






IT와 비즈니스혁신

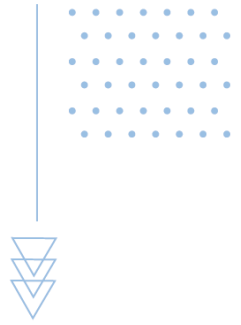
W14. 마이닝 기법 IV: 인공신경망





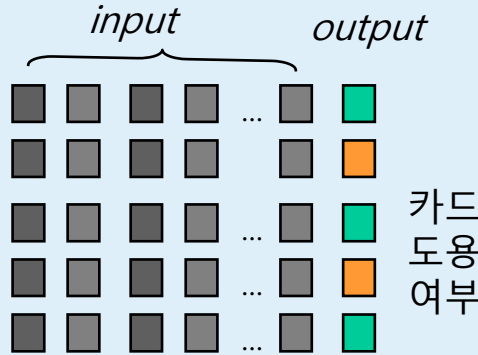
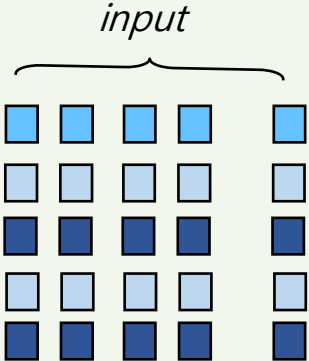
Contents

- I. 인공지능경망 개요
 - II. 인공지능경망 기법원리: 역전파 기법
 - III. 텐서플로우로 ANN구현
 - IV. 정리
- 
- 
- 



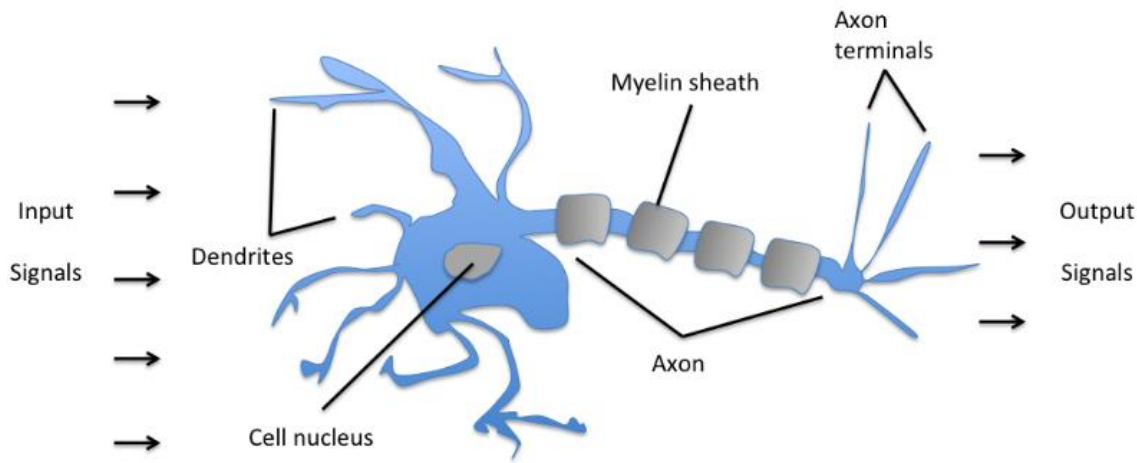
I . 인공지능 개요

데이터 마이닝은 크게 **출력 변수**의 존재 여부에 따라 **지도학습과 자율학습**으로 나눌 수 있음

	지도학습 (supervised learning)	자율학습 (unsupervised learning)
의미	<ul style="list-style-type: none"> 입력 데이터와 정답(Label)을 제공 받아 이를 통해 입력(독립)과 출력 (Label, 종속,타겟) 으로 매칭할 수 있는 규칙 생성 <p>예. 카드번호, 성별, 나이, 거래 내역 등 →  카드도용 여부</p>	<ul style="list-style-type: none"> 외부에서 정답(Label)이 주어지지 않음 입력 데이터에서 패턴을 찾아내는 작업 <p>예. 군집화: 주어진 데이터를 3 개의 그룹으로 나눔 </p>
특징	출력 변수가 존재함	출력 변수가 존재하지 않음
분석 기법	의사결정나무, 회귀분석, 인공신경망 , 판별분석 등	군집분석, 연관성 분석 등

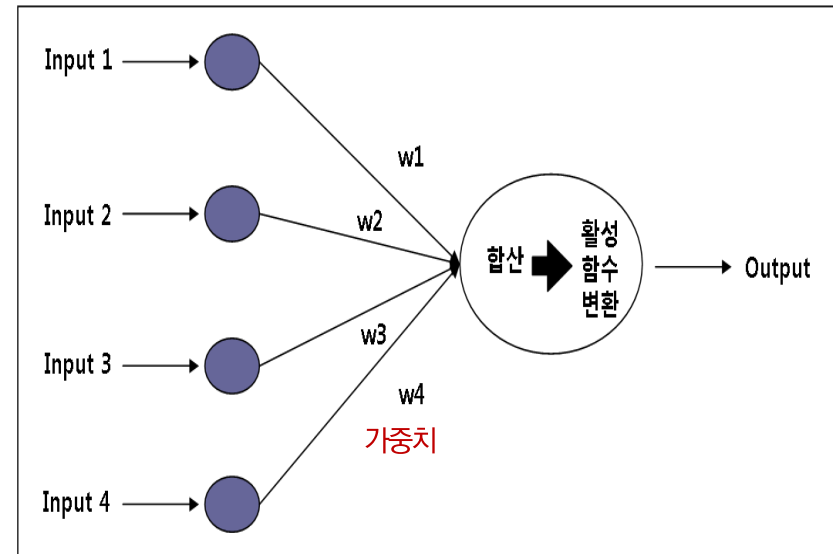
인간의 의사결정 방식을 모방

- 인간이 의사결정을 위하여 사고하는 방식을 컴퓨터에서도 구현하기 위하여 개발된 방법으로 인간 두뇌구조와 유사하게 인공 신경망을 구성
- 뉴런의 생물학적 과정을 닮은 수학적 관계를 개발
 - 인간의 두뇌에는 1000억개 가까운 뉴런이 연결되어 있음
 - 뉴런들은 서로 화학적 신호를 통해서 정보를 전달함



Schematic of a biological neuron.

