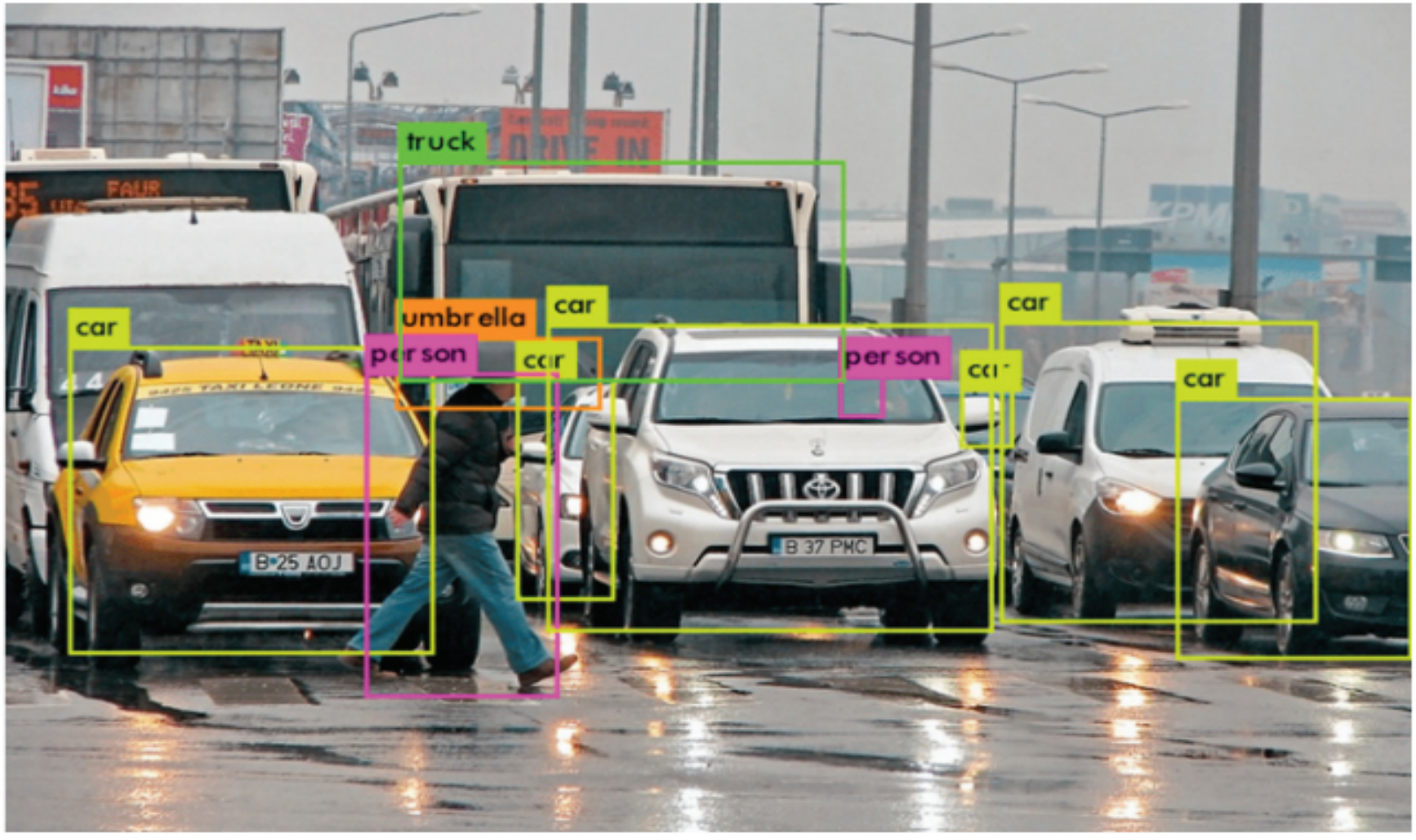
신경망과 심층 신경망



역전파 계산

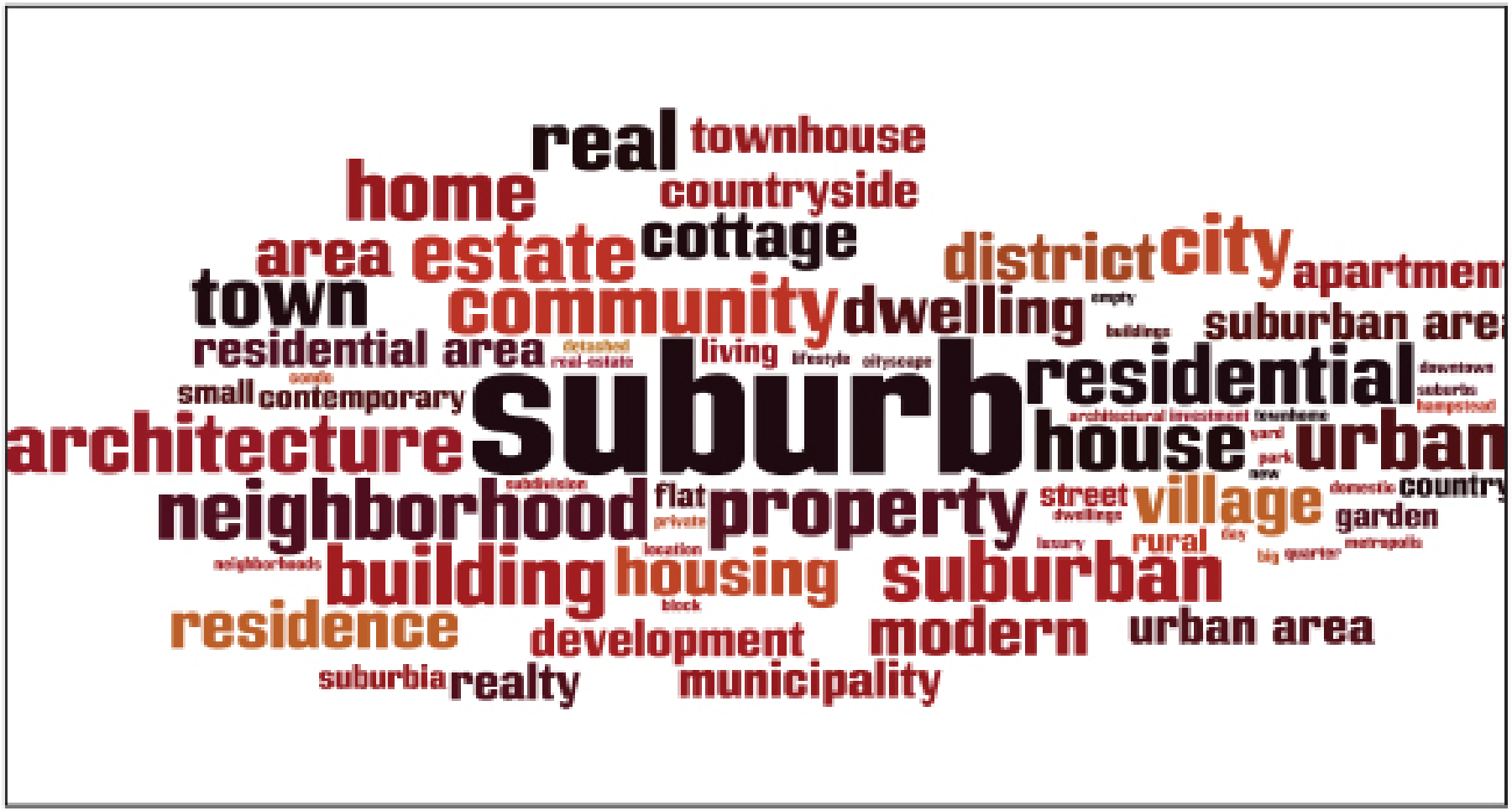

입력층
은닉층
출력층
w_1
w_2
w_3
w_4
w_5
w_6
w_7
w_8
w_9
w_{10}
결괏값
(전체 오차)
오차를 각각의 가중치로 편미분 $\frac{\partial E}{\partial W_n}$

