



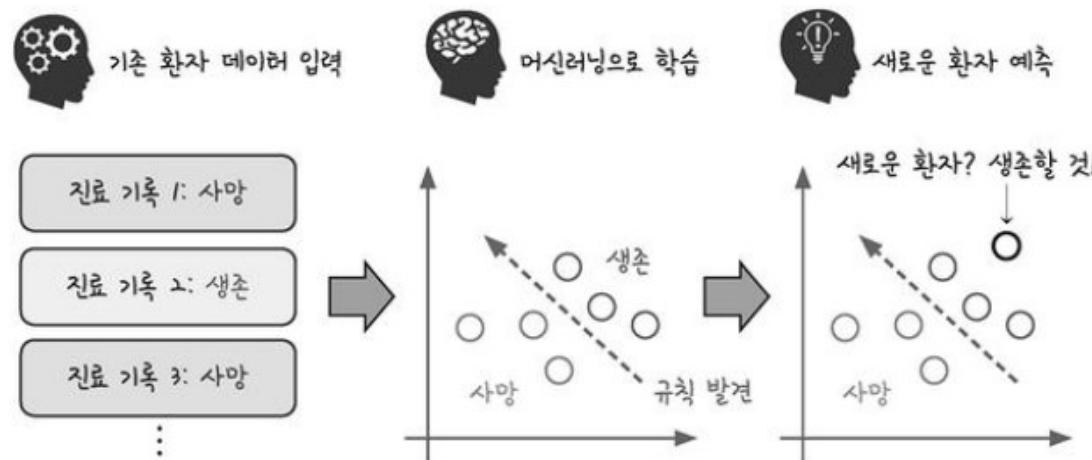
Session 1

인공 지능 머신 러닝 딥 러닝



➤ 머신러닝

- 머신러닝은 데이터 안에서 규칙을 발견하고 그 규칙을 새로운 데이터에 적용해서 새로운 결과를 도출
- 기존 데이터를 이용해 아직 일어나지 않은 미지의 일을 예측
- 데이터가 입력되고 패턴이 분석되는 과정을 학습(training)이라고 부릅니다.



딥 러닝

➤ 딥러닝(심층학습, 강화학습)

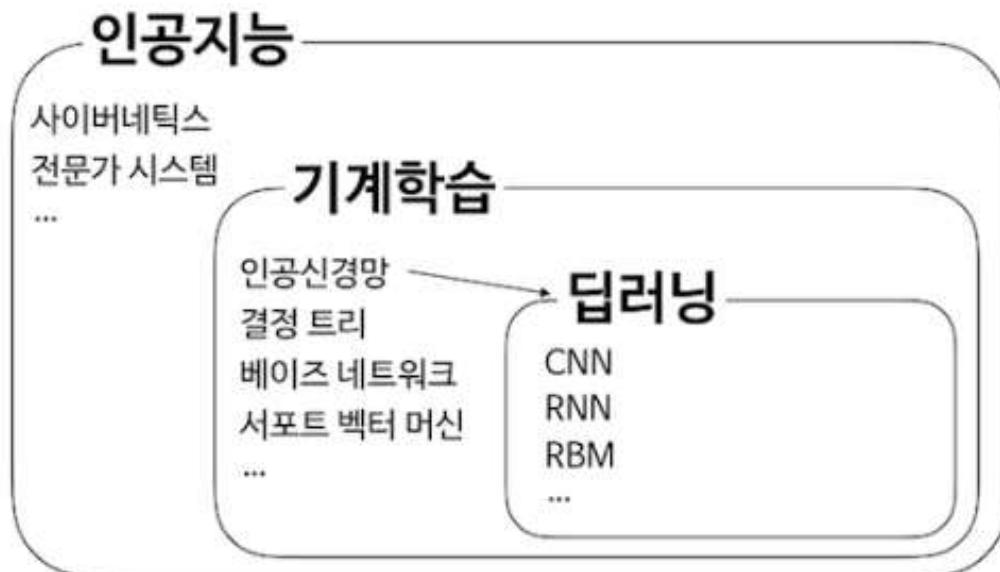
- 여러 층을 가진 인공신경망을 이용하여 머신러닝을 수행하는 것
- 인공신경망을 넓고 다층(Deep)으로 쌓으면 딥러닝(Deep-Learning)
- 신경망을 사용해 수많은 데이터 속에서 어떤 패턴을 파악하고 이것을 통해 인간이 사물을 구분하듯이 컴퓨터가 데이터를 나누는 것 (사물이나 데이터를 군집화하거나 분류할 때 사용)
- 데이터 자체를 컴퓨터에게 전달, 인공신경망 구조를 사용하여 데이터 전체를 학습시킨다 (인간의 뇌에서 일어나는 의사결정 과정을 모방한 인공신경망(Artificial Neural Network) 구조를 통해서 학습한다)
- 예] 컴퓨터에 개와 고양이 사진을 학습시켜 특정사진의 동물이 개인지 고양이인지 분류
(일반적인 기계학습의 경우 개와 고양이의 구별되는 큰 특징들만 뽑아 컴퓨터에게 전달)
딥러닝은 개, 고양이 사진 자체를 컴퓨터가 학습하도록 하는 것



딥 러닝

➤ 인공 지능과 머신 러닝, 딥 러닝의 차이점

- 딥러닝 단계에서 군집화된 데이터를 분류 → 머신러닝 단계를 거쳐 학습 → 인공지능의 단계에서는 어떤 판단이나 결과를 내는 과정



딥 러닝

➤ 인공 신경망

- 인공 신경망(딥러닝)이 2012년 ILSVRC2012 대회에서 깊게 쌓은 딥러닝 모델 AlexNet이 우승하면서 다시 주목 받게 된 계기
- 빅데이터와 컴퓨터 하드웨어 성능과 Matrix 연산에 고성능인 GPU로 인해 신경망을 학습을 가속화
- 신경망은 다른 머신러닝 알고리즘보다 규모가 크고 복잡한 문제에서 성능이 좋다
- 1990년대 이후 크게 발전된 상대적으로 짧은 시간 안에 대규모의 신경망을 학습시킬 수 있게 되었다.
- 최초의 인공 뉴런 - 하나 이상의 이진(on/off) 입력과 하나 이상의 출력을 가진다.

딥 러닝

➤ 딥러닝

- 기본 층을 겹겹이 쌓아 올려 구성한 신경망(neural network)이라는 모델을 사용하여 표현 층을 학습
- 층 기반 표현 학습(layered representations learning) 또는 계층적 표현 학습(hierarchical representations learning)

➤ 깊은 신경망(Deep Neural Network : DNN)

- 신경망을 3개 이상 중첩

환경설정 및 라이브러리

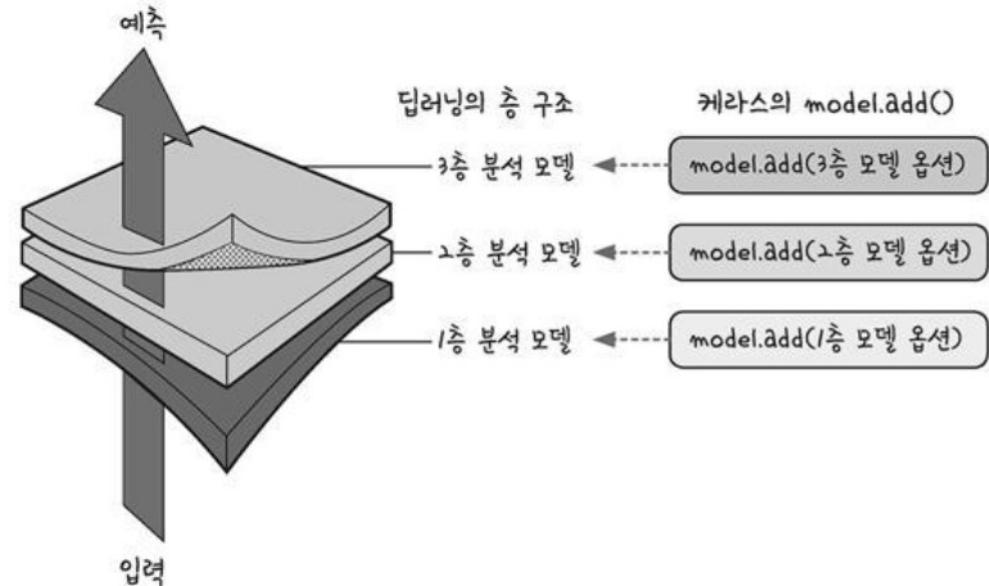
➤ 딥러닝 라이브러리 Keras

- 케라스(keras)를 사용해 딥러닝을 실행시킵니다.
- 케라스가 구동되려면 텐서플로(TensorFlow) 또는 씨아노(theano)라는 두 가지 라이브러리 중 하나가 미리 설치되어 있어야 합니다
- 머신러닝 라이브러리 Theano와 TensorFlow를 래핑한 라이브러리
- <https://keras.io/>
- Sequential로 딥러닝의 각 층을 add()로 추가
- 활성화 함수, Dropout 등 add()로 간단하게 추가
- compile()로 모델 구축
- loss로 최적화 함수 지정
- fit()로 모델에 데이터를 학습시킴
- Keras로 머신러닝을 수행할 때 Numpy 배열 데이터를 전달해야 한다

환경설정 및 라이브러리

➤ keras 라이브러리 - Sequential() 와 Dense()

- Sequential() - 딥러닝의 구조를 한 층 한 층 쉽게 쌓아올릴 수 있게 해 줍니다.
- Sequential 함수를 선언하고 나서 model.add() 함수를 사용해 필요한 층을 차례로 추가합니다.
- Dense(activation=, loss=, optimizer=) 각 층이 각각 어떤 특성을 가질지 옵션을 설정하는 역할을 합니다.
 - activation: 다음 층으로 어떻게 값을 넘길지 결정하는 부분(가장 많이 사용되는 relu와 sigmoid 함수)
 - loss: 한 번 신경망이 실행될 때마다 오차 값을 추적하는 함수
 - optimizer: 오차를 어떻게 줄여 나갈지 정하는 함수
- model.evaluate() - 딥러닝의 모델이 어느 정도 정확하게 예측하는지를 점검

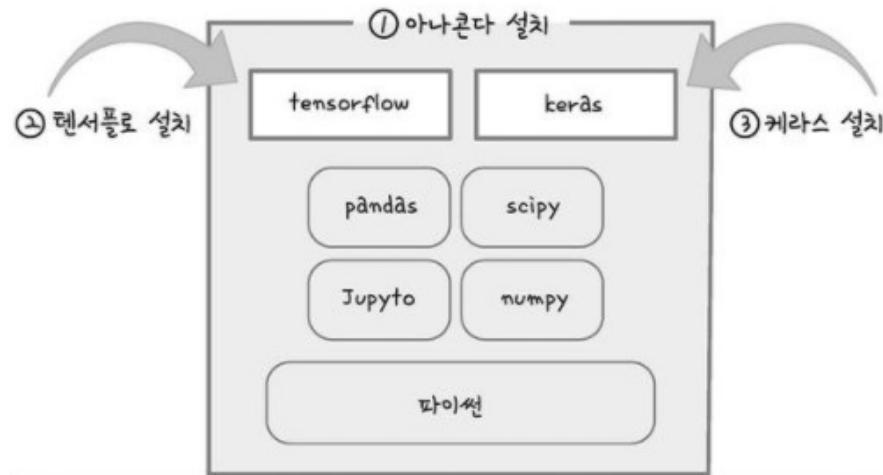


인공 지능과 머신 러닝, 딥 러닝의 차이점

➤ TensorFlow

- 구글이 공개한 대규모 숫자 계산을 해주는 머신러닝 및 딥러닝 전문 라이브러리
- Tensor는 다차원 행렬 계산을 의미
- 상업적인 용도로 사용할 수 있는 오픈소스(Apache 2.0)
- C++로 만들어진 라이브러리
- 파이썬을 사용해서 호출할 때 오버헤드가 거의 없는 구조로 설계
- <https://www.tensorflow.org>
- 이미지 처리와 음향 처리 등을 할 때는 추가적으로 이미지 처리에 특화된 OpenCV 라이브러리등과 함께 사용