최초의 인공지능망: Perceptron 퍼셉트론 모형 (Frank Rosenblatt 1957)

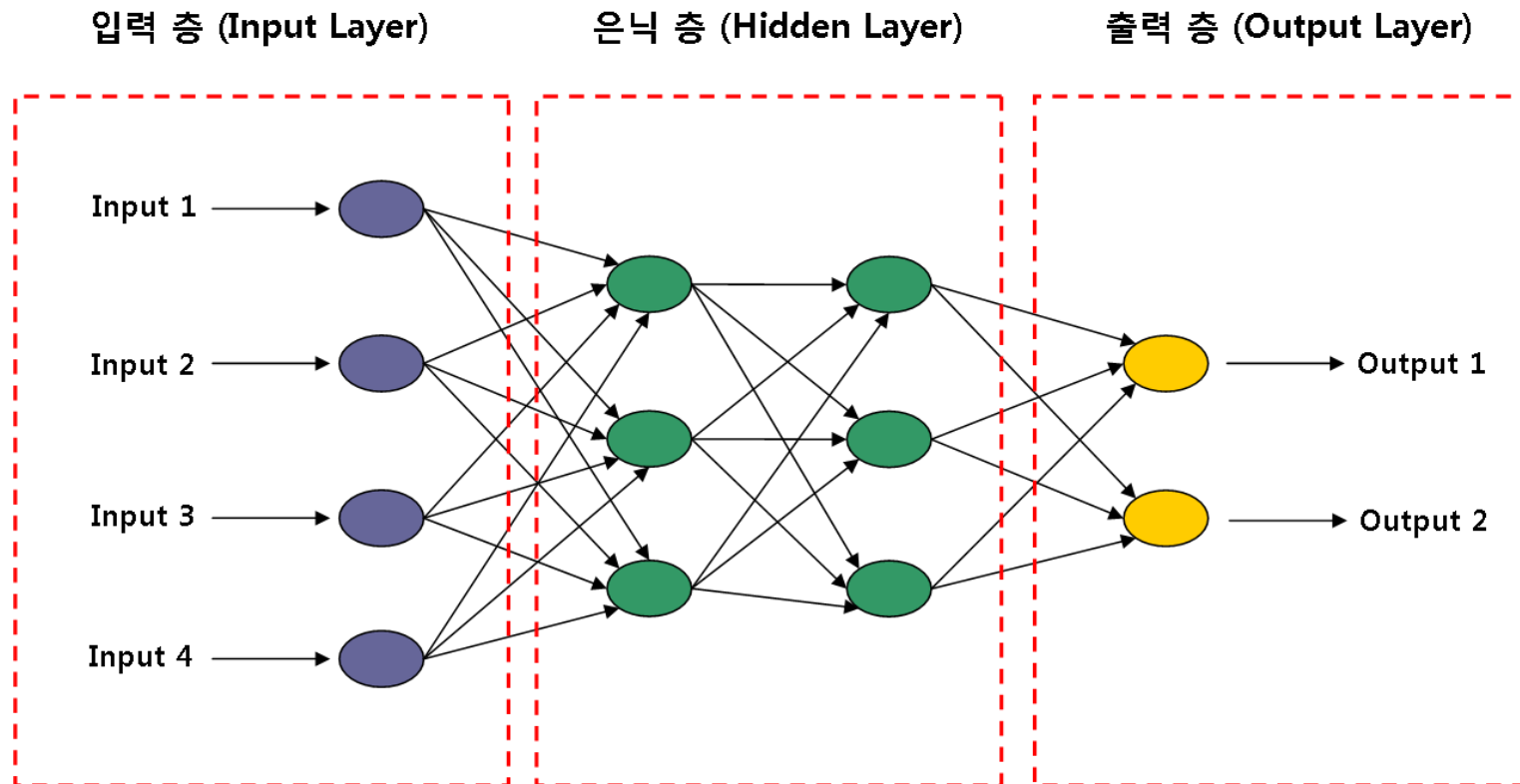


* 하나의 퍼셉트론은 회귀분석과 동일하게 동작함

$$Y = \text{activation}(W_1 * X_1 + W_2 * X_2 + W_3 * X_3 + W_4 * X_4)$$

Multi Layer Perceptron (다층 퍼셉트론)

- 입력 층 (Input layer), 은닉 층 (Hidden layer), 출력 층 (Output layer)으로 구성
- 단층 신경망보다 복잡한 비선형 문제를 해결하려면 하나 이상의 은닉 층 필요
- 입력 또는 바로 앞의 은닉 층으로부터 전달되는 값들을 이용하여 합성, 활성화 함수를 처리하고 다음으로 전달

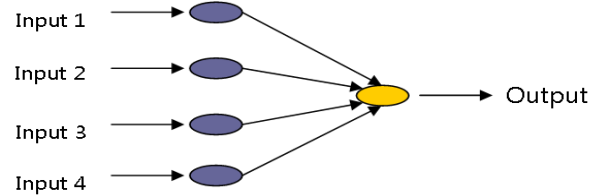


■ 텐서플로우 플레이그라운드 (<https://playground.tensorflow.org/>)

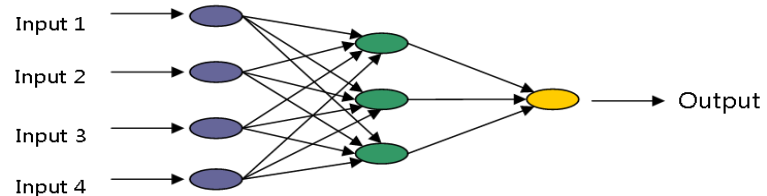
- 텐서플로 팀에서 만든 신경망 시뮬레이터로 신경망 구조와 하이퍼 파라미터(hyper parameter)를 조작하여 신경망 작동방식을 시뮬레이션으로 보여줌
- 신경망 구조, 하이퍼 파라미터 튜닝이 성능에 크게 영향을 미침

은닉 층 구조에 따른 신경망의 의미 * 은닉층이 깊어지면 deep learning

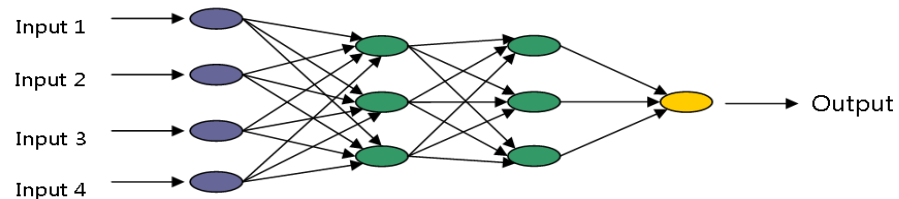
은닉 층이 없이 입력 층과 출력
층으로 구성된 모델. 로지스틱
회귀분석과 동일한 결과를 제공



은닉 층이 추가되면 이것이 없는
경우보다 더 많은 패턴을 찾을 수
있기 때문에 강력한 분류/예측 결
과를 제공할 수 있음



은닉 층의 노드가 증가할수록 모
형의 설명력이 강화되나, 과적합
(over-fit)문제가 발생할 수 있음



■ 신경망 구조

- 은닉층의 수
- 은닉층별 뉴런(노드)의 수
- (은닉층의 형태: CNN, RNN 등)