이미지 인식

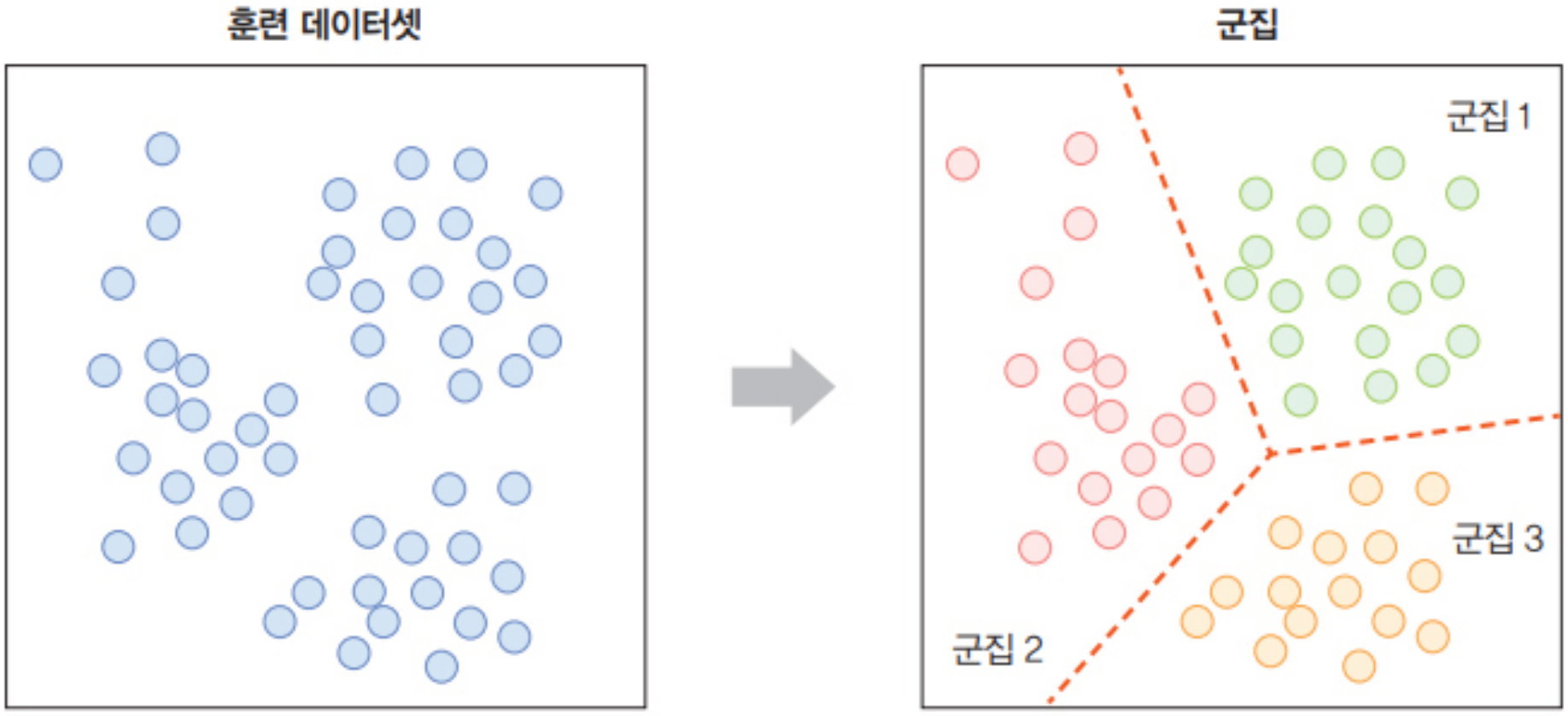

truck
car
umbrella
car
person
car
person
car
car
B 37 PMC
B 25 AOJ

구글과 아마존 주식에 대한 시계열 데이터 사례

워드 임베딩을 이용한 워드 클라우드



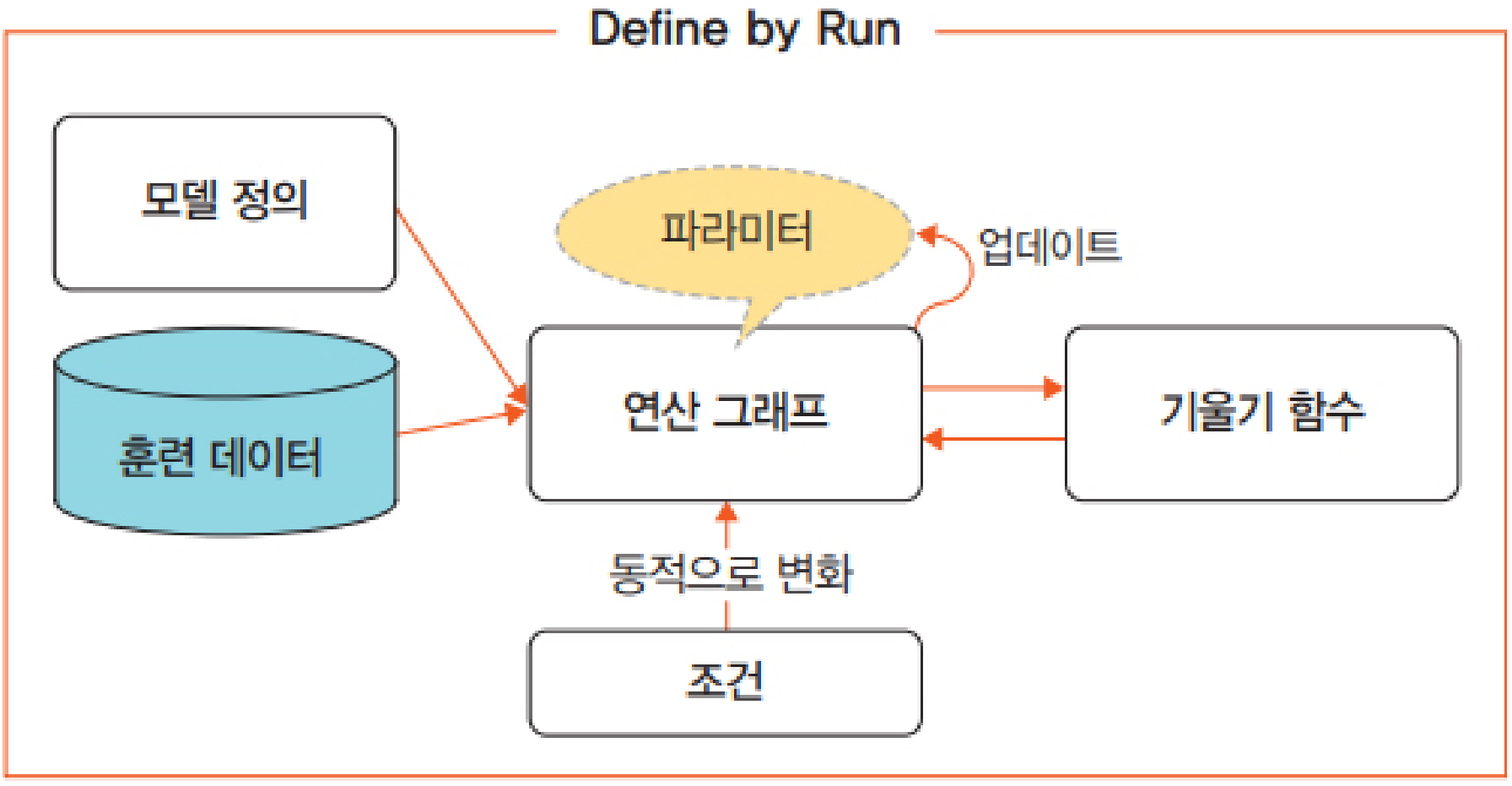
군집



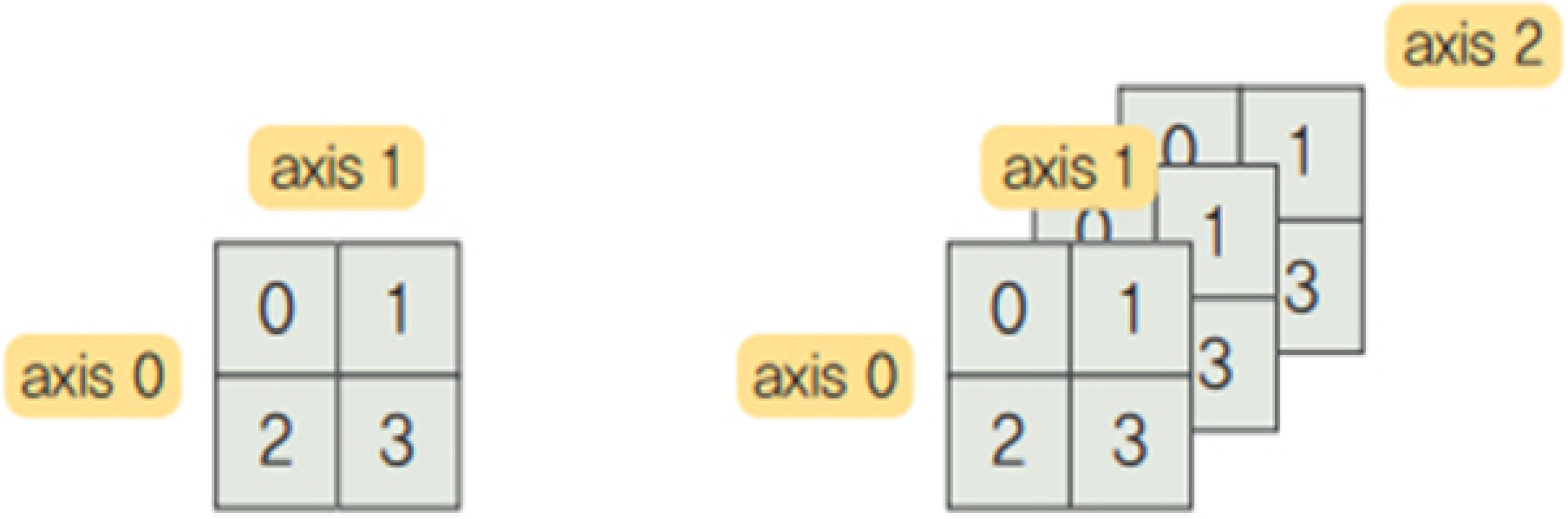
딥러닝에서 지도 학습, 비지도 학습, 강화 학습

구분	유형	알고리즘
지도 학습(supervised learning)	이미지 분류	• CNN • AlexNet • ResNet
	시계열 데이터 분석	• RNN • LSTM
비지도 학습 (unsupervised learning)	군집 (clustering)	• 가우시안 혼합 모델(Gaussian Mixture Model, GMM) • 자기 조직화 지도(Self-Organizing Map, SOM)
	차원 축소	• 오토인코더(AutoEncoder) • 주성분 분석(PCA)
전이 학습(transfer learning)	전이 학습	• 버트(BERT) • MobileNetV2
강화 학습(reinforcement learning)	–	마르코프 결정 과정(MDP)

파이토치 ‘Define by Run’