➤ 딥러닝을 구동하기 위한 세가지 설치 작업

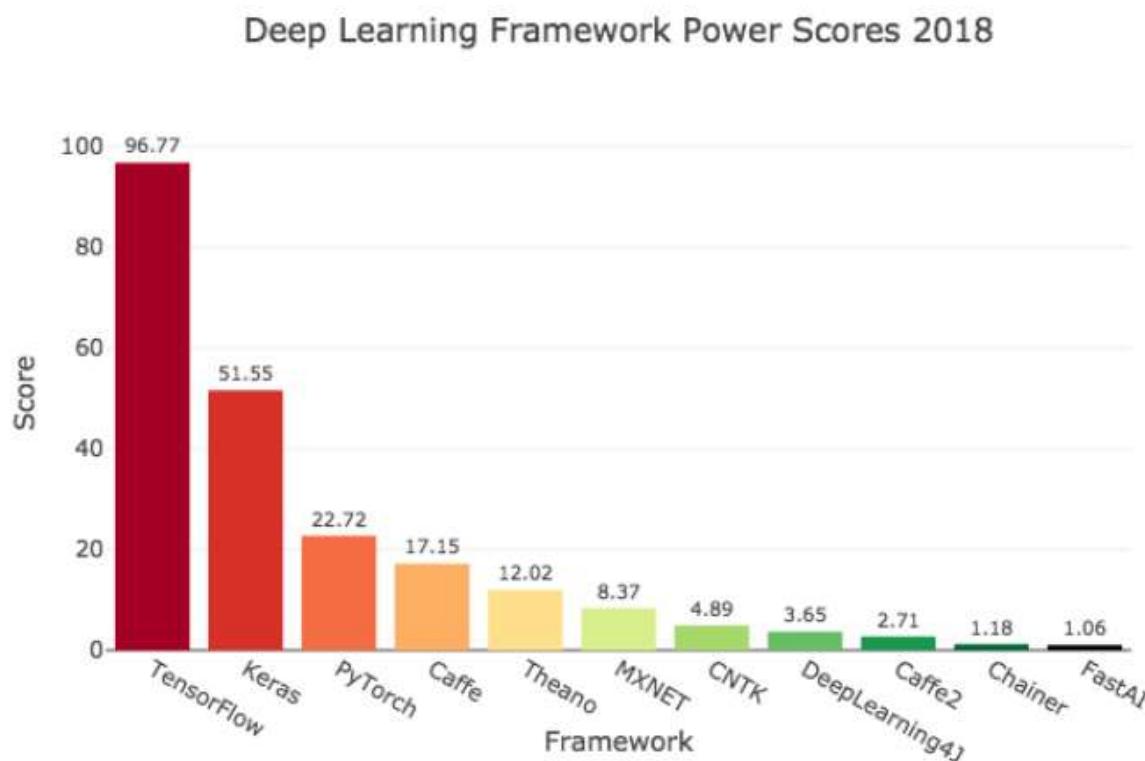


conda update -n base conda
conda update --all

pip install tensorflow
pip install keras
pip install --upgrade tensorflow
pip3 install --upgrade tensorflow
conda install tensorflow

```
import tensorflow as tf  
print(tf.__version__)
```

➤ 딥러닝 라이브러리 순위



인공 지능과 머신 러닝, 딥 러닝의 차이점

➤ Keras이용 BMI 판정

```
from keras.models import Sequential  
from keras.layers.core import Dense, Dropout, Activation  
from keras.callbacks import EarlyStopping  
import pandas as pd, numpy as np  
# BMI 데이터를 읽어 들이고 정규화하기  
csv = pd.read_csv("bmi.csv")
```

인공 지능과 머신 러닝, 딥 러닝의 차이점

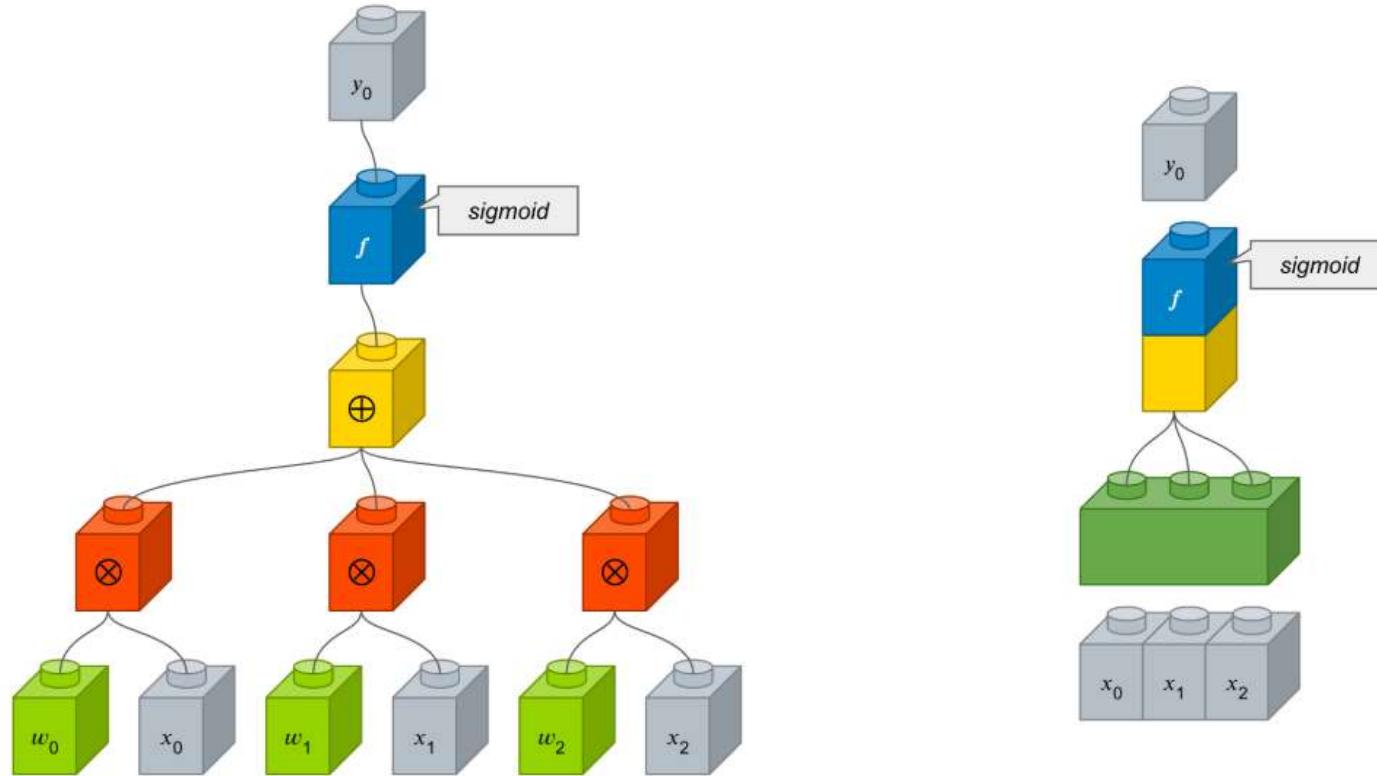
➤ Keras이용 BMI 판정

- weight decay(가중치 감소) - 학습중 가중치가 큰 것에 대해서 패널티를 부과해 과적합의 위험을 줄이는 방법
- Dropout - 복잡한 신경망에서 가중치 감소만으로 과적합을 피하기 어려운 경우 뉴런의 연결을 임의로 삭제시켜 신호를 전달하지 못하도록 하는 방법
- softmax 회귀 - 입력받은 값을 출력으로 0~1사이의 값으로 모두 정규화하여 출력값들의 총합은 항상 1이 되는 특성의 함수
- 분류하고 싶은 클래스 수 만큼 출력으로 구성
- 소프트 맥스 결과값을 One hot encoder의 입력으로 연결하면 가장 큰 값만 True값, 나머지는 False값이 나오게 하여 이용 가능하다
- val_loss는 에포크 횟수가 많아질 수록 감소하다가 다시 증가됨을 보이는 경우, 과적합이 발생한 것입니다.
- 학습에 더 이상 개선의 여지가 없을 경우 학습을 종료시키는 콜백함수(수행중인 함수에서 지정된 함수를 호출, 되부름)는 EarlyStopping 입니다.
- Dense(출력 뉴런의 수, input_dim=입력 뉴런의 수, init= 가중치 초기화 방법, activation=활성화 함수)
- relu 활성화 함수는 은닉층에 주로 사용
- sigmoid 활성화 함수는 이진 분류 문제에서 출력층에 주로 사용
- softmax 활성화 함수는 다중 클래스 분류 문제에서 출력층에 주로 사용