■ 하이퍼파라미터

- 활성화함수: 합성된 값을 출력값으로 변환
 - 은닉층: ReLu, tanh, sigmoid 등
 - 출력층: sigmoid, softmax
- 학습률
- 손실함수
- 규제 및 규제율
- Epoch 수

- 은닉 층과 은닉 노드가 많을수록 신경망을 통해 더 많은 패턴을 찾을 수 있음
- 은닉 층의 개수가 너무 많다면 학습용 데이터에만 예측력이 높아짐
 - 학습용 데이터에만 적합한 과적합(overfitting)의 문제 발생
- 은닉 층의 개수는 얼마가 적당한가?
 - 데이터의 양, 찾고자 하는 출력 패턴의 개수, 신경망의 유형에 따라 달라짐
 - 은닉 층과 은닉 노드의 개수를 충분히 많이 정한 후 줄여가면서 정확도는 높으면서 은닉 층과 노드의 수가 적은 모델을 선택함
 - 은닉 층의 개수보다 은닉 노드의 개수 (즉 계산에 이용되는 가중치의 개수)가 더 중요

1943년, 뉴런의 작동 원리를 설명하기 위한 간단한 모형 제안

- Warren McCulloch와 Walter Pits (예일대 신경심리학자, 논리학자)
- 두뇌의 해부학적 원리를 탐구하는 목적, 인공지능 분야에 영감 제공

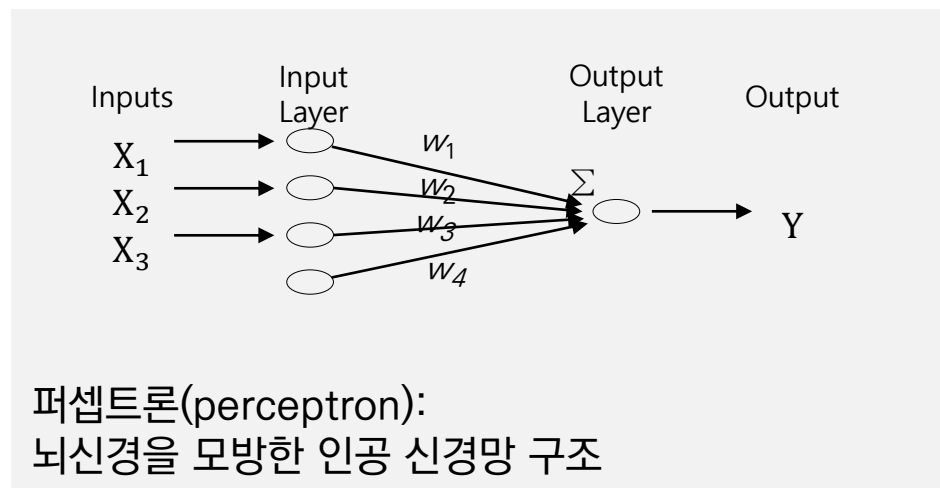
1950년대, 퍼셉트론(perceptron) 모형 개발

- Frank Rosenblatt (코넬대 심리학자)
- 어느 정도 성공적이었지만 일반적인 방법으로는 실망스러운 수준
- 당시 컴퓨터의 성능의 한계

1968년, 신경망의 이론적 한계

- Papert와 Minsky (MIT)
- 단순한 신경망이 비선형 문제를 해결하지 못함을 지적함

1970년대까지 침체기



1982년, 역전파 (back propagation) 알고리즘

- John Hopfield (물리학자)
- 다층 신경망의 학습 방법, 이전 방법의 이론적 함정들을 극복
- 인공신경망의 르네상스

1980년대, 인공신경망의 상업적 사용 확대

- 사기성 신용카드 거래 인식, 수표 금액 인식 등 금융 분야에 활용
- 특정 영역에서 전문가와 같이 문제 해결이 가능한 전문가 시스템 (Experts System) 이 상업적으로 성공

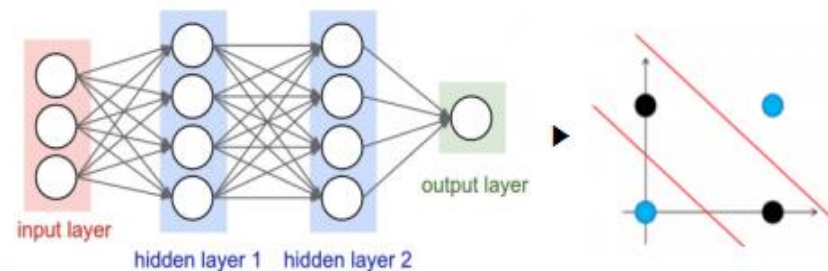
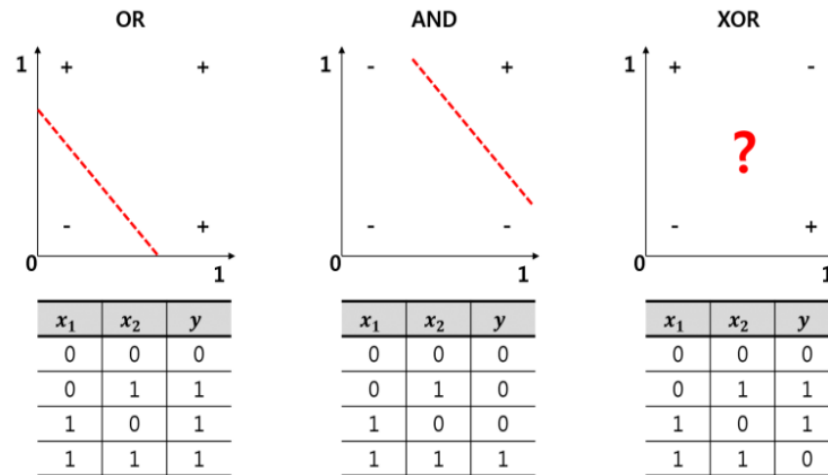
1990년대 암흑기