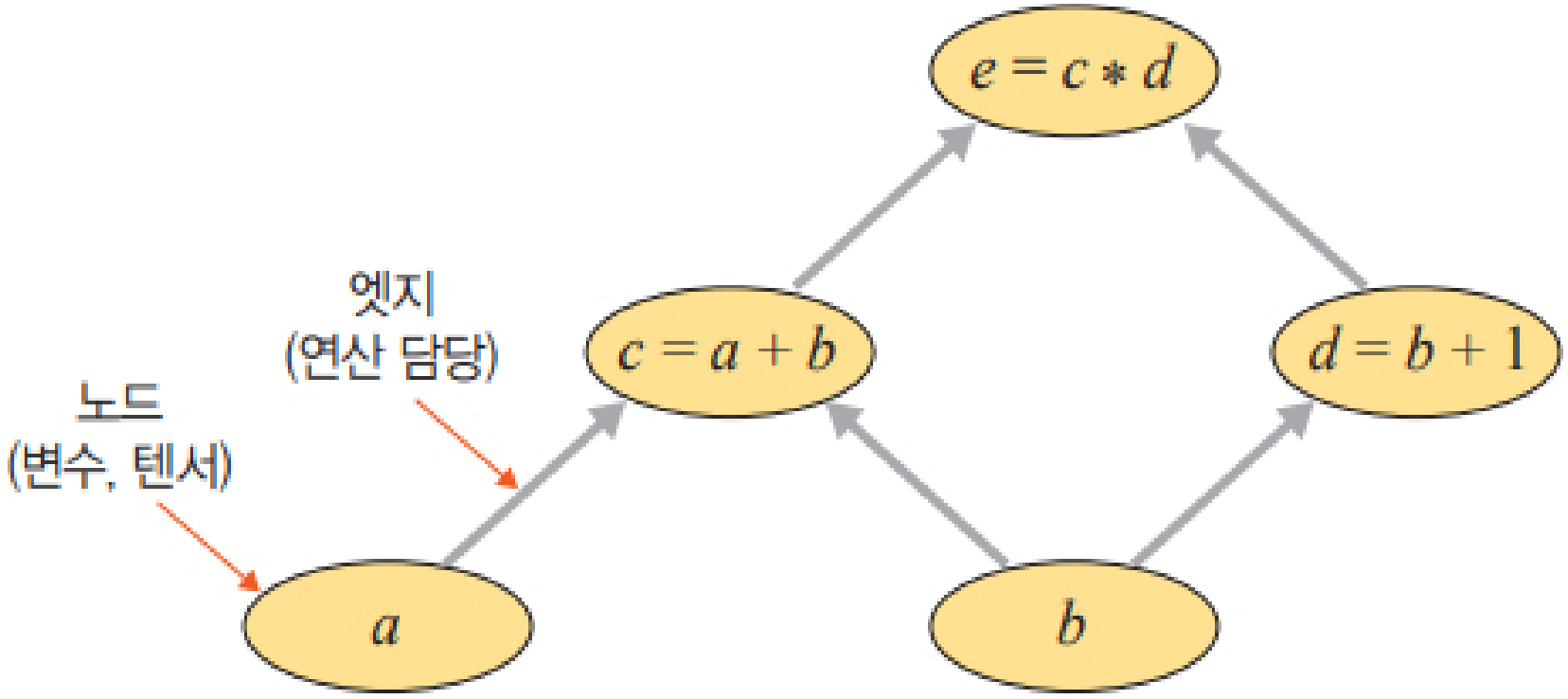


벡터, 행렬, 텐서

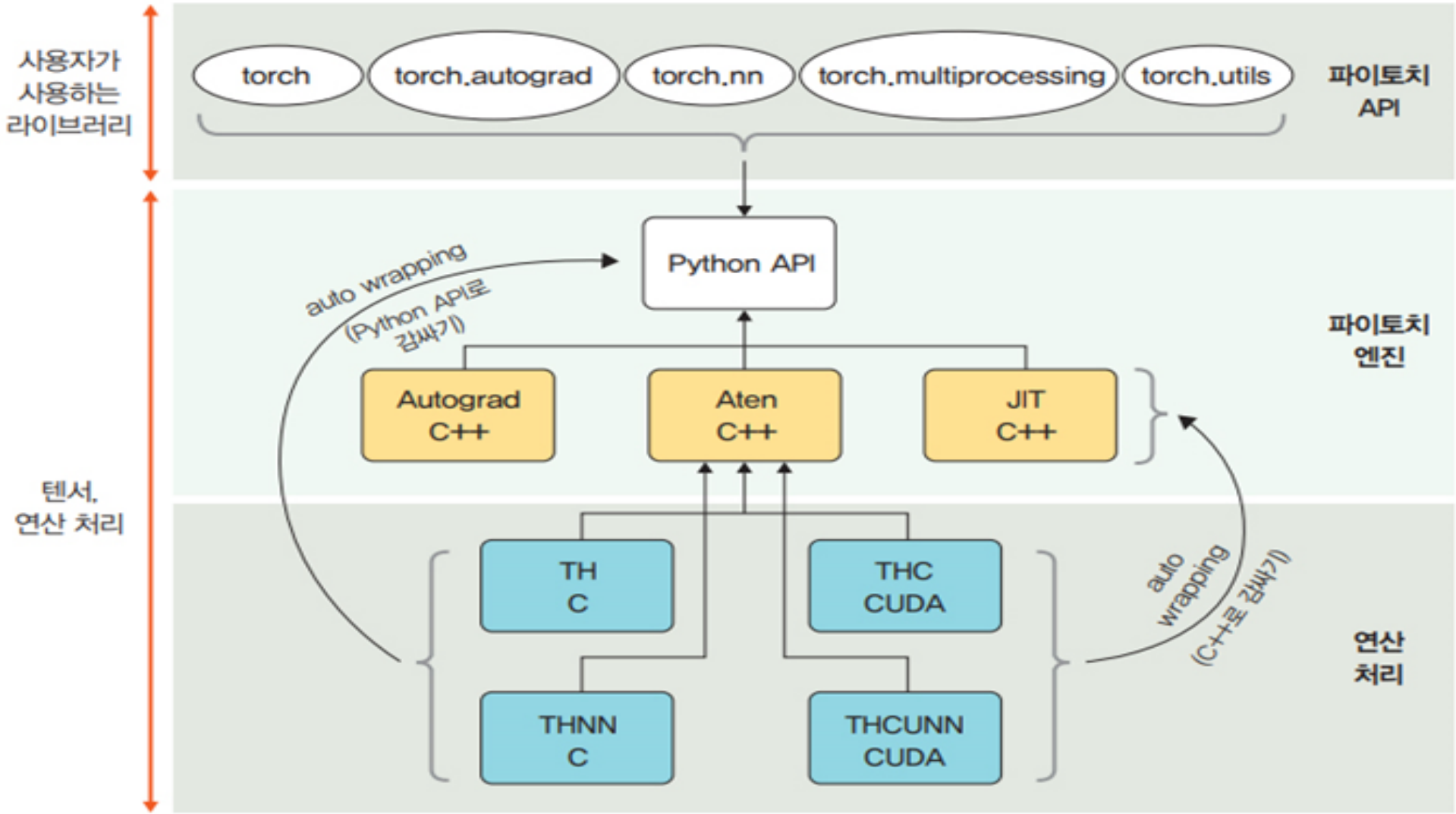


1차원 축(행)= axis 0= 벡터
2차원 축(열)= axis 1= 행렬
3차원 축(채널)= axis 2= 텐서

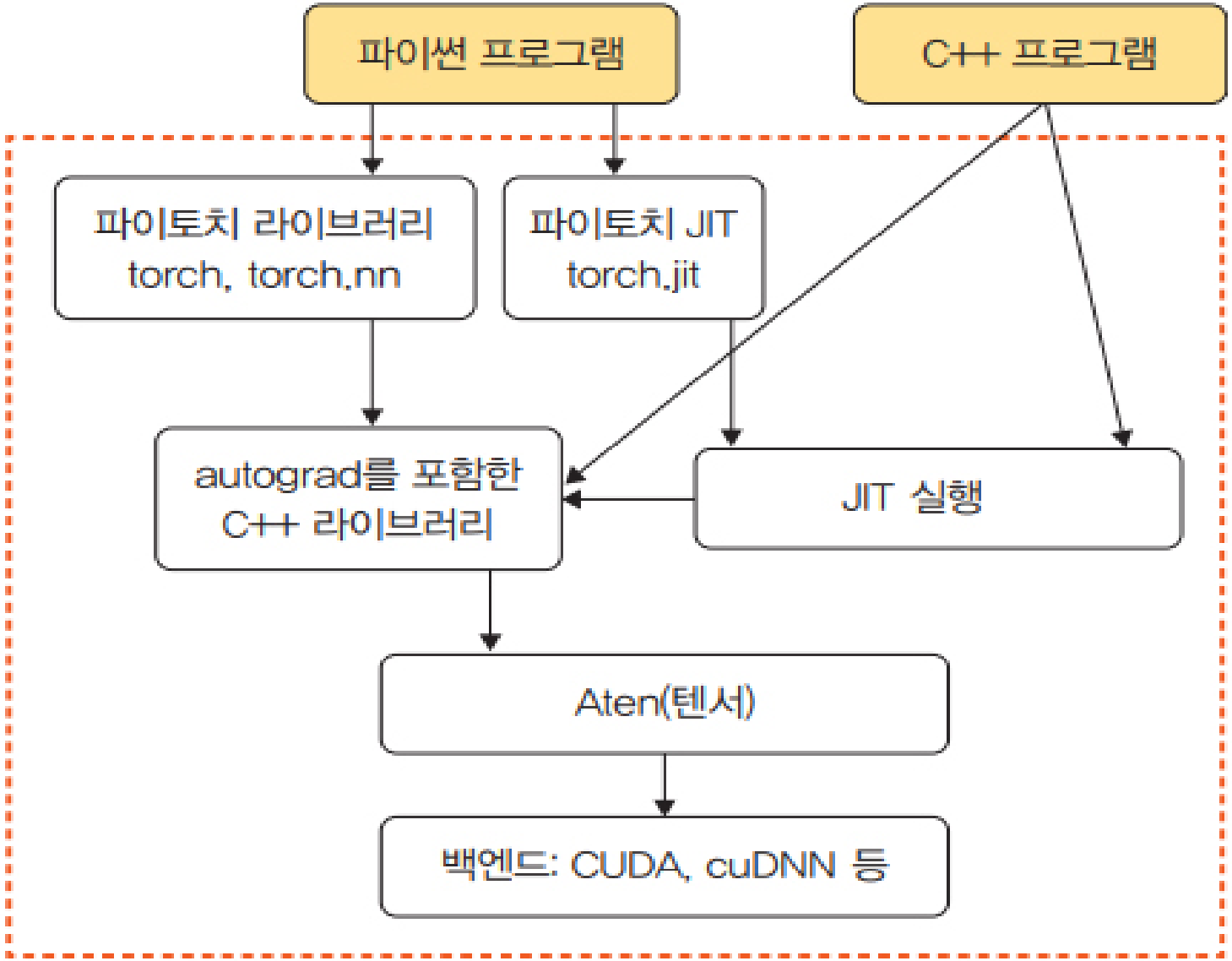
파이토치 연산 그래프



파이토치의 아키텍처


사용자가
사용하는
라이브러리
torch
torch.autograd
torch.nn
torch.multiprocessing
torch.utils
파이토치
API
텐서,
연산 처리
auto wrapping
(Python API로
감싸기)
Python API
Autograd
C++
Aten
C++
JIT
C++
파이토치
엔진
TH
C
THC
CUDA
THNN
C
THCUNN
CUDA
auto
wrapping
(C++로 감싸기)
연산
처리

파이토치의 아키텍처


파이썬 프로그램
C++ 프로그램
파이토치 라이브러리
torch, torch.nn
파이토치 JIT
torch.jit
autograd를 포함한
C++ 라이브러리
JIT 실행
Aten(텐서)
백엔드: CUDA, cuDNN 등