인공 지능과 머신 러닝, 딥 러닝의 차이점

➤ Keras

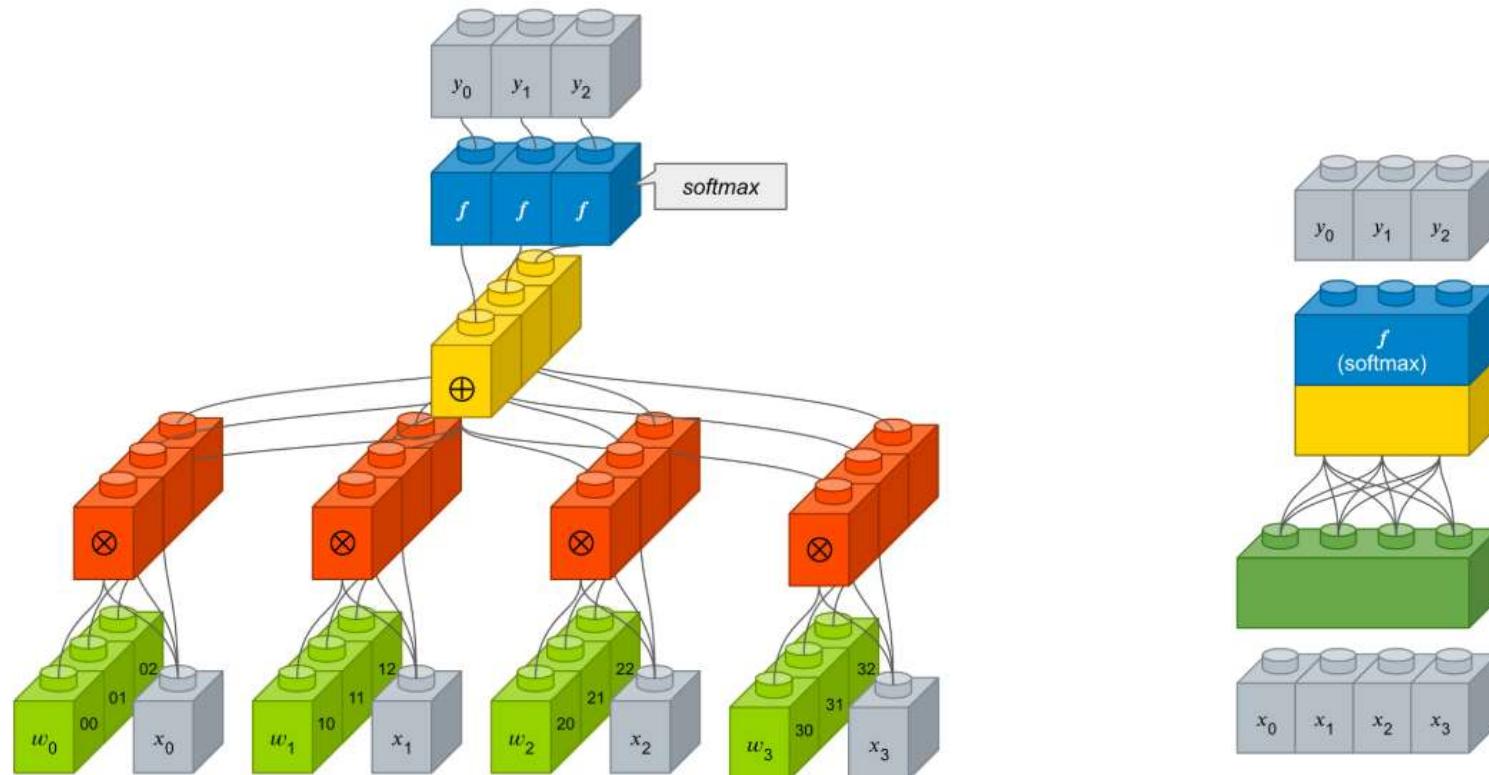
```
Dense(1, input_dim=3, activation='sigmoid')
```



인공 지능과 머신 러닝, 딥 러닝의 차이점

➤ Keras

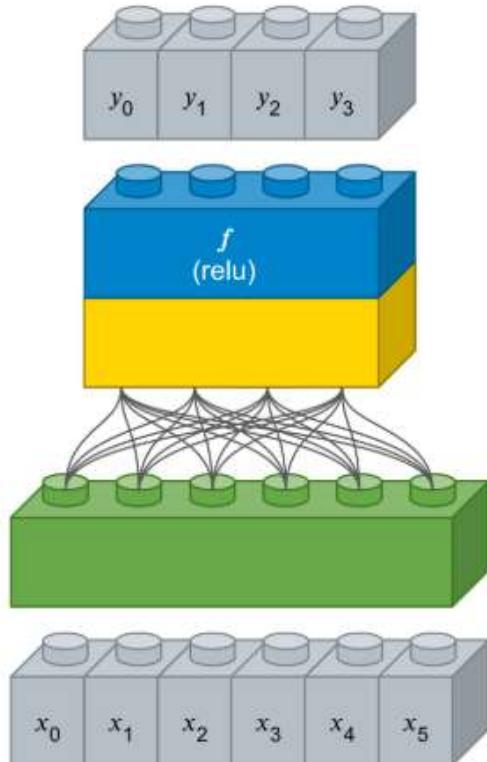
```
Dense(3, input_dim=4, activation='softmax')
```



인공 지능과 머신 러닝, 딥 러닝의 차이점

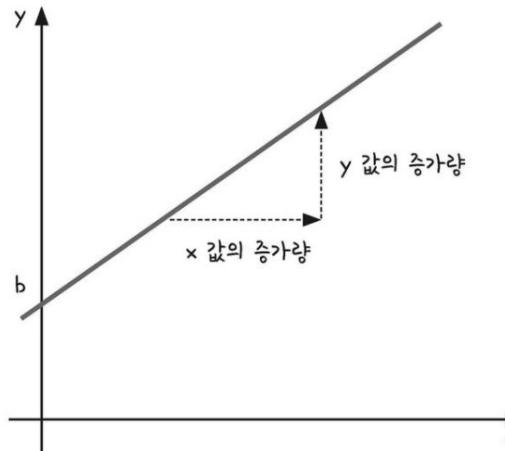
➤ Keras

```
Dense(4, input_dim=6, activation='relu'))
```



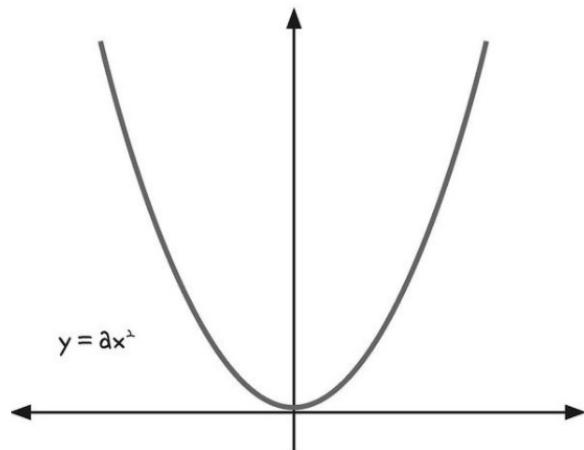
➤ 일차 함수, 기울기와 y 절편

- 딥러닝 – x 가 주어지고 원하는 y 값이 있을 때 적절한 a 와 b 를 찾는 것
- x 가 일차인 형태이며 x 가 일차로 남으려면 a 는 0이 아니어야 합니다.
- 일차 함수식 $y = ax + b$ 에서 a 는 기울기, b 는 절편

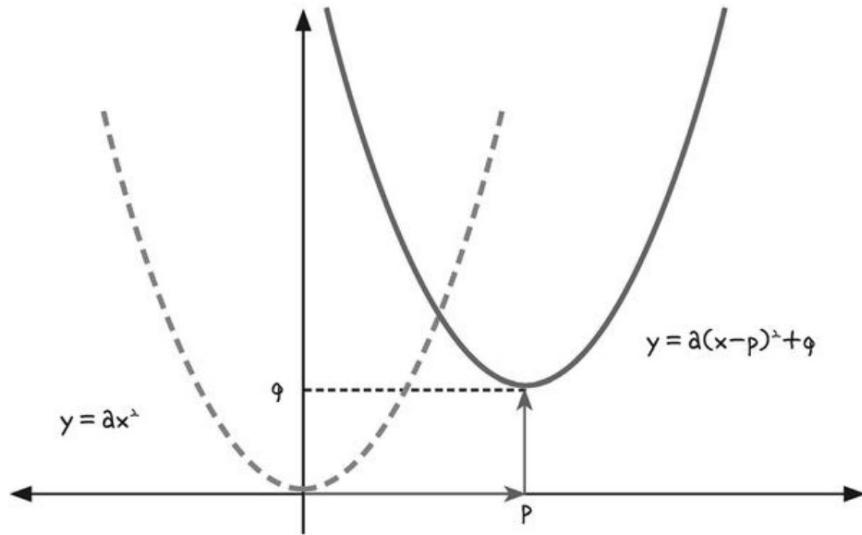


▶ 이차 함수와 최솟값

- $a > 0$ 이면 아래로 볼록한 그래프



$$y = ax^2 \quad (a \neq 0)$$



$y = ax^2$ 의 그래프를 x 축 방향으로 p 만큼, y 축 방향으로 q 만큼 평행 이동시키면, 점 p 와 q 를 꼭짓점으로 하는 포물선이 됩니다.
포물선의 맨 아래에 위치한 지점이 **최솟값**
딥러닝을 실행할 때는 이 최솟값을 찾아내는 과정이 매우 중요합니다.

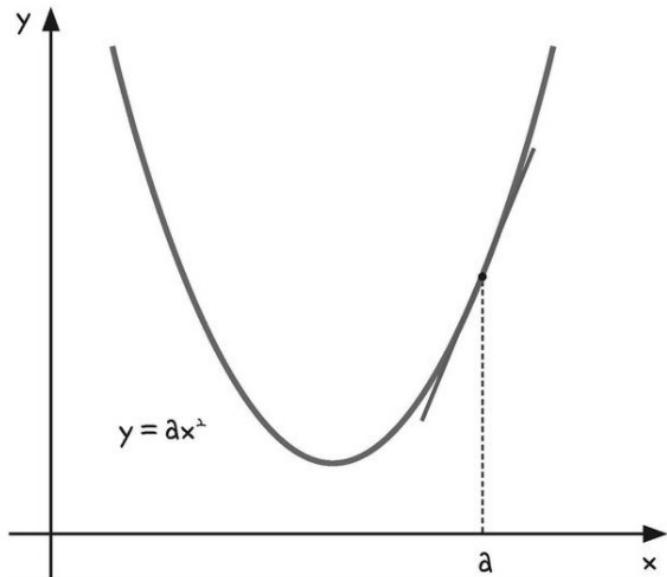
최솟값은 '최소 제곱법' 공식으로 쉽게 알아낼 수 있습니다.

딥러닝을 실행할 때 만나는 문제에서는 최소 제곱법을 계산하기 위해 꼭 필요한 조건들을 알 수 없기 때문에 대부분 최소 제곱법을 활용할 수가 없습니다.

따라서 미분과 기울기를 이용해야 합니다

➤ 일차 함수, 기울기와 y 절편

- 절편



$y = x^2$ 그래프에서 x축에 있는 한 점 a 에 대응하는 y 의 값은 a^2 입니다.

a 가 오른쪽이나 왼쪽으로 조금씩 이동하면 이에 따라 y 도 조금씩 변화할 것입니다.

순간 변화율은 어느 쪽을 향하는 방향성을 지니고 있으므로, 이 방향을 따라 직선을 길게 그려주면 그래프와 맞닿는 접선이 그려집니다. 이 선이 바로 이 점에서의 기울기가 됩니다.

a 에서의 순간 변화율은 곧 기울기다

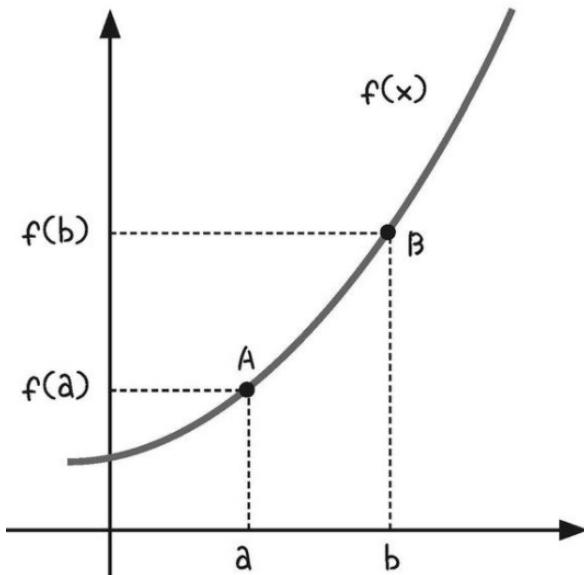
➤ 미분 , 순간 변화율과 기울기

미분을 한다는 것은 '순간 변화율'을 구한다는 것입니다.