- 연산처리 속도의 한계
- 전문가 시스템의 용도도 제한적이며 유지 관리 어려움

2006년, 딥러닝 (Deep Learning) 기술의 실용화

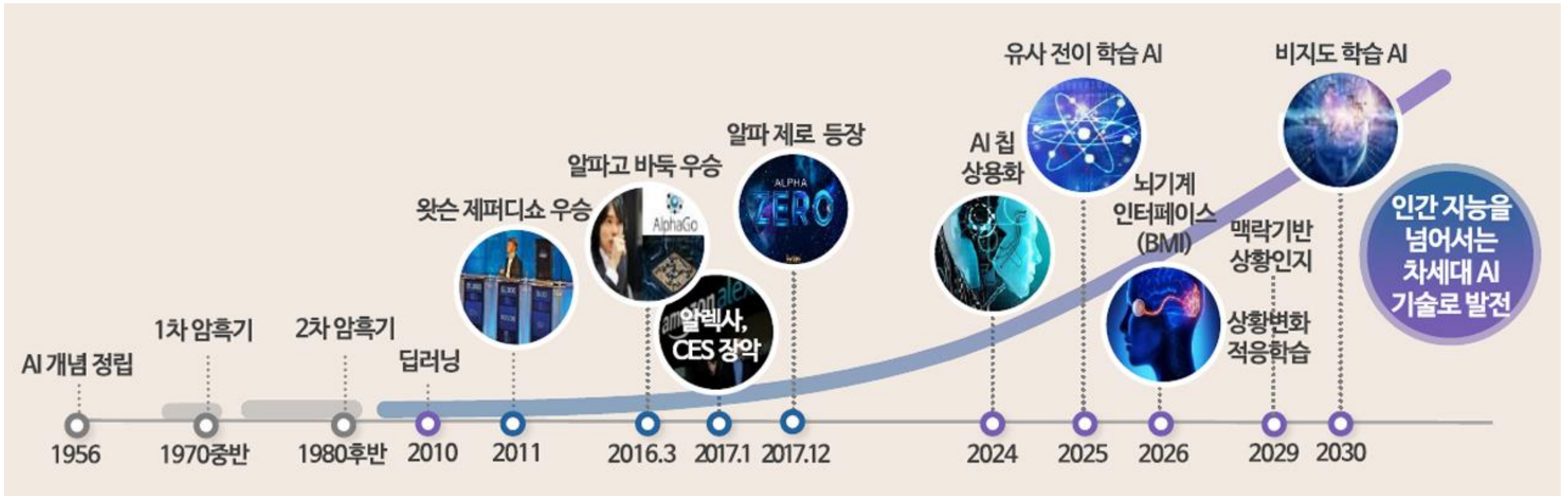
- 하드웨어의 발전, 빅데이터 등으로 최적의 알고리즘 개발
- 이미지 인식, 음성 인식, 번역 등에서 활발하게 활용

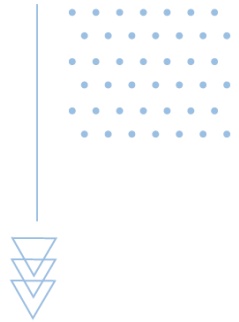
* Perceptron의 한계라고 할 수 있는 XOR 문제를 Backpropagation (역전파) 알고리즘으로 해결



인공신경망의 발전 흐름

- Perceptron의 한계 : XOR 문제 (Marvin Minsky & Seymour Papert 1969)
- **1차 암흑기** (1960년대 말 - 1970년대)
- Backpropagation (역전파) 알고리즘으로 해결 (Paul Webros 1974, 1982, Geoffrey Hinton 1986)
- 르네상스 (1980년대, Expert System)
- **2차 암흑기** (1980년대 후반 - 1990년대 초)

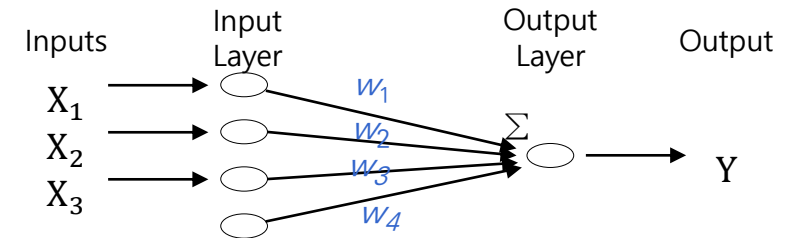




II. 인공신경망 기법원리: 역전파 기법

역전파 (back propagation) 알고리즘

- Hopfield에 의해 개발 (1982)
- 인공신경망에서 가장 많이 널리 사용되는 학습 기법
- 오차(실제값과 예측값의 차이)의 역전파를 통해 입력과 출력간의 관계를 학습하는 기법
- 학습의 핵심 과제는 **연결의 가중치**를 정하는 것
 - 입력 값을 이용하여 출력 예측값을 구하고 실제값과 비교하여 오차 추정
 - 오차를 이용하여 가중치를 조정
 - 오차가 허용 가능한 범위 안으로 작아질 때까지 이 단계를 반복
- 한 단계에서 가중치를 수정하는 비율 (학습률)은 적절하게 관리해야 함
 - 모델을 과하게 수정하지 않도록 함

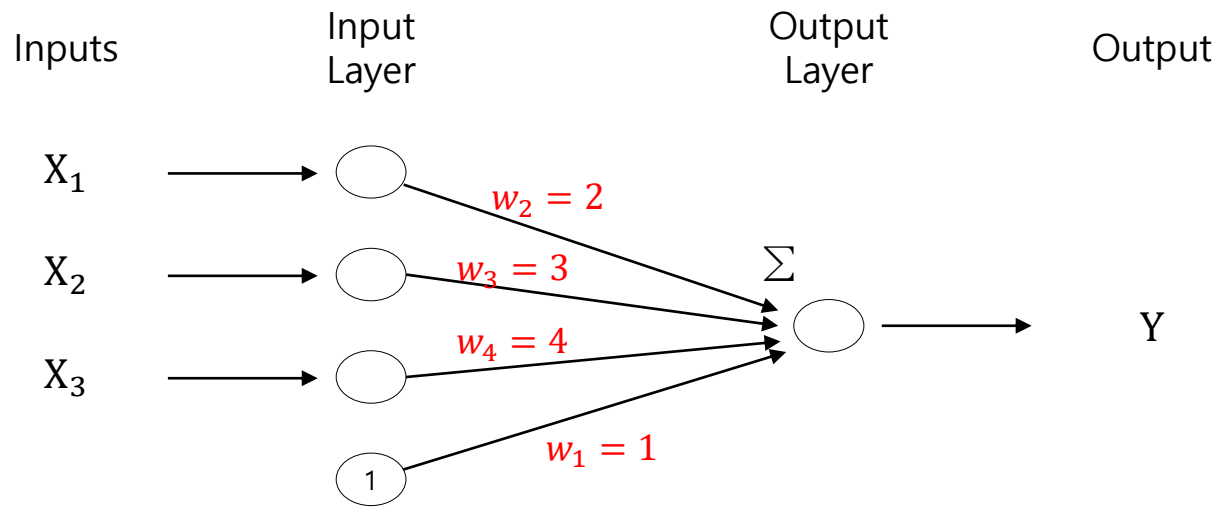


■ 단계 1: 기본 모델 설정 - 입력과 출력 개수, 활성화함수, 은닉 계층과 노드의 수 결정

- 예. 3개의 수치형 입력 데이터(X_1, X_2, X_3)와 1개의 수치형 출력(Y)