3차원 텐서를 1차원으로 변환

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

1차원 텐서로 변환

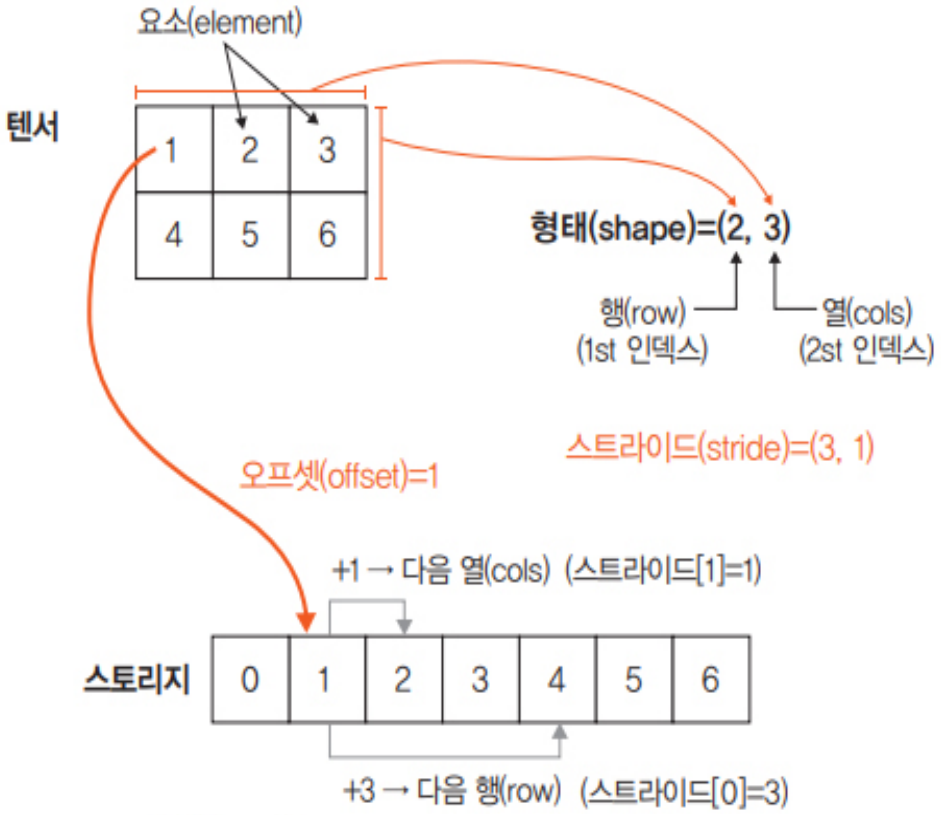
0	1	2	3	4	5	6

$$A^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

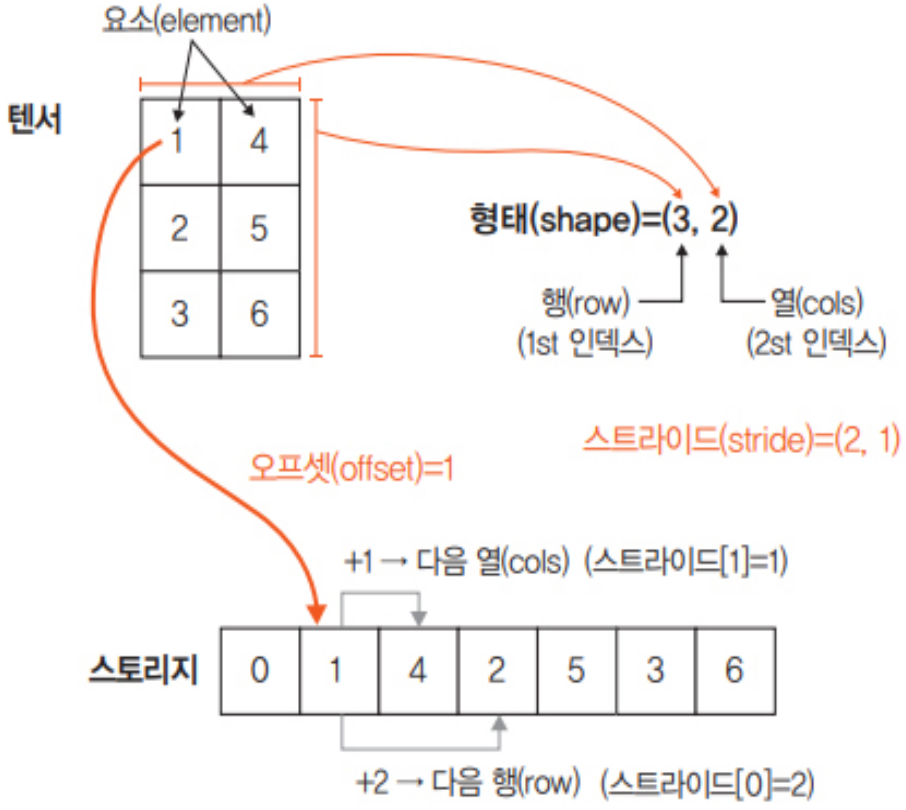
1차원 텐서로 변환

0	1	4	2	5	3	6

A를 1차원으로 변환



A의 전치 행렬을 1차원으로 변환



텐서 다루기

텐서 생성 및 변환

텐서는 파이토치의 가장 기본이 되는 데이터 구조
넘파이의 ndarray와 비슷하며 GPU에서의 연산도 가능
텐서 생성은 다음과 같은 코드를 이용

```
import torch
print(torch.tensor([[1,2],[3,4]])) ------ 2차원 형태의 텐서 생성
print(torch.tensor([[1,2],[3,4]], device="cuda:0")) ------ GPU에 텐서 생성
print(torch.tensor([[1,2],[3,4]], dtype=torch.float64)) ------ dtype을 이용하여 텐서 생성
```

```
tensor([[1, 2],
        [3, 4]])

tensor([[1., 2.],
        [3., 4.]], dtype=torch.float64)
```

텐서 다루기

텐서를 ndarray로 변환해 보자

```
temp = torch.tensor([[1,2],[3,4]])
print(temp.numpy()) ------ 텐서를 ndarray로 변환

temp = torch.tensor([[1,2],[3,4]], device="cuda:0")
print(temp.to("cpu").numpy()) ------ GPU상의 텐서를 CPU의 텐서로 변환한 후 ndarray로 변환
```

```
[[1 2]
 [3 4]]

[[1 2]
 [3 4]]
```

텐서 다루기

텐서의 인덱스 조작

```
temp = torch.FloatTensor([1, 2, 3, 4, 5, 6, 7]) ------ 파이토치로 1차원 벡터 생성
print(temp[0], temp[1], temp[-1]) ------ 인덱스로 접근
print('-----------------------')
print(temp[2:5], temp[4:-1]) ------ 슬라이스로 접근
```

```
tensor(1.) tensor(2.) tensor(7.)
-----------------------
tensor([3., 4., 5.]) tensor([5., 6.])
```

torch.FloatTensor : 32 비트의 부동 소수점
torch.DoubleTensor : 64 비트의 부동 소수점
torch.LongTensor : 64 비트의 부호가 있는 정수

텐서 다루기

텐서 연산 및 차원 조작

```
v = torch.tensor([1, 2, 3]) ------ 길이가 3인 벡터 생성
w = torch.tensor([3, 4, 6])
print(w - v) ------ 길이가 같은 벡터 간 뺄셈 연산
```

```
tensor([2, 2, 3])
```

텐서 다루기

텐서의 차원을 조작하는 코드

```
temp = torch.tensor([
    [1, 2], [3, 4]]) ------ 2×2 행렬 생성

print(temp.shape)
print('-----------------------')
print(temp.view(4, 1)) ------ 2×2 행렬을 4×1로 변형
print('-----------------------')
print(temp.view(-1)) ------ 2×2 행렬을 1차원 벡터로 변형
print('-----------------------')
print(temp.view(1, -1)) ---------- -1은 (1, ?)와 같은 의미로 다른 차원으로부터 해당 값을 유추하겠다는 것입니다. temp의 원소 개수(2×2=4)를 유지한 채 (1, ?)의 형태를 만족해야 하므로 (1, 4)가 됩니다.
print('-----------------------')
print(temp.view(-1, 1)) ------ 앞에서와 마찬가지로 (?, 1)의 의미로 temp의 원소 개수(2×2=4)를 유지한 채 (?, 1)의 형태를 만족해야 하므로 (4, 1)이 됩니다.
```

```
torch.Size([2, 2])
-----------------------
tensor([[1],
        [2],
        [3],
        [4]])
-----------------------
tensor([1, 2, 3, 4])
-----------------------
tensor([[1, 2, 3, 4]])
-----------------------
tensor([[1],
        [2],
        [3],
        [4]])
```