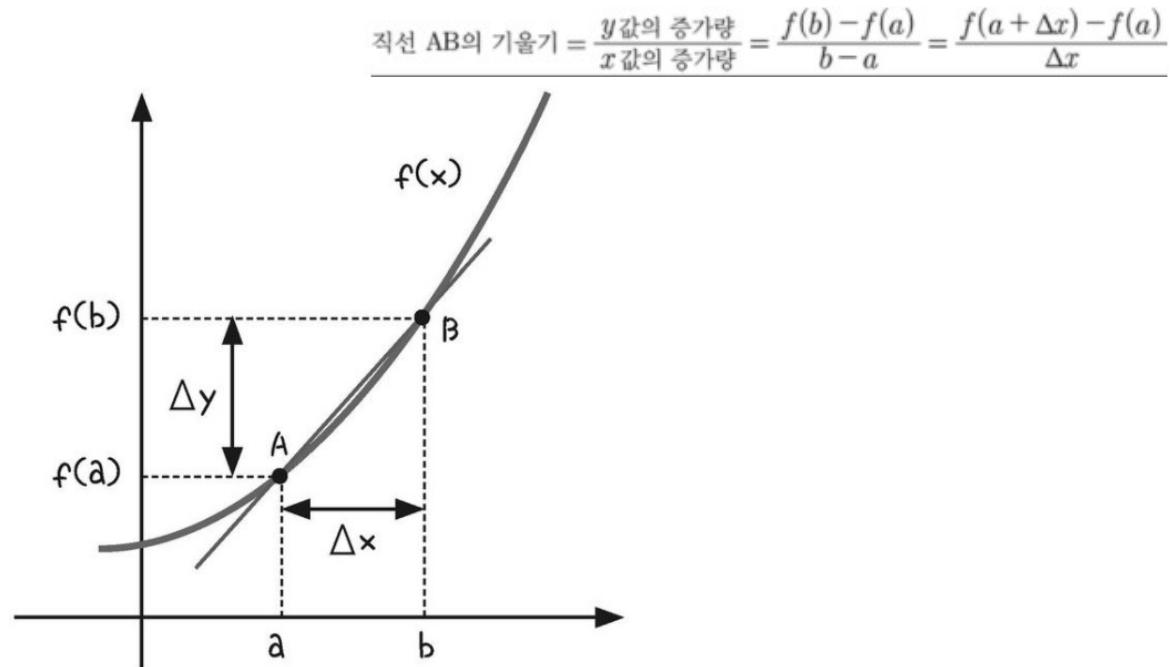
어느 순간에 어떤 변화가 일어나고 있는지를 숫자로 나타낸 것을 미분 계수라고 하며,

미분 계수는 곧 그래프에서의 기울기를 의미합니다.

기울기가 0일 때, 즉 x축과 평행한 직선으로 그어질 때가 바로 그래프에서 최솟값인 지점이 됩니다



함수에 x 축 위의 두 실수 a 와 b 를 대입하면 두 점 A , B 는 각각 $A(a, f(a))$, $B(b, f(b))$ 에 해당하는 곳에 표시됩니다.

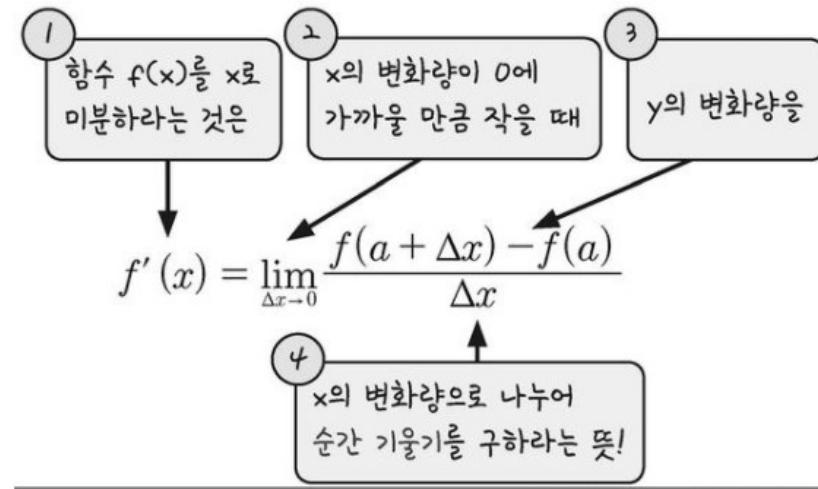
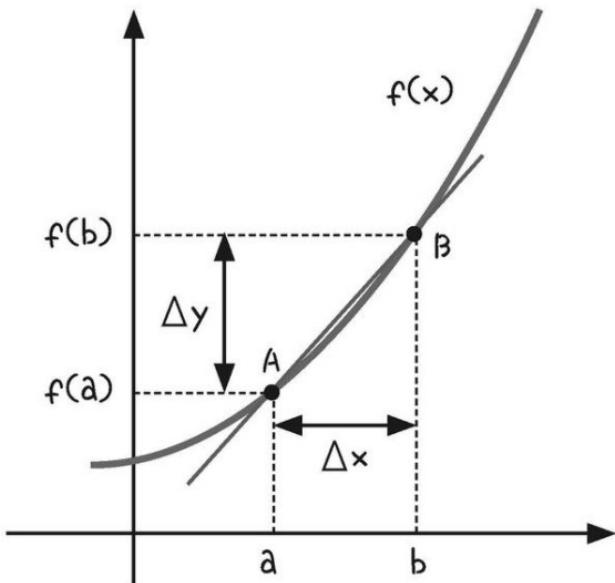


두 점 A 와 B 를 이어 직선을 만들면 두 점 A 와 B 를 지나는 직선의 기울기가 그려집니다. Δ (델타)는 변화량을 나타내는 기호입니다

➤ 미분 , 순간 변화율과 기울기

직선 AB의 기울기 = A와 B 사이의 '평균 변화율' = 순간 변화율

순간 변화율은 x의 증가량(Δx)이 0에 가까울 만큼 아주 작을 때의 순간적인 기울기



$f(x) = a$ 에서 a 가 상수일 때 미분 값은 0입니다.

$f(x) = x$ 일 때의 미분 값은 1입니다.

$f(x) = ax$ 에서 a 가 상수이면 미분 값은 a 입니다.

$f(x) = x^a$ 에서 a 가 자연수이면 미분 값은 ax^{a-1} 이 됩니다.

➤ 편미분

한 가지 변수만 미분하고 그 외에는 모두 상수로 취급하는 것

$$f(x, y) = x^2 + yx + a \quad (a \text{는 상수})$$

변수가 x 와 y , 두 개 중 어떤 변수로 미분해야 하는지를 정해야 하므로 편미분을 사용하는 것입니다.

여러 변수 중에서 x 에 대해서만 미분하고 싶다면, 함수 f 를 ' x 에 관해 편미분하라'고 하며 $\frac{\partial f}{\partial x}$ † 같이 식을 씁니다.

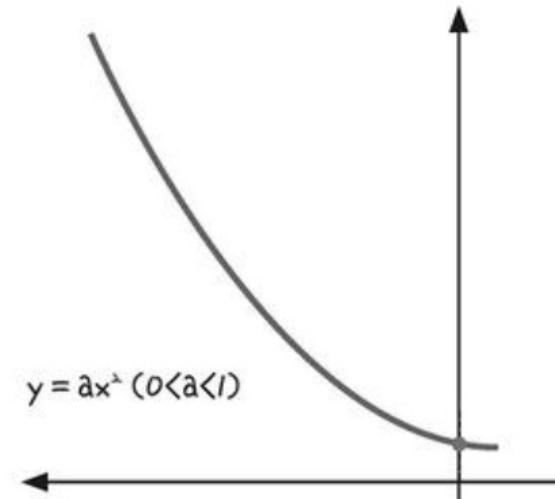
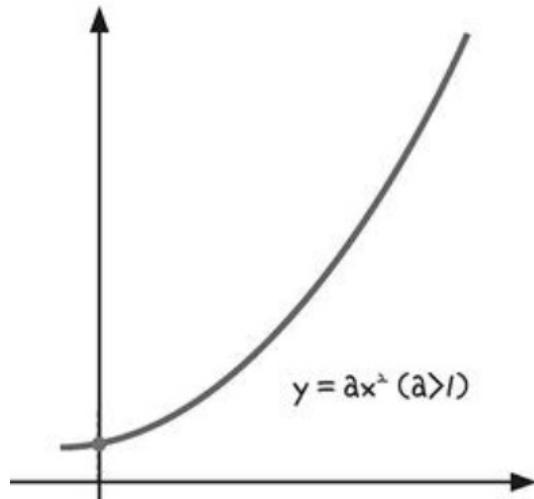
$$\underline{f(x, y) = x^2 + xy + a \text{ 일 때}},$$

$$\underline{\frac{\partial f}{\partial x} = 2x + y}$$

➤ 지수와 지수함수

지수 함수에서는 밑(a)의 값이 무엇인지가 중요합니다. 이 값이 1이면 함수가 아닙니다.
또 0보다 작으면 허수를 포함하게 되므로 안 됩니다.
따라서 밑의 값은 $a > 1$ 이거나 $0 < a < 1$, 둘 중 하나가 되어야 합니다.

$$y = a^x \quad (a \neq 1, a > 0)$$



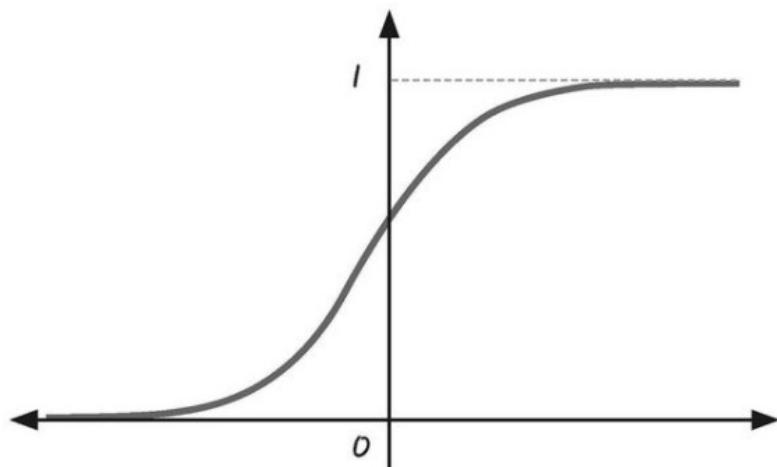
➤ 시그모이드 함수

시그모이드 함수는 지수 함수에서 밑의 값이 자연 상수 e 인 함수를 말합니다.

자연 상수 e 는 '자연 로그의 밑', '오일러의 수' 등 여러 이름으로 불리는데, 파이(π)처럼 수학에서 중요하게 사용되는 무리수이며, 그 값은 대략 2.718281828...입니다.

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

x 가 큰 값을 가지면 $f(x)$ 는 1에 가까워지고, x 가 작은 값을 가지면 $f(x)$ 는 0에 가까워집니다.
S자 형태로 그려지는 이 함수의 속성은 0 또는 1, 두 개의 값 중 하나를 고를 때 유용하게 쓰입니다.



➤ 로그와 로그 함수

a를 x만큼 거듭제곱한 값이 b라고 할 때,

$$a^x = b$$

a와 b를 알고 있는데 x를 모르는 경우, x를 구하기 위해 사용하는 방법이 로그입니다.

$$\begin{array}{c} a^x = b \\ \hline \log_a b = x \end{array}$$

로그 함수와 지수 함수와의 관계는 역함수의 관계입니다. 역함수는 x와 y를 서로 바꾸어 가지는 함수입니다.