■ 단계 2: 초기화

- 예. 4개의 연결에 대한 초기 가중치를 1, 2, 3, 4로 설정함



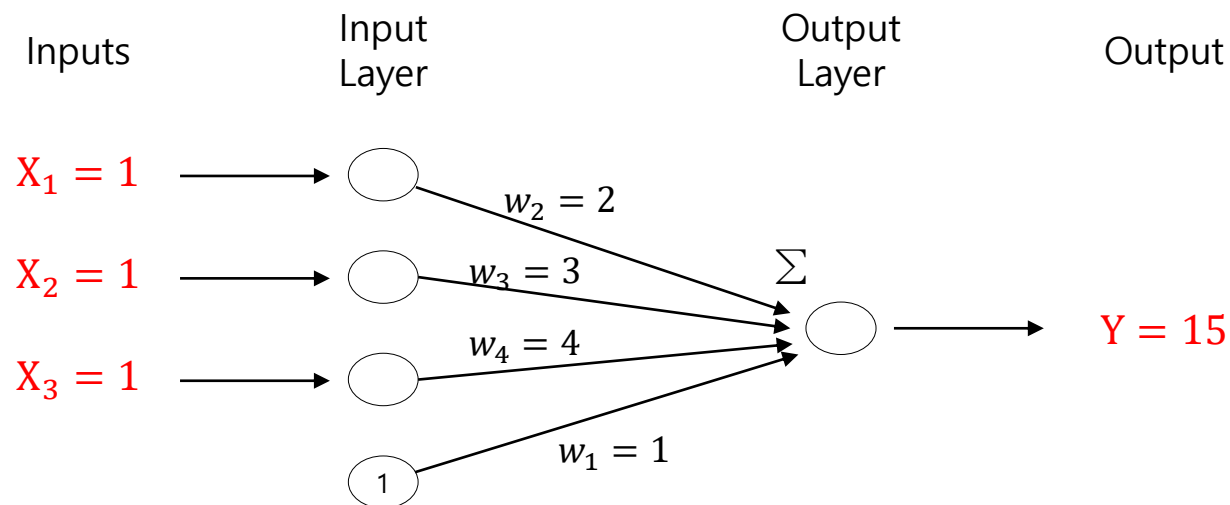
■ 단계 3: 오차 계산

- 오차는 학습용 데이터의 실제값과 모델의 예측값과의 차이를 나타냄

$$e = Y - \overline{Y}$$

- 예. 학습용 데이터의 첫번째 입력값이 모두 1이고 실제 출력값이 15인 경우
출력 예측값과 오차는 다음과 같이 계산됨

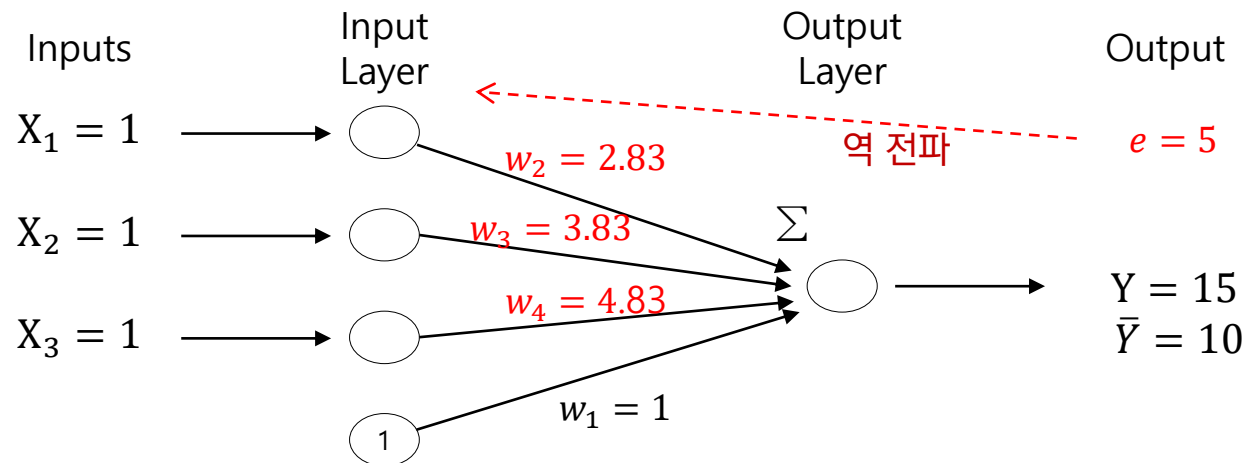
$$\overline{Y} = 1 + 2 \times 1 + 3 \times 1 + 4 \times 1 = 10 \quad \text{오차} = 15 - 10 = 5$$



❏ 단계 4: 가중치 (weight) 조정

* 학습: 최선의 가중치를 결정하는 과정

- 오차는 출력 노드로부터 반대 방향의 다른 모든 노드로 역전파
- 오차의 일정 비율만큼 연결의 가중치들을 조정
 - 오차에 적용되는 조정 비율 λ 를 “학습률” (learning rate) 이라함
 - 0 과 1 사이의 값: 0에 가까울수록 변화가 적음
- 새로운 가중치 $\omega = \omega' + \lambda \times e$
 - ω' : 이전 가중치
 - e : 출력 노드로부터 전달된 오차 (앞 계층의 노드에 동일하게 나누어줌)