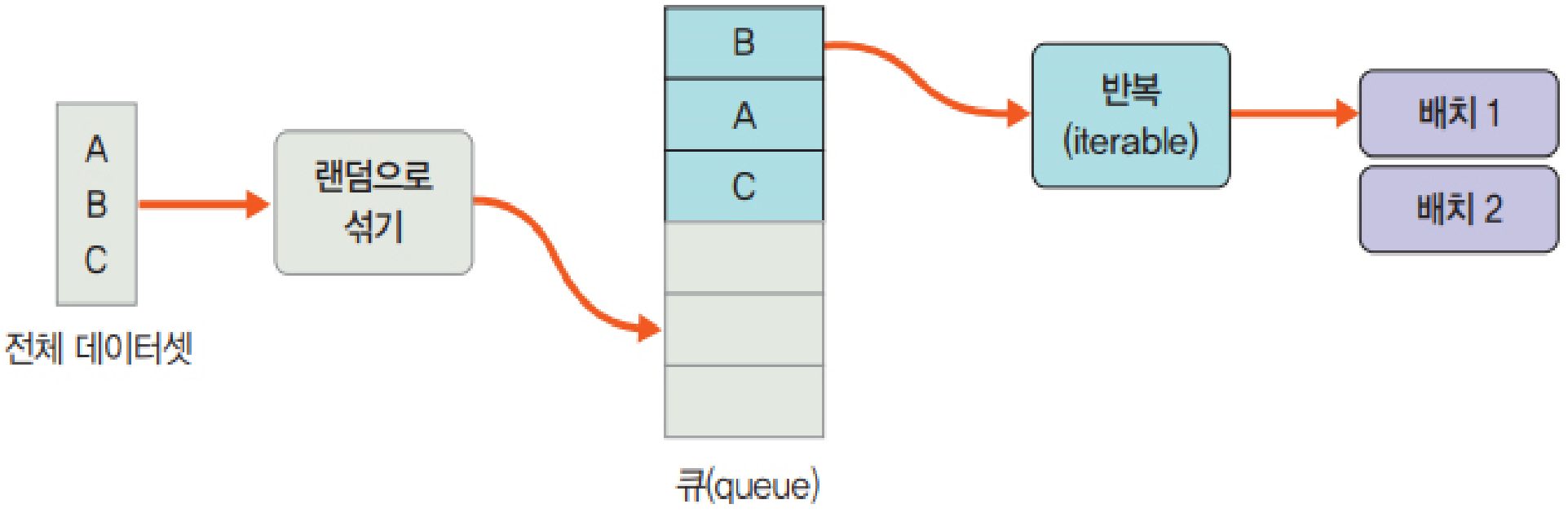
파이토치 기초 문법

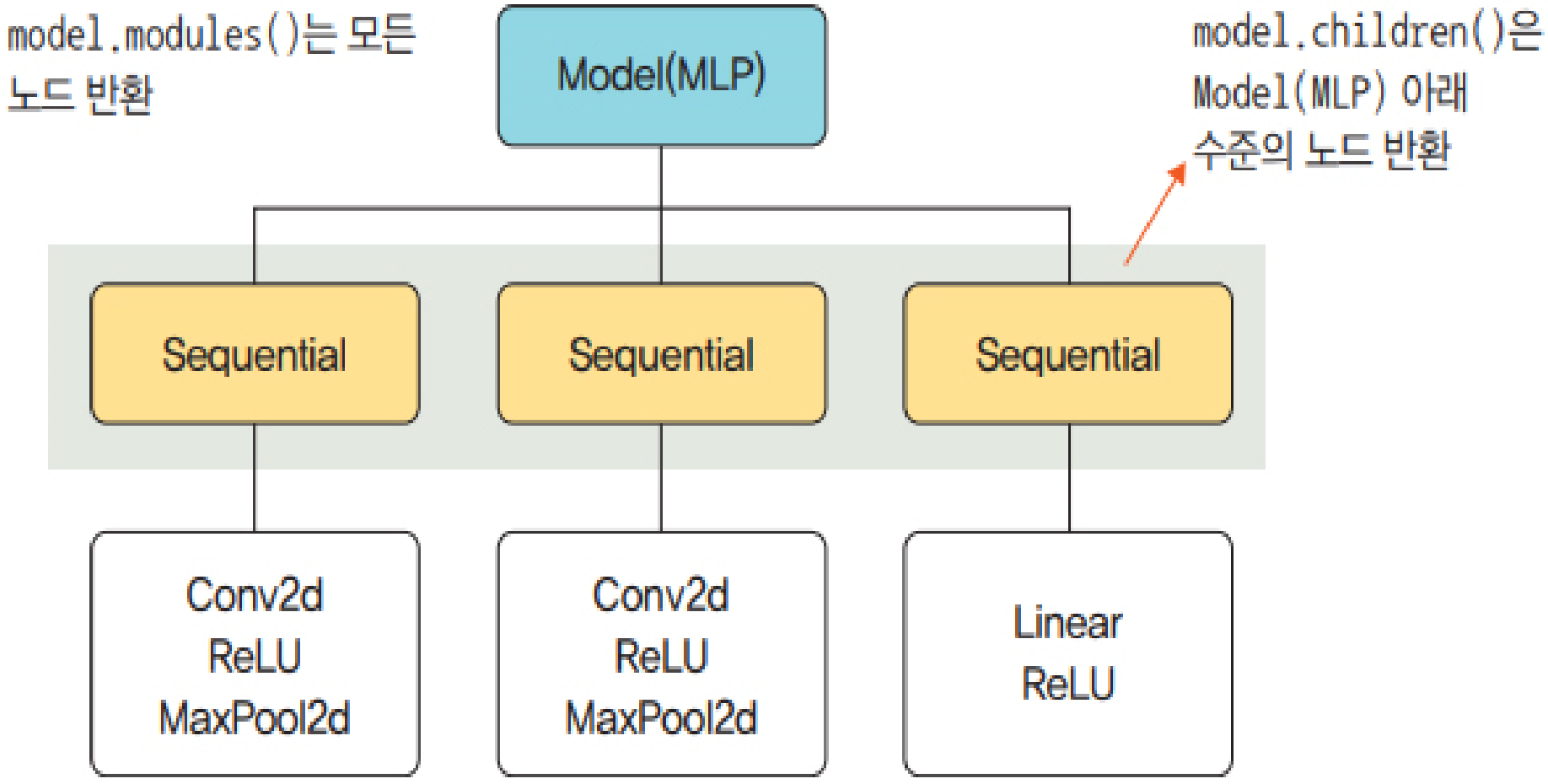
```
import pandas as pd ------ pandas 라이브러리 호출
import torch ------ torch 라이브러리 호출
data = pd.read_csv('../class2.csv') ------ csv 파일을 불러옵니다.

x = torch.from_numpy(data['x'].values).unsqueeze(dim=1).float()
y = torch.from_numpy(data['y'].values).unsqueeze(dim=1).float()
```

CSV 파일의 x 칼럼의 값을 넘파이 배열로 받아 Tensor(dtype)으로 바꾸어 줍니다.

CSV 파일의 y 칼럼의 값을 넘파이 배열로 받아 Tensor(dtype)으로 바꾸어 줍니다.