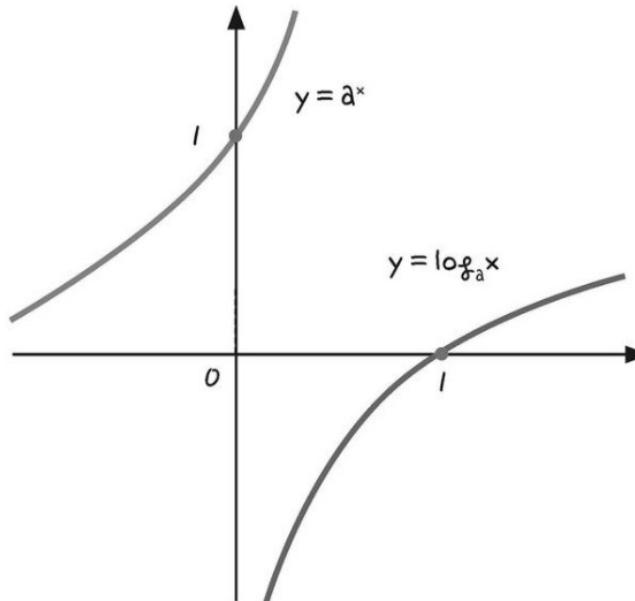
지수 함수 $y = a^x$ ($a \neq 1, a > 0$)는 로그의 정의를 따라 $x = \log_a y$ 로 바꿀 수 있습니다. 역함수를 만들기 위해 x와 y를 서로 바꾸어 줍니다.

$$y = \log_a x$$

역함수의 그래프는 $y = x$ 에 대하여 대칭인 선으로 나타납니다.

지수 함수 $y = a^x$ 의 그래프를 $y = x$ 에 대칭으로 이동시킨

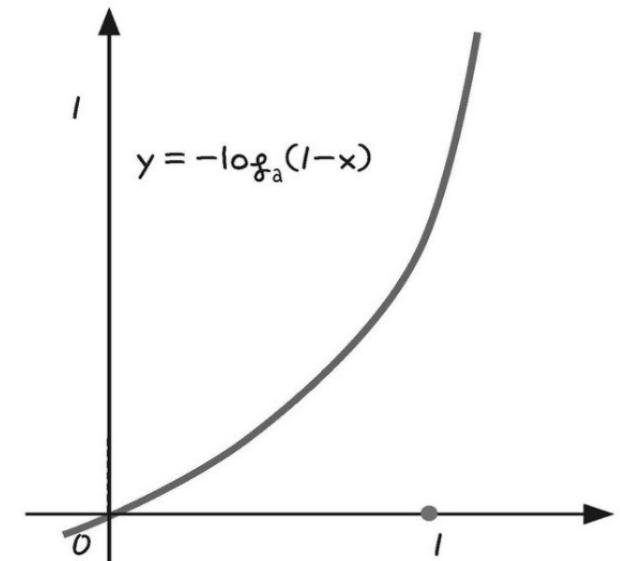
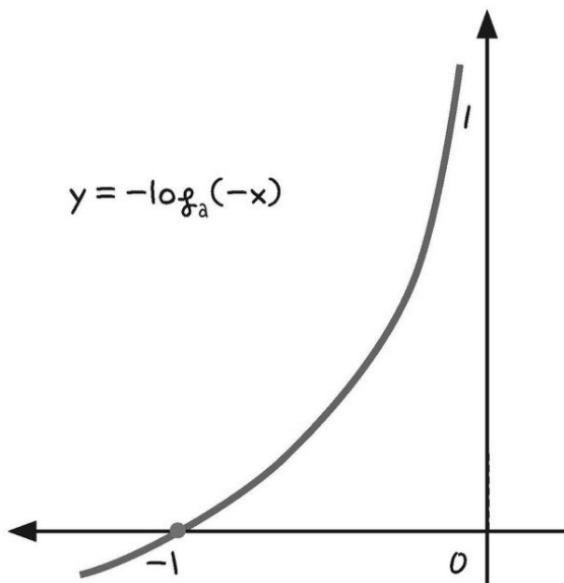
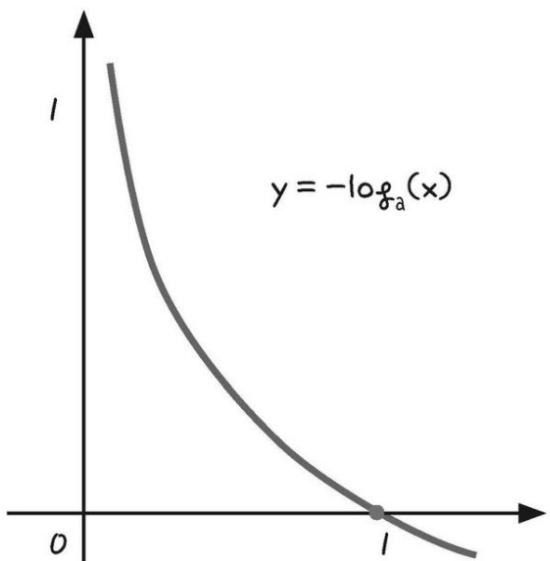
로그 함수 $y = \log_a x$ 의 그래프를 보여줍니다.



➤ 로그와 로그 함수

x 가 1에 가까워지거나, 0에 가까워질수록 오차가 커지는 그래프

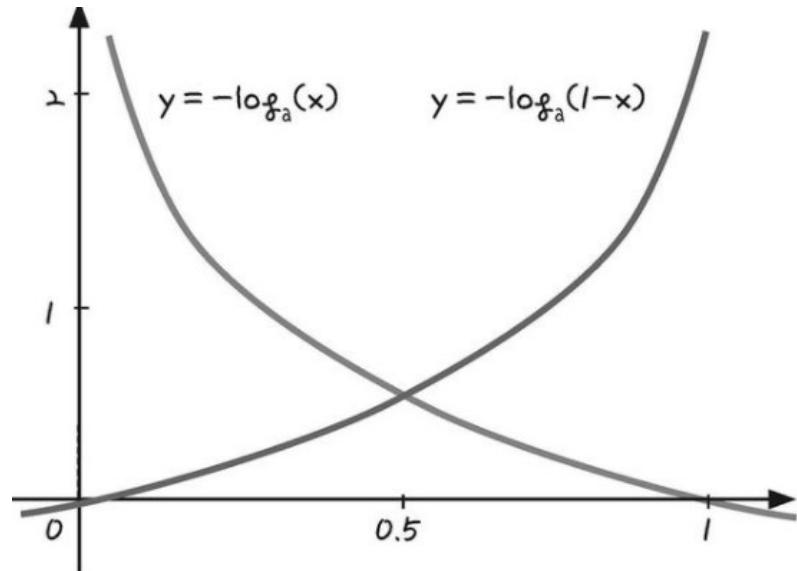
$y = \log_a x$ 를 x 축 또는 y 축으로 대칭 이동하거나 알맞게 평행 이동하면 다음과 같습니다.



그래프를 x 축 오른쪽 방향으로 1만큼 평행 이동

➤ 로그와 로그 함수

$y = -\log_a x$ 와 $y = -\log_a(1-x)$ 그래프



합을 표현하기 위해 만들어진 시그마 기호가 종종 나옵니다. \sum

$\sum_{i=1}^n F(i)$ i를 1부터 n까지 F(i)에 대입해 더하라는 뜻입니다



Session 2

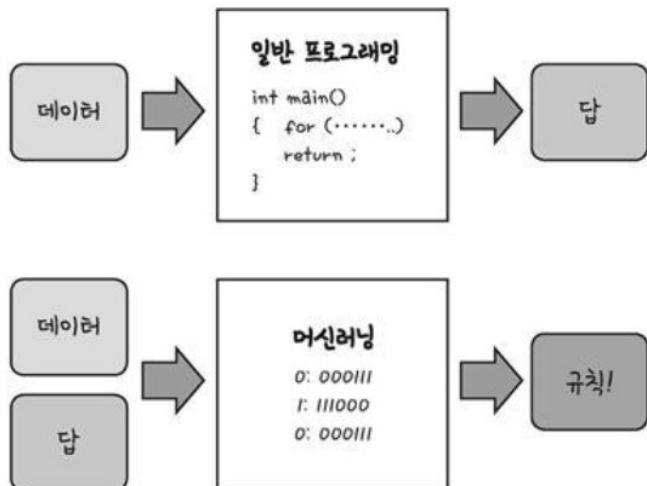
딥러닝 동작원리



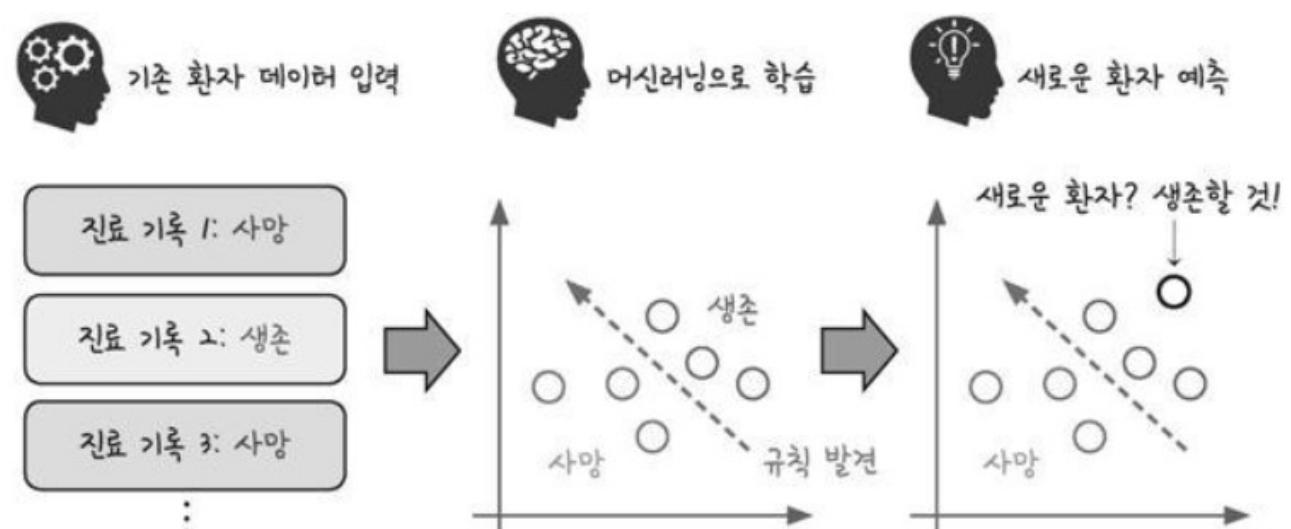
Deep Learning

➤ 머신러닝

- 데이터 안에서 규칙을 발견하고 그 규칙을 새로운 데이터에 적용해서 새로운 결과를 도출
 - 기존 데이터를 이용해 아직 일어나지 않은 미지의 일을 예측하기 위해 만들어진 기법