(예) $\omega_2' = 2$, $\omega_3' = 3$, $\omega_4' = 4$, $\lambda = 0.5$

$$\omega_2 = \omega_2' + \lambda \times e/3 = 2 + 0.5 \times 5/3 = 2.83$$

$$\omega_3 = \omega_3' + \lambda \times e/3 = 3 + 0.5 \times 5/3 = 3.83$$

$$\omega_4 = \omega_4' + \lambda \times e/3 = 4 + 0.5 \times 5/3 = 4.83$$

■ 단계 5: 다른 레코드에 대해 반복

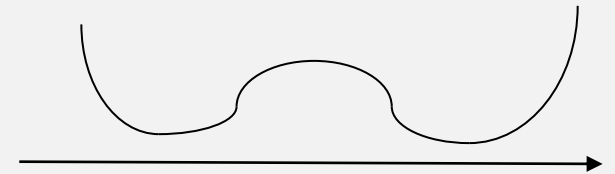
- 모든 학습용 레코드에 대해서 단계 3, 4를 반복

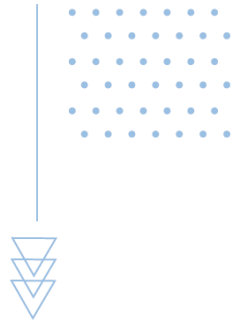
■ 단계 6: 다음의 정지 조건이 만족될 때까지 단계 3~5를 반복

- 가중치의 변화량이 매우 작을 때
- 정해진 반복횟수에 도달했을 때
- 시험용 데이터에 대한 성능이 나빠질 때

역전파의 문제점: 국소 최적값(local optimum)으로 수렴할 수 있음

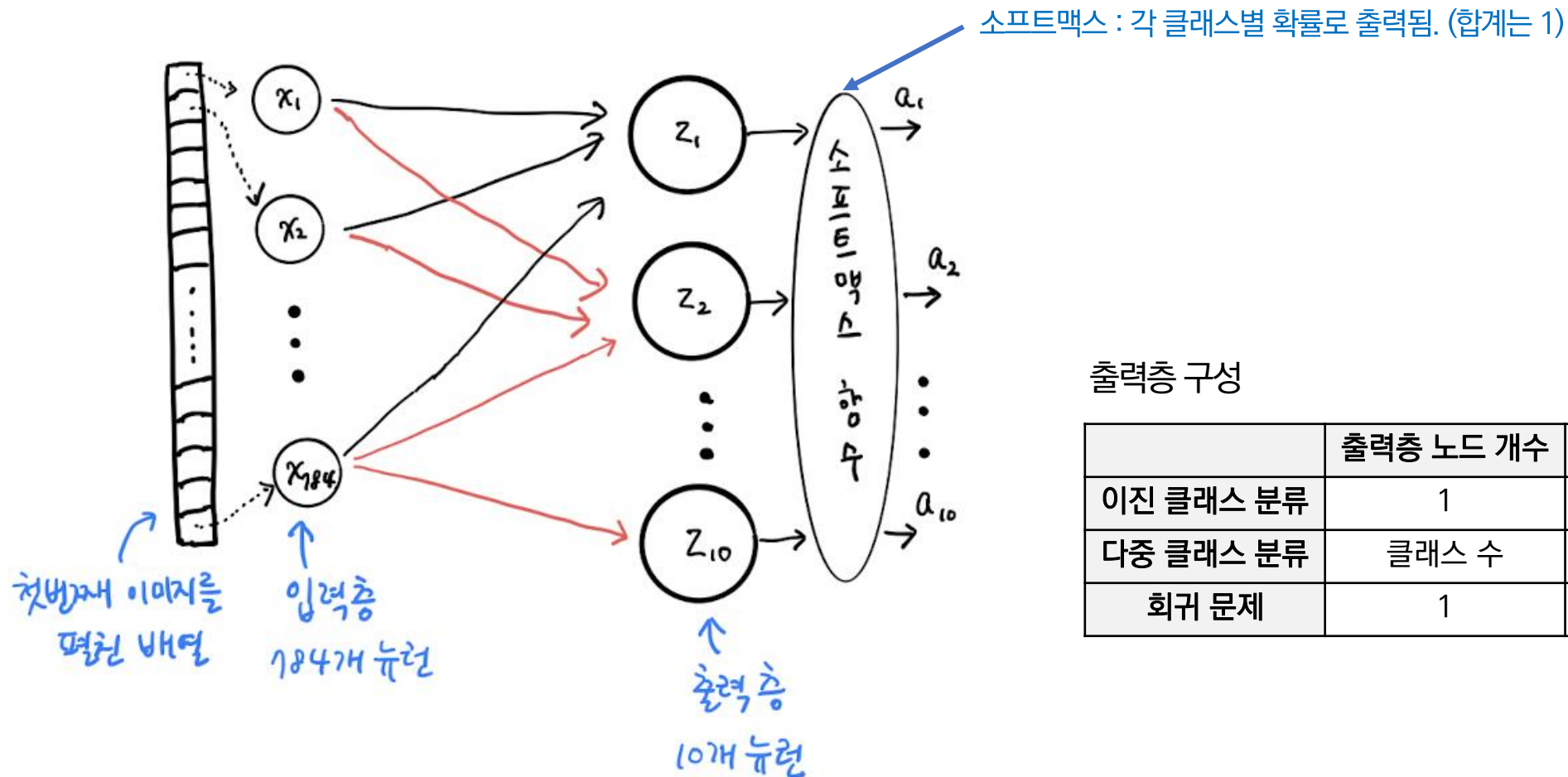
- 신경망이 학습용 데이터에 대하여 좋은 결과들을 내고, 가중치의 조정이 더 이상 신경망의 성능을 향상시키지 않는 경우
- 하지만, 더 좋은 결과를 제공할 수 있는 가중치의 조합이 따로 존재하는 경우





III. 텐서플로우로 ANN구현

입력층과 출력층으로만 구성된 NN을 텐서플로우로 구현



출력층 구성

	출력층 노드 개수	활성화 함수
이진 클래스 분류	1	Sigmoid
다중 클래스 분류	클래스 수	Softmax
회귀 문제	1	X

1. 데이터 준비

- 데이터 불러오기 (Keras에서 Fashion-MNIST dataset을 import)
 - 10개의 패션 상품 이미지(28*28 크기의 회색조 이미지) 데이터: 학습데이터 60,000개, 테스트 데이터 10,000개

```
from tensorflow import keras
(train_input, train_target), (test_input, test_target) = keras.datasets.fashion_mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
32768/29515 [=====] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26427392/26421880 [=====] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
8192/5148 [=====] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4423680/4422102 [=====] - 0s 0us/step
```

```
print(train_input.shape, train_target.shape)
```

```
(60000, 28, 28) (60000,)
```

```
print(test_input.shape, test_target.shape)
```

```
(10000, 28, 28) (10000,)
```

● 데이터의 이해

– 학습데이터 10개의 input값 그림으로 출력해보기

```
import matplotlib.pyplot as plt
fig, axs = plt.subplots(1,10, figsize = (10,10))
for i in range(10):
    axs[i].imshow(train_input[i], cmap = 'gray_r')
    axs[i].axis('off')
plt.show()
```



– 학습데이터의 target값 확인

```
print([train_target[i] for i in range(10)])
```

```
[9, 0, 0, 3, 0, 2, 7, 2, 5, 5]
```

```
class_names = ['티셔츠', '바지', '폴오버', '드레스', '코트', '샌달', '셔츠', '스니커즈', '가방', '부츠']
```

```
for i in range(10):
    print(class_names[train_target[i]], end = ",")
```

```
부츠,티셔츠,티셔츠,드레스,티셔츠,폴오버,스니커즈,폴오버,샌달,샌달,
```

– 학습데이터의 target class종류 및 수

```
import numpy as np
```

```
print(np.unique(train_target, return_counts=True))
```

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint8),
```

```
array([6000, 6000, 6000, 6000, 6000, 6000, 6000, 6000, 6000, 6000]))
```

● 데이터 정제

- 입력값의 scale 맞추기(0~1), 구조 변환 (이미지 데이터 2차원을 1차원으로 구성)
- target 값은 class 번호로 이미 1차원이므로 변환 불필요

```
train_scaled = train_input/255.0 # 학습데이터 입력값을 0~1사이로 변환
train_scaled = train_scaled.reshape(-1,28*28) # 2차원을 1차원으로 펼쳐서 입력값으로 사용
test_scaled = test_input/255.0
test_scaled = test_scaled.reshape(-1,28*28)
```

2. 신경망 구성

● 신경망의 구조 및 파라미터 설정

- 밀집층(Dense) 만들기, 밀집층 쌓기(Sequential), 컴파일하기(Compile)