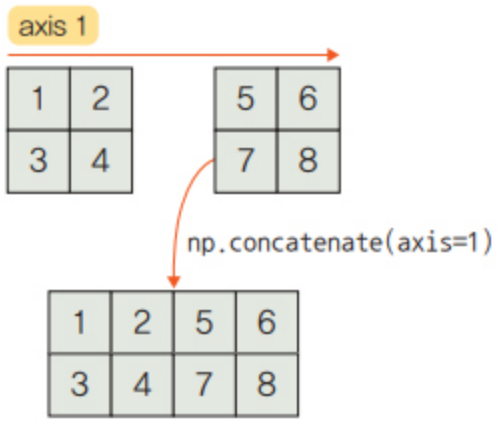

A
B
C
전체 데이터셋
랜덤으로
섞기
B
A
C
큐(queue)
반복
(iterable)
배치 1
배치 2


model.modules()는 모든
노드 반환
Model(MLP)
model.children()은
Model(MLP) 아래
수준의 노드 반환
Sequential
Sequential
Sequential
Conv2d
ReLU
MaxPool2d
Conv2d
ReLU
MaxPool2d
Linear
ReLU

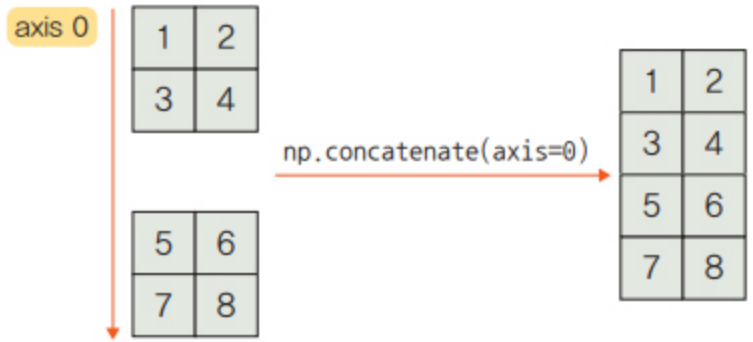
파이토치 학습 절차


딥러닝 학습 절차
파이토치 학습 절차
모델, 손실 함수, 옵티마이저 정의
전방향 학습(입력 → 출력 계산)
손실 함수로 출력과 정답의 차이(오차) 계산
역전파 학습(기울기 계산)
기울기 업데이트
모델, 손실 함수, 옵티마이저 정의
optimizer.zero_grad():
전방향 학습, 기울기 초기화
output = model(input): 출력 계산
loss = loss_fn(output, target): 오차 계산
loss.backward(): 역전파 학습
optimizer.step(): 기울기 업데이트
모델 학습 과정

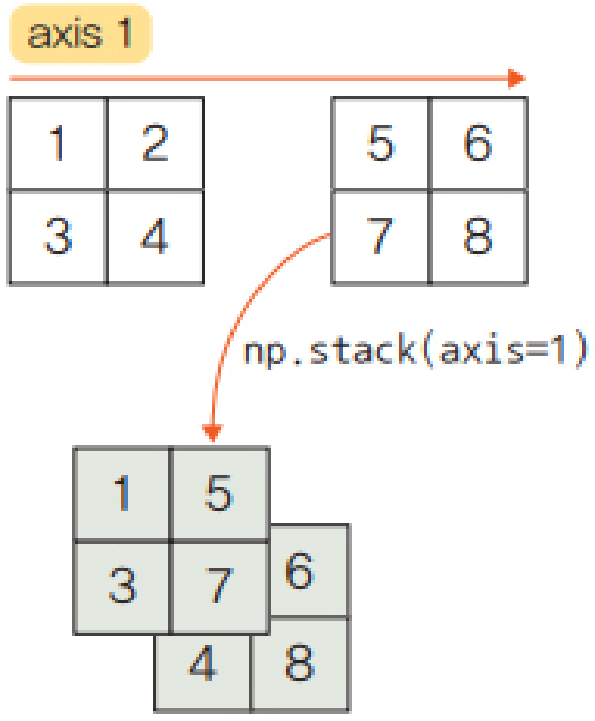
np.concatenate(axis=1)



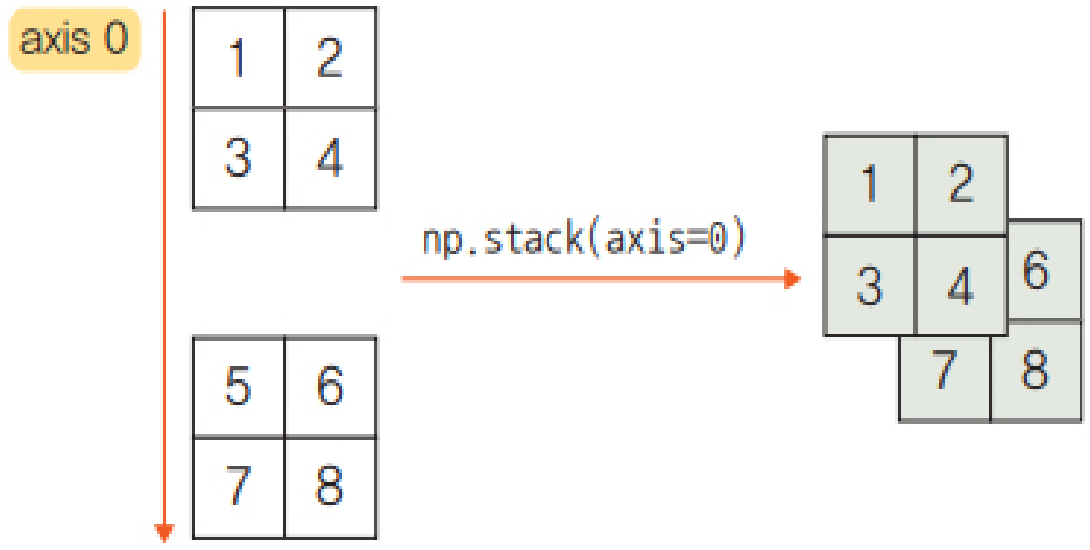
np.concatenate(axis=0)



np.stack(axis=1



np.stack(axis=0)



같은 차원을 갖는 a와 b에 대해 np.concatenate 와 np.stack을 적용

```
a = np.array([[1, 2], [3, 4]]) ------ a.shape=(2, 2)
b = np.array([[5, 6], [7, 8]]) ------ b.shape=(2, 2)
c = np.array([[5, 6], [7, 8], [9, 10]]) ------ c.shape=(3, 2)

print(np.concatenate((a, b), axis=0)) ------ shape=(4, 2)
print('------------------------------')
print(np.stack((a, b), axis=0)) ------ shape=(2, 2, 2)
```

```
[[1 2]
 [3 4]
 [5 6]
 [7 8]]
------------------------------
[[[1 2]
  [3 4]]

 [[5 6]
  [7 8]]]
```