머신러닝과 일반 프로그래밍 비교



머신러닝의 학습 및 예측 과정

Deep Learning

➤ 폐암 수술 환자의 생존율 예측하기

```
.....
# 실행할 때마다 같은 결과를 출력하기 위해 설정하는 부분입니다.
seed = 0
numpy.random.seed(seed)
tf.set_random_seed(seed)

# 준비된 수술 환자 데이터를 불러들입니다.
Data_set = numpy.loadtxt("../dataset/ThoracicSurgery.csv", delimiter=",")

# 환자의 기록과 수술 결과를 X와 Y로 구분하여 저장합니다.
X = Data_set[:,0:17]
Y = Data_set[:,17]

# 딥러닝 구조를 결정합니다(모델을 설정하고 실행하는 부분입니다).
model = Sequential()
model.add(Dense(30, input_dim=17, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# 딥러닝을 실행합니다.
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
model.fit(X, Y, epochs=30, batch_size=10)
....
```

딥러닝 동작원리

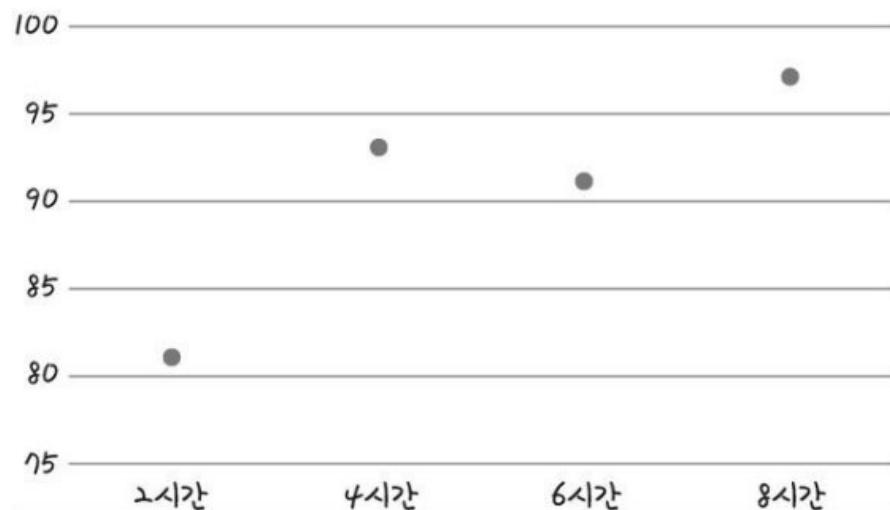
➤ 선형 회귀(linear regression)

- 단순 선형 회귀(simple linear regression) - 하나의 x 값만으로 y 값을 설명
- 다중 선형 회귀(multiple linear regression) - 여러개의 x 값이 y 값을 설명

공부한 시간	2시간	4시간	6시간	8시간
성적	81점	93점	91점	97점

$$x = \{2, 4, 6, 8\}$$

$$y = \{81, 93, 91, 97\}$$



선형(직선으로 표시될 만한 형태)은 일차 함수 그래프로 표현됩니다.

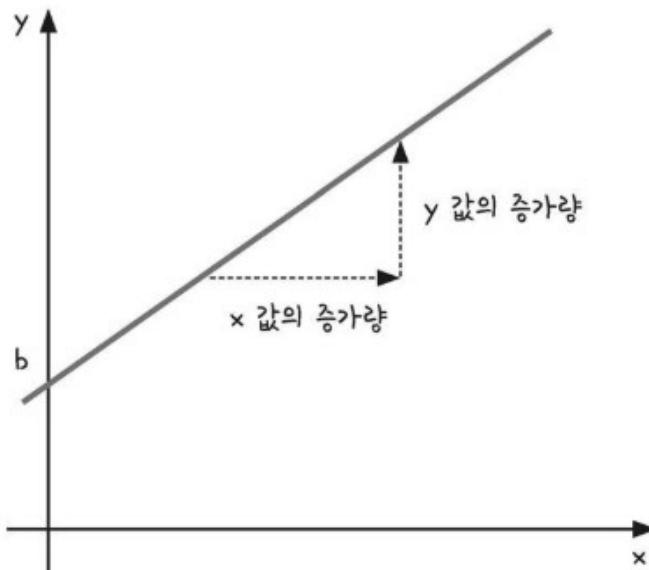
$$y = ax + b$$

딥러닝 동작원리

➤ 선형 회귀(linear regression)

- 선형 회귀는 정확한 직선을 그려내는 과정으로 직선의 기울기 a 값과 y 절편 b 값을 정확히 예측해 내야 하는 것

a 는 직선의 기울기, 즉 $\frac{y \text{ 값의 증가량}}{x \text{ 값의 증가량}}$ 이고, b 는 y 축을 지나는 값인 ‘ y 절편’



딥러닝 동작원리

➤ 최소 제곱법(method of least squares)

- 최소 제곱법 - 회귀 분석에서 사용되는 표준 방식
- 실험이나 관찰을 통해 얻은 데이터를 분석하여 미지의 상수를 구할 때 사용되는 공식
- 최소 제곱법을 통해 일차 함수의 기울기 a 와 y 절편 b 를 구할 수 있습니다.

$$a = \frac{\sum_{i=1}^n (x - \text{mean}(x))(y - \text{mean}(y))}{\sum_{i=1}^n (x - \text{mean}(x))^2}$$
$$a = \frac{(x - x \text{ 평균})(y - y \text{ 평균}) \text{의 합}}{(x - x \text{ 평균})^2 \text{의 합}}$$

x 의 편차(각 값과 평균과의 차이)를 제곱해서 합한 값을 분모로 놓고, x 와 y 의 편차를 곱해서 합한 값을 분자로 놓으면 기울기가 나옵니다.

- 공부한 시간(x) 평균: $(2 + 4 + 6 + 8) \div 4 = 5$
- 성적(y) 평균: $(81 + 93 + 91 + 97) \div 4 = 90.5$

$$a = \frac{(2 - 5)(81 - 90.5) + (4 - 5)(93 - 90.5) + (6 - 5)(91 - 90.5) + (8 - 5)(97 - 90.5)}{(2 - 5)^2 + (4 - 5)^2 + (6 - 5)^2 + (8 - 5)^2}$$
$$= \frac{46}{20}$$
$$= 2.3$$

딥러닝 동작원리

➤ 최소 제곱법(method of least squares)

y 절편인 b를 구하는 공식 : $b = \text{mean}(y) - (\text{mean}(x) * a)$

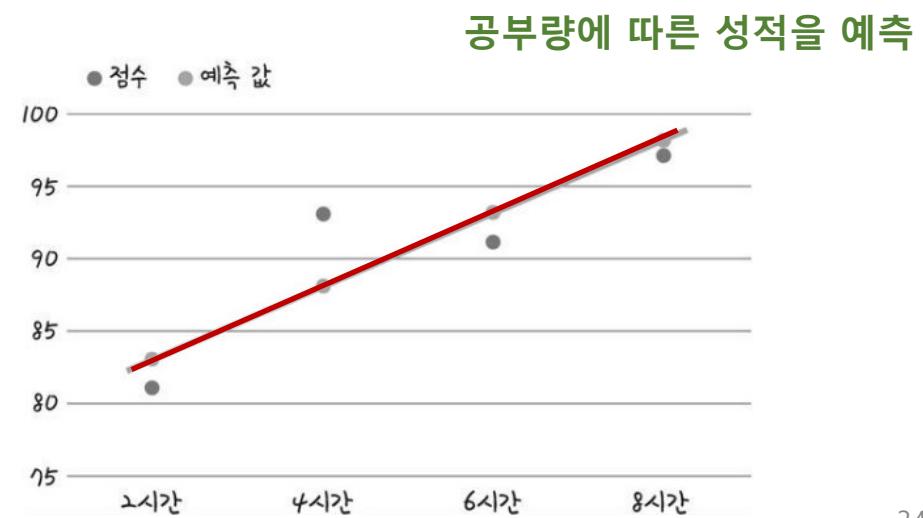
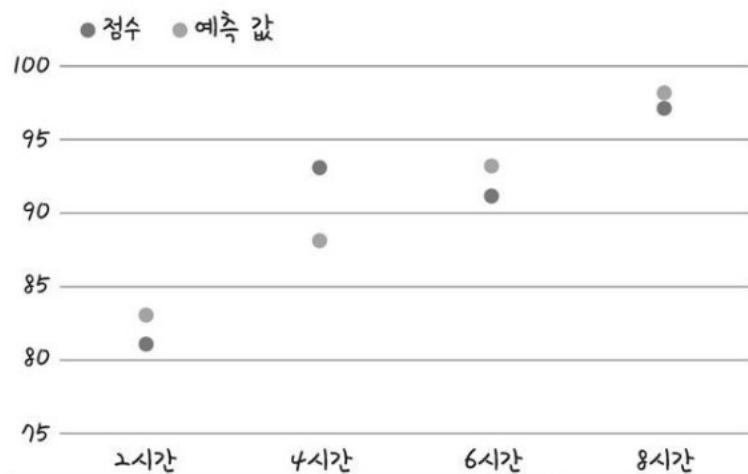
$$b = y\text{의 평균} - (x\text{의 평균} \times \text{기울기 } a)$$

$$b = 90.5 - (2.3 \times 5) = 79$$

직선의 방정식 : $y = 2.3x + 79$

최소 제곱법 공식으로 구한 성적 예측 값

공부한 시간	2	4	6	8
성적	81	93	91	97
예측 값	83.6	88.2	92.8	97.4



딥러닝 동작원리

- 최소 제곱법 공식으로 구한 성적 예측 코딩

```
import numpy as np
```

```
# x 값과 y 값  
x=[2, 4, 6, 8]  
y=[81, 93, 91, 97]
```

```
# x와 y의 평균값
```

```
# 기울기 공식의 분모
```

```
# 기울기 공식의 분자
```

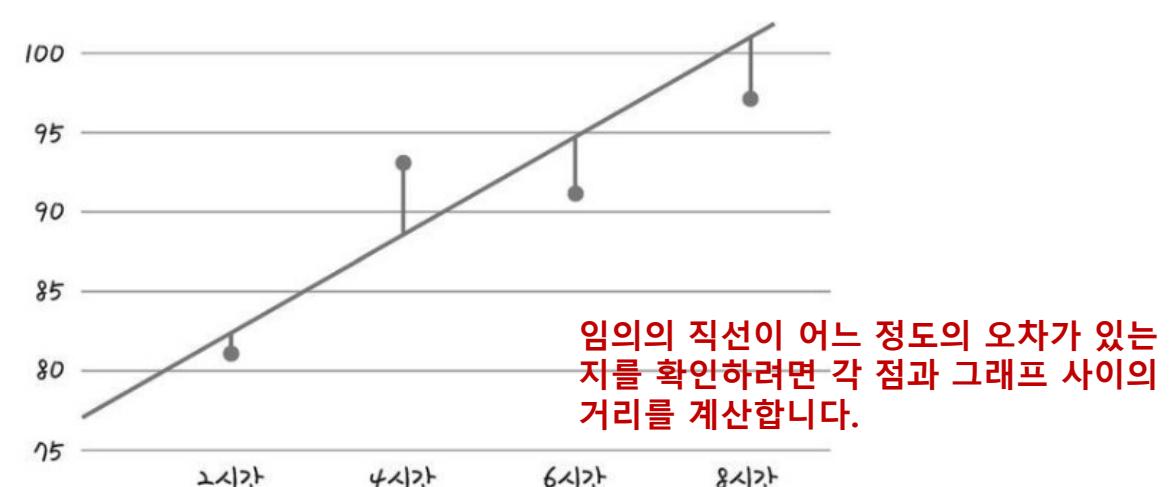
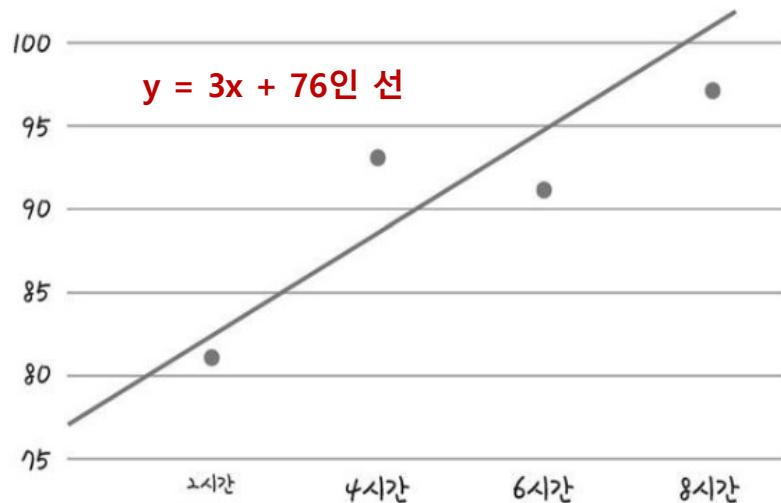
```
# 기울기와 y 절편 구하기
```

```
# 출력으로 확인
```