```
dense1=keras.layers.Dense(노드 수, activation = '활성함수명', input_shape=(입력크기)) # 밀집층 원하는 만큼 구성 dense1, dense2, ...
model = keras.Sequential([dense1, dense2,...])
model.compile(loss = '손실함수명', metrics = '성능평가 기준')
```

```
## . 입력과 출력층만 갖는 신경망 구조 생성 (분류 10개 클래스)
dense = keras.layers.Dense(10, activation = 'softmax', input_shape=(784,))
model = keras.Sequential([dense]) # 생성된 dense층 추가
model.compile(loss = 'sparse_categorical_crossentropy', metrics = 'accuracy')
```

3. 학습 데이터로 학습

- 구성된 신경망 모델을 학습데이터로 훈련

- model명.fit(학습 입력 데이터, 학습 target 데이터, epochs = 수)

```
## 위에서 구성된 신경망을 데이터를 사용해서 훈련
```

```
model.fit(train_scaled, train_target, epochs = 5)
```

```
Epoch 1/5
```

```
1875/1875 [=====] - 3s 1ms/step - loss: 0.5916 - accuracy: 0.7995
```

```
Epoch 2/5
```

```
1875/1875 [=====] - 2s 1ms/step - loss: 0.4705 - accuracy: 0.8412
```

```
Epoch 3/5
```

```
1875/1875 [=====] - 2s 1ms/step - loss: 0.4514 - accuracy: 0.8505
```

```
Epoch 4/5
```

```
1875/1875 [=====] - 2s 1ms/step - loss: 0.4419 - accuracy: 0.8543
```

```
Epoch 5/5
```

```
1875/1875 [=====] - 2s 1ms/step - loss: 0.4362 - accuracy: 0.8565
```

```
<tensorflow.python.keras.callbacks.History at 0x7fd8c51d6090>
```

4. 예측 및 모델 평가

● 테스트 데이터로 결과 예측 및 성능 평가

```
예측값 = model.predict(테스트 데이터 입력값) #scikitlean과 달리 각 클래스에 대한 확률 반환  
예측 성능 = model.evaluate(테스트 데이터 입력값)
```

● 전체 성능 평가: 10,000개의 테스트 데이터에 적용 시 성능

```
# #. 성능 평가하기  
model.evaluate(test_scaled, test_target)
```

```
313/313 [=====] - 0s 1ms/step - loss: 0.4787 - accuracy: 0.8439
```

```
[0.47873756289482117, 0.8439000248908997] ← loss값과 metrics에서 지정한 성능 평가 결과 출력
```

● 각 사례별 세부 예측값

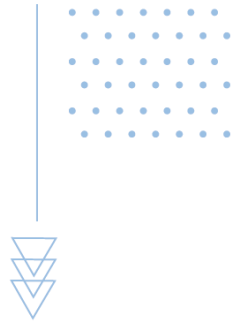
- Predict함수는 클래스 예측시 모든 클래스들에 대한 예측 확률을 반환하며, 가장 큰 확률을 갖는 클래스가 예측 결과임
(* Batch_size: 한번에 몇 개씩 처리할 지로 생략 가능)

```
# 세부 예측결과 확인  
pred_detail = model.predict(test_scaled, batch_size=1) # 세부값 예측. 각 클래스별 확률로 출력됨.
```

```
pred_index = np.argmax(pred_detail[0]) #가장 큰 인덱스 반환  
print(class_names[pred_index]) # 첫번째 테스트데이터 예측결과 출력
```

* print(class_names[test_target[0]])로 실제 label 도 확인해보면 부츠임

부츠



IV. 정리

인공신경망 모델은 다양한 분야에서 사용되고 있으나 설명하기 어려움

- 주로 사기 탐지와 같은 분류 문제에 많이 사용되며, 입력과 출력 간의 고차원 비선형 관계를 다루는 데에 사용됨
- 입력과 출력 사이에 은닉 계층이 있어 입력과 출력의 관계에 대한 해석이 어려움

최적의 인공신경망 모델을 구하는데 시간 필요

- 모델의 파라미터를 다양하게 조정하면서 최적값을 찾는데 오랜 시간이 소요됨
- 최종적으로 얻은 가중치들이 최적(optimal)이라고 보장하지는 못함

설명하기는 어렵지만 분류 속도는 빠름

- 모델 구축 후 레코드 분류 속도는 매우 빠름
- 실시간의 빠른 반응시간이 필요한 경우 인공 신경망을 활용하기 좋음

인공지능 (Artificial Intelligence)

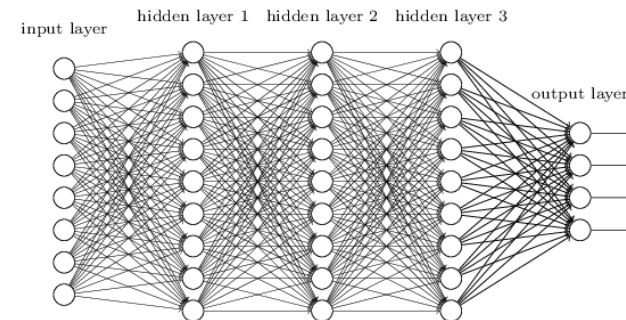
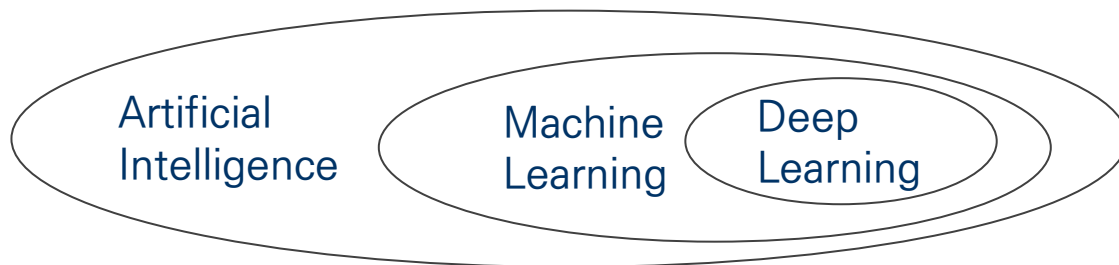
- 인간의 지적 능력(인지, 추론, 학습 등)을 컴퓨터를 이용해 구현하는 지능
 - John McCarthy가 Dartmouth 대학 학술회의에서 처음으로 AI 용어 사용 (1956)
 - Natural Language Processing, Speech Recognition, Computer Vision, Robots 등
- 인공지능의 구분
 - Weak AI: 학습을 통해 특정한 문제를 해결
 - Strong AI: 인간의 사고와 비슷하거나 더 뛰어난 지능을 나타내며 스스로 사고하고 행동

기계학습 (Machine Learning)

- 컴퓨터가 스스로 학습하여 문제를 해결할 수 있도록 구현하는 AI의 한 분야
- 데이터와 정답을 주고 규칙을 찾아내도록 함
 - 체커 게임: 스스로 학습하는 최초의 프로그램, 처음으로 ML 용어 사용 (Arthur Samuel 1959)

딥러닝 (Deep Learning)

- 심층 **신경망** (Deep **Neural Network**) 을 통해 학습하는 기법
- 딥러닝의 등장으로 산업과 사회 전반에 걸쳐 혁신적으로 진화함
 - 활성화 함수, 가중치 초기화 방법, 최적화 방법 개발



□ 은닉층을 한 개 가지는 NN을 텐서플로우로 구현