딥러닝 동작원리

➤ 평균 제곱근 오차(root mean square error)

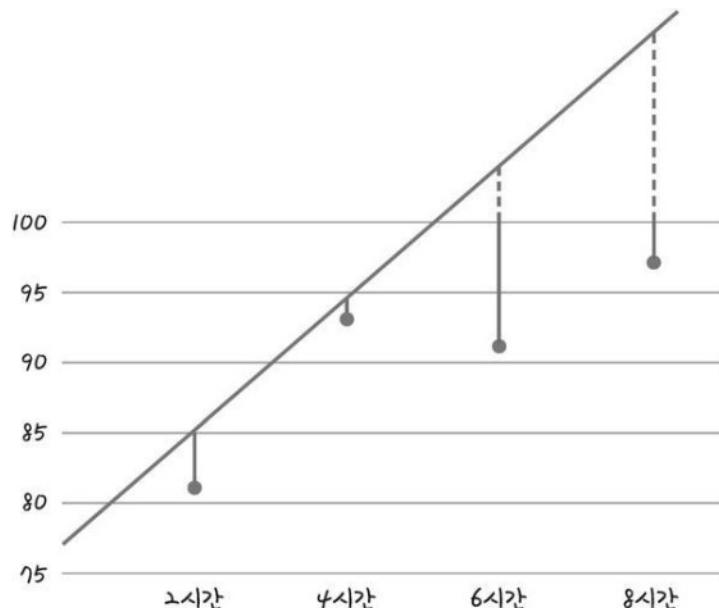
- 딥러닝은 대부분 입력 값이 여러 개인 상황에서 이를 해결하기 위해 실행됩니다.
- 여러 개의 입력 값을 계산할 때는 임의의 선을 그리고 난 후, 이 선이 얼마나 잘 그려졌는지를 평가하여 조금씩 수정해 가는 방법을 사용합니다.
- 가설을 하나 세운 뒤 이 값이 주어진 요건을 충족하는지 판단하여 조금씩 변화를 주고, 이 변화가 긍정적 이면 오차가 최소가 될 때까지 이 과정을 계속 반복하는 방법입니다.
- 선을 긋고 나서 수정하는 과정에서 나중에 그린 선이 먼저 그린 선보다 더 좋은지 나쁜지를 판단하는 방법이 필요합니다.
- 각 선의 오차를 계산할 수 있어야 하고, 오차가 작은 쪽으로 바꾸는 알고리즘이 필요합니다.



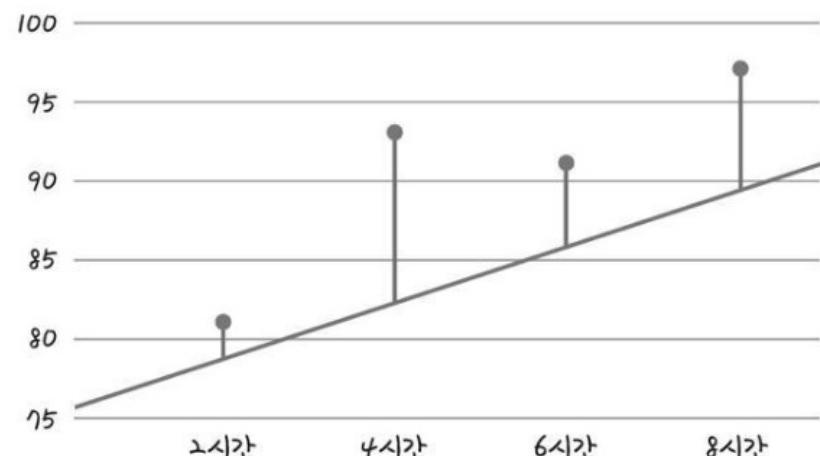
딥러닝 동작원리

➤ 평균 제곱근 오차(root mean square error)

- 기울기가 잘못되었을 수록 직선과의 거리의 합, 즉 오차의 합도 커집니다.
- 기울기가 무한대로 커지면 오차도 무한대로 커지는 상관관계가 있습니다.



기울기를 너무 크게 잡았을 때의 오차



기울기를 너무 작게 잡았을 때의 오차

딥러닝 동작원리

➤ 평균 제곱근 오차(root mean square error)

- 오차 = 실제 값 - 예측 값
- 오차의 합을 구할 때, 오차에 양수와 음수가 섞여 있기 때문에 오차를 단순히 더해 버리면 합이 0이 될 수도 있으므로 각 오차의 값을 제곱해 줍니다
- 평균 제곱 오차(Mean Squared Error, MSE) - 오차 합의 평균
- 평균 제곱근 오차(Root Mean Squared Error, RMSE) - 평균 제곱 오차는 각 오차를 제곱한 값을 사용하므로 대용량 데이터를 이용할 때는 오차가 커지고, 계산 속도가 느려지는 경우 제곱근을 씌워 줍니다.

공부한 시간(x)	2	4	6	8
성적(실제 값, y)	81	93	91	97
예측 값	82	88	94	100
오차	1	-5	3	3

$y = 3x + 76$ 식의 데이터에 대한 오차의 값

$$\text{오차의 합} = \sum_{i=1}^n (p_i - y_i)^2$$

$$\text{평균 제곱 오차(MSE)} = \frac{1}{n} \sum_{i=1}^n (p_i - y_i)^2$$

$$\text{평균 제곱근 오차(RMSE)} = \sqrt{\frac{1}{n} \sum_{i=1}^n (p_i - y_i)^2}$$

선형 회귀란 임의의 직선을 그어 이에 대한 평균 제곱근 오차를 구하고, 이 값을 가장 작게 만들어 주는 a와 b 값을 찾아가는 작업입니다.

딥러닝 동작원리

➤ 평균 제곱근 오차(root mean square error) 구현 코드

```
import numpy as np

# 기울기 a와 y 절편 b
ab = [3, 76]

# x, y의 데이터 값
data = [[2, 81], [4, 93], [6, 91], [8, 97]]
x = [i[0] for i in data]
y = [i[1] for i in data]

# y = ax + b에 a와 b 값을 대입하여 결과를 출력하는 함수

# RMSE 함수

# RMSE 함수를 각 y 값에 대입하여 최종 값을 구하는 함수

# 예측 값이 들어갈 빈 리스트

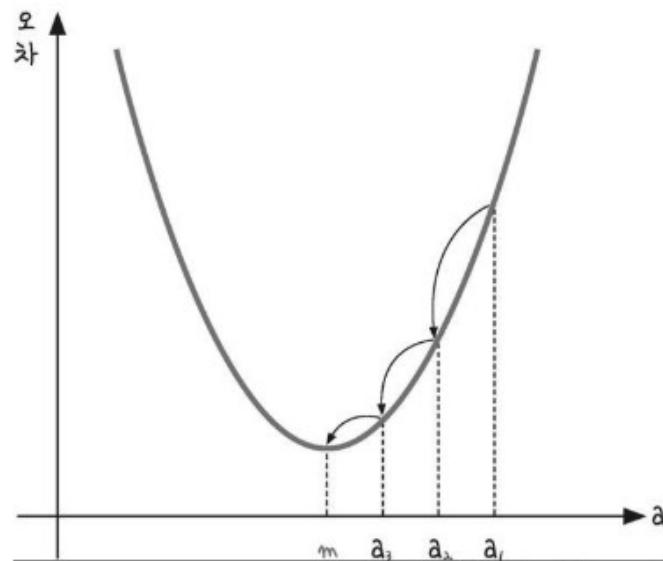
# 모든 x 값을 한 번씩 대입하여

# 최종 RMSE 출력
```

딥러닝 동작원리

➤ 오차 수정 – 경사하강법(gradient decent)

- 기울기 a 를 무한대로 키우면 오차도 무한대로 커지고 a 를 무한대로 작게 해도 역시 오차도 무한대로 작아지는 상관관계는 이차 함수 그래프로 표현할 수 있습니다.



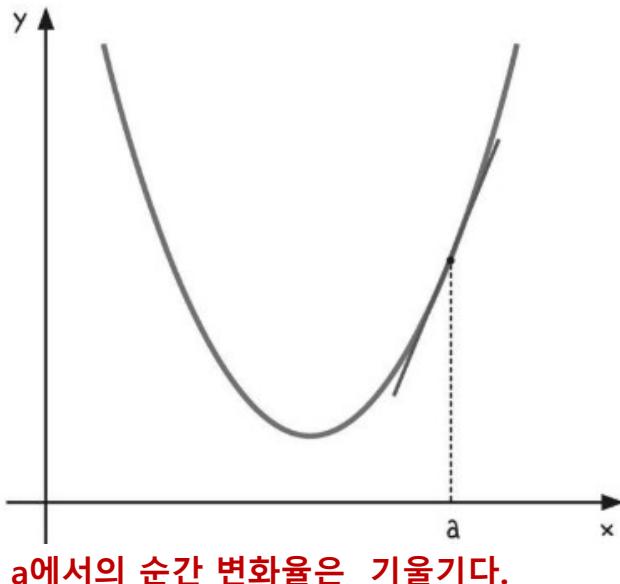
- 오차가 가장 작은 때는 x 가 그래프의 가장 아래쪽의 볼록한 부분에 이르렀을 때입니다.
- 즉, 기울기 a 가 m 위치에 있을 때입니다.
- 컴퓨터를 이용해 m 의 값을 구하려면 임의의 한 점(a_1)을 찍고 이 점을 m 에 가까운 쪽으로 점점 이동($a_1 \rightarrow a_2 \rightarrow a_3$)시키는 과정이 필요합니다.
- (a_1 의 값보다 a_2 의 값이 m 에 더 가깝고 a_2 의 값보다 a_3 가 m 에 더 가깝다는 것을 컴퓨터가 알아야 하겠지요)
- **미분 기울기를 이용하는 경사 하강법(gradient decent)은 이차 함수 그래프에서 오차를 비교하여 가장 작은 방향으로 이동시키는 방법입니다.**

기울기 a 와 오차와의 관계: 적절한 기울기를 찾았을 때 오차가 최소화된다.