```
from tensorflow import keras

(train_input, train_target), (test_input, test_target) = keras.datasets.fashion_mnist.load_data()
```

1. 데이터 입력 준비

```
train_scaled = train_input/255.0
test_scaled = test_input/255.0
train_scaled = train_scaled.reshape(-1, 28*28)
test_scaled = test_scaled.reshape(-1, 28*28)
```

2. 신경망 구성

```
dense1 = keras.layers.Dense(100, activation = 'relu', input_shape = (784,))
dense2 = keras.layers.Dense(10, activation = 'softmax')
model = keras.Sequential([dense1, dense2])
model.compile(loss = 'sparse_categorical_crossentropy', metrics = 'accuracy')
```

← 은닉층 한 개 만들고 쌓은 것만 달라짐

3. 신경망 학습

```
model.fit(train_scaled, train_target, epochs = 5)
```

#4. 성능평가

```
model.evaluate(test_scaled, test_target)
```

```
Epoch 5/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.3197 - accuracy: 0.8871
313/313 [=====] - 0s 1ms/step - loss: 0.3771 - accuracy: 0.8747
[0.3770688772201538, 0.8747000098228455]
```

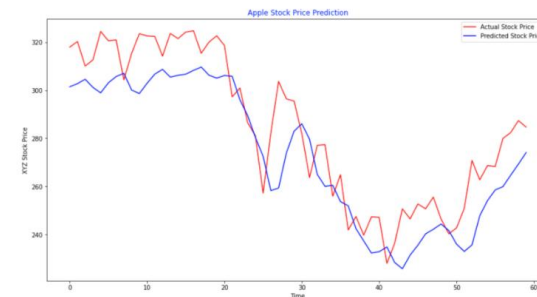
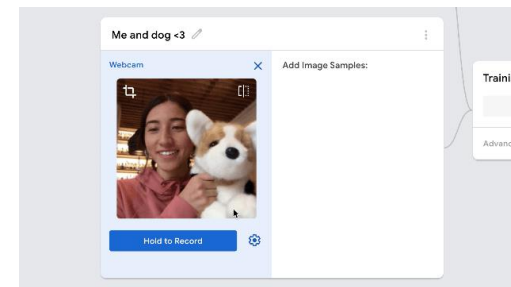
← 학습데이터 성능

← 테스트 데이터 성능

→ 앞의 0.84~대비 정확도 개선

■ 딥러닝의 다변화

- **CNN**: DNN의 Dense층 부분에 1개 이상의 **Convolution층(합성곱)**을 사용
 - 이미지 처리에 특화 (e.g., 얼굴인식, 동작 인식 등)
 - 구글 티처블 머신: 안면 인식: <https://teachablemachine.withgoogle.com/>
- **RNN**: DNN의 Dense층 부분에 1개 이상의 **Recurrent층(순환층)**을 사용
 - 시계열 처리에 특화 (e.g., 주식예측, 음성인식, 텍스트 분석 등)
 - 순서가 중요한 데이터의 분석에 사용됨
- **GAN**: DNN의 Dense층 부분에 1개 이상의 **Generative(생성층)**과 **Adversarial(적대층)**을 사용
 - 창작, 만들기에 특화 (e.g., 가상의 인물 만들기, 유명화가 모방작 만들기)



■ (Advanced) CNN 구현 예제

```
from tensorflow import keras
# 1. 데이터 입력 준비(패션 Minst)
(train_input, train_target),(test_input, test_target) = keras.datasets.fashion_mnist.load_data()
train_scaled = train_input/255.0
test_scaled = test_input/255.0
train_scaled = train_scaled.reshape(-1, 28,28,1) # -1은 데이터 수, 뒤의 (28,28,1)이 각 그림의 크기임
test_scaled = test_scaled.reshape(-1, 28,28, 1)

# 2. 신경망 구성
convol = keras.layers.Conv2D(32, kernel_size = 3, activation = 'relu', padding='same', input_shape = (28,28,1)) #2차원 모양을 그대로 유지, 흑백이라 1
pooling = keras.layers.MaxPooling2D(2)
flatten = keras.layers.Flatten()
dense = keras.layers.Dense(10, activation = 'softmax') # 출력층 - 밀집층으로 이전과 동일
model = keras.Sequential([convol, pooling, flatten, dense]) #CNN 또는 dense층을 상황에 따라서 추가하면 됨.
model.compile(loss = 'sparse_categorical_crossentropy', metrics = 'accuracy')

model.summary() #신경망 구성 확인
#####이후 완전 동일#####/
#과대적합 방지 - 콜백
checkpoint_cb = keras.callbacks.ModelCheckpoint('best-model.h5')
early_stopping_cb = keras.callbacks.EarlyStopping(patience = 2, restore_best_weights = True)
#####
# 3. 신경망 학습
model.fit(train_scaled, train_target, epochs = 30, validation_data = (test_scaled, test_target), callbacks = [checkpoint_cb, early_stopping_cb])

#4. 성능평가
model.evaluate(test_scaled, test_target)
```

1차원 변환 안함

Convolution층
Pooling층
1차원 펼치기

← 층 쌓기

Model: "sequential_1"

Layer (type)	Output Shape	Param #
=====		
conv2d_1 (Conv2D)	(None, 28, 28, 32)	320
=====		
max_pooling2d_1 (MaxPooling2D)	(None, 14, 14, 32)	0
=====		
flatten (Flatten)	(None, 6272)	0
=====		
dense_1 (Dense)	(None, 10)	62730
=====		
Total params: 63,050		
Trainable params: 63,050		
Non-trainable params: 0		

Deep Learning 의 개척자들

- Yann LeCun (NYU, Facebook)
- Geoffrey Hinton (Univ of Toronto, Google)
- Yoshua Bengio (Univ of Montreal)



2018 ACM A.M. Turing Award
<https://awards.acm.org/about/2018-turing>

인공지능의 온라인 교육과 자연어 처리에 기여

- Andrew Ng (Stanford Univ, Baidu 2014~2018)
 - Co-founder of Coursera, Machine Learning course “democratize” deep learning





Never Never give up!