딥러닝 동작원리

➤ 미분

- $Y = X^2$ 이라는 그래프의 x축에 한 점 a 가 있을 때 y 값은 $Y = a^2$ 입니다.
- a 가 아주 미세하게 오른쪽이나 왼쪽으로 이동하면 종속 변수인 y 값도 그에 따라 아주 미세하게 변화합니다.
- a 가 변화량이 0에 가까울 만큼 아주 미세하게 변화하면 y 값의 변화 역시 아주 미세해서 0에 가깝게 변화합니다. (순간변화율)
- 순간 변화율은 '어느 쪽'이라는 방향성을 지니고 있으므로 이 방향에 맞추어 직선(점에서의 '기울기', 접선)을 그릴 수가 있습니다.
- 미분은 x 값이 아주 미세하게 움직일 때의 y 변화량을 구한 뒤, 이를 x 의 변화량으로 나누는 과정입니다.



"함수 $f(x)$ 를 미분하라"

$$\frac{d}{dx}f(x)$$

$$\frac{d}{dx}f(x) = \lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

① 함수 $f(x)$ 를 x 로 미분하라는 것은
② x 의 변화량이 0에 가까울 만큼 작을 때
③ y 변화량의 차이를
④ x 변화량으로 나눈 값(= 순간 변화율)을 구하라는 뜻!

a에서의 순간 변화율은 기울기다.

딥러닝 동작원리

- 도함수: 어떤 함수 안에 포함된 값 각각이 0에 한없이 가까워지는 극한값(미분계수)을 구하는 함수
- $y = f(x)$ 의 도함수 $f'(x)$ 는 아래와 같이 정의

$$f'(x) = \lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

- $f(x) = 3x$ 일 때, 도함수 계산 과정

$$f'(x) = \lim_{\Delta x \rightarrow 0} \frac{3(x + \Delta x) - 3x}{\Delta x} = \lim_{\Delta x \rightarrow 0} \frac{3\Delta x}{\Delta x} = 3$$

- $f(x) = x^2$ 일 때, 도함수 계산 과정

$$\begin{aligned} f'(x) &= \lim_{\Delta x \rightarrow 0} \frac{(x + \Delta x)^2 - x^2}{\Delta x} = \lim_{\Delta x \rightarrow 0} \frac{2x\Delta x + (\Delta x)^2}{\Delta x} \\ &= \lim_{\Delta x \rightarrow 0} (2x + \Delta x) = 2x \end{aligned}$$

- 함수 $f(x)$ 의 도함수 $f'(x)$ 를 구하는 것을 “함수 $f(x)$ 를 미분한다”라고 하며, 위와 같이 값을 계산할 수 있다면 미분 가능이라고 함

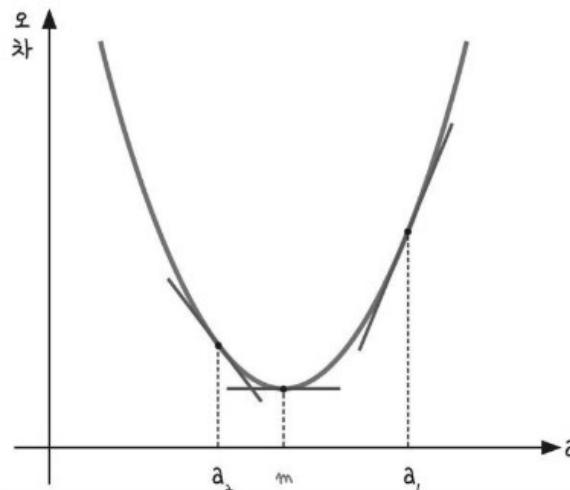
➤ 미분 결과

- $f(x) = a$ (a 는 상수)일 때 미분 값은 0
- $f(x) = x$ 일 때 미분 값은 1
- $f(x) = ax$ (a 는 상수)일 때 미분 값은 a
- $f(x) = x^a$ (a 는 자연수)일 때 미분 값은 ax^{a-1}

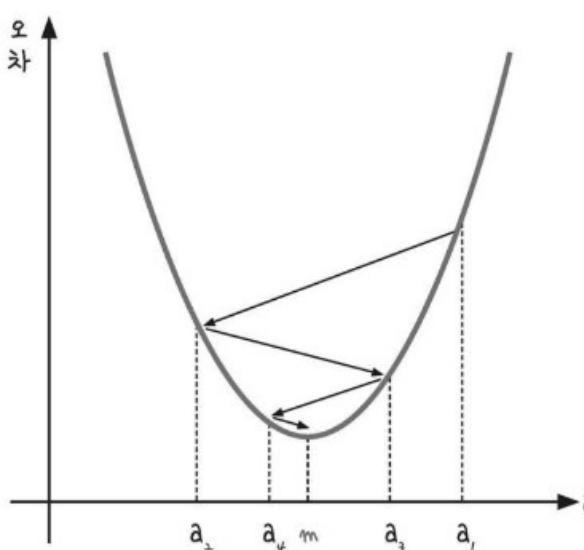
딥러닝 동작원리

▶ 경사하강법(gradient decent)

- $y = X^2$ 그래프에서 X 에 a_1, a_2 그리고 m 을 대입하여 미분하면 각 점에서의 순간 기울기가 그려집니다.
- 순간 기울기가 0인 점이 최솟값 m 이다.
- 그래프가 이차 함수 포물선이므로 꼭짓점의 기울기는 x 축과 평행한 선이 되며, 기울기가 0입니다.
- 경사 하강법은 반복적으로 기울기 a 를 변화시켜서 m 의 값을 찾아내는 방법 ('미분 값이 0인 지점'을 찾는 것)입니다.



a 순간 기울기가 0인 점이 최솟값 m이다.

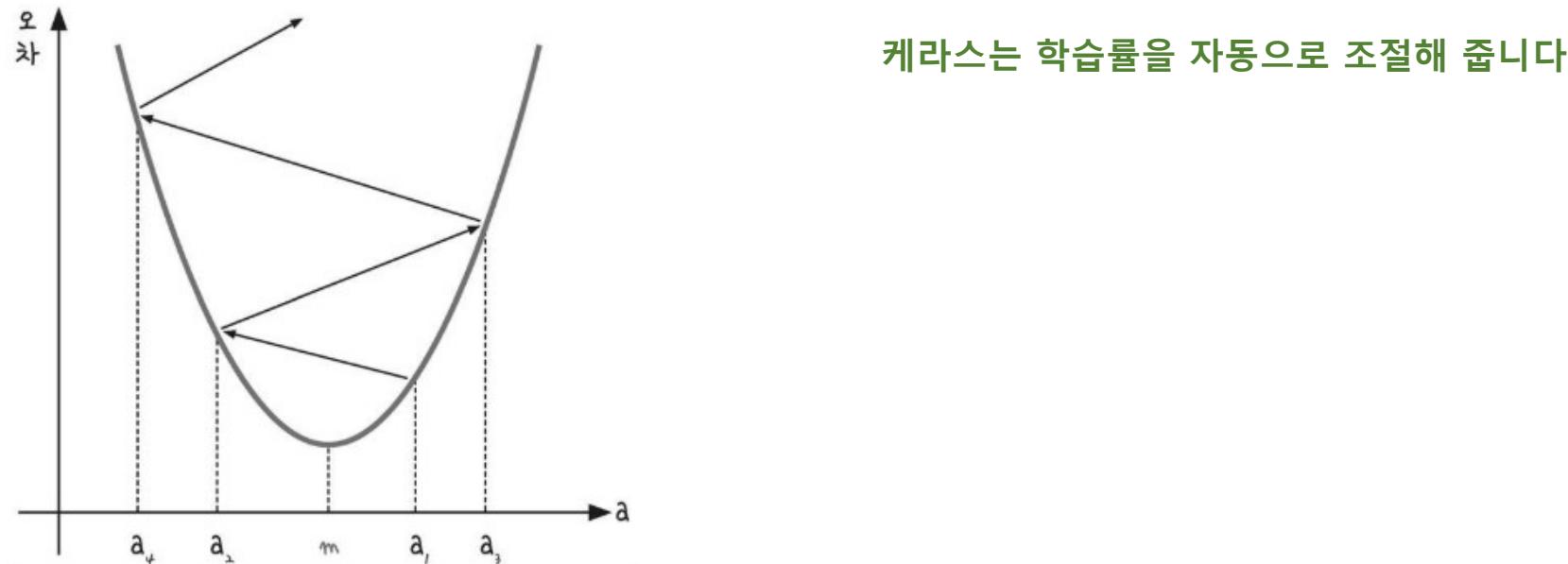


- a_1 에서 미분을 구한다.
- 구해진 기울기의 반대 방향(기울기가 +면 음의 방향, -면 양의 방향)으로 얼마간 이동시킨 a_2 에서 미분을 구한다
- a_3 에서 미분을 구한다.
- 0이 아니면 위 과정을 반복한다.

딥러닝 동작원리

➤ 학습률(learning rate)

- 기울기의 부호를 바꿔 이동시킬 때 적절한 거리를 찾지 못해 너무 멀리 이동시키면 a 값이 한 점으로 모아지 않고 위로 치솟아 버립니다
- 이동 거리를 정해주는 것이 바로 **학습률(learning rate)**입니다.
- 딥러닝에서 학습률의 값을 적절히 바꾸면서 최적의 학습률을 찾는 것은 중요한 최적화 과정 중 하나입니다..



학습률을 너무 크게 잡으면 한 점으로 수렴하지 않고 발산한다.

딥러닝 동작원리

➤ 경사하강법(gradient decent) 코드 구현

- 경사 하강법은 오차의 변화에 따라 이차 함수 그래프를 만들고 적절한 학습률을 설정해 미분 값이 0인 지점을 구하는 것입니다.
- y 절편 b의 값도 b 값이 너무 크면 오차도 함께 커지고 너무 작아도 오차가 커집니다.
- 최적의 b 값을 구할 때 역시 경사 하강법을 사용합니다.
- **GradientDescentOptimizer()** : 경사 하강법 함수
- 텐서플로는 session 함수를 이용해 구동에 필요한 리소스를 컴퓨터에 할당하고 이를 실행시킬 준비를 합니다.
- Session을 통해 구현될 함수를 텐서플로에서는 '그래프'라고 부른다
- Session이 할당되면 session.run('그래프명')의 형식으로 해당 함수를 실행시킵니다.
- **global_variables_initializer()** : 변수를 초기화하는 함수
- **gradient_descent** : 총 필요한 수만큼 반복하여 실행

딥러닝 동작원리

➤ 경사하강법(gradient decent) 코드 구현

```
import tensorflow as tf

# x, y의 데이터 값
data = [[2, 81], [4, 93], [6, 91], [8, 97]]
x_data = [x_row[0] for x_row in data]
y_data = [y_row[1] for y_row in data]

# 기울기 a와 y 절편 b의 값을 임의로 정한다.
# 단, 기울기의 범위는 0 ~ 10 사이이며, y 절편은 0 ~ 100 사이에서 변하게 한다.

# y에 대한 일차 방정식 ax+b의 식을 세운다.

# 텐서플로 RMSE 함수

# 학습률 값

# RMSE 값을 최소로 하는 값 찾기

# 텐서플로를 이용한 학습
```

- 에포크(Epoch)는 입력 값에 대해 몇 번이나 반복하여 실험했는지를 나타냅니다.

딥러닝 동작원리

➤ 경사 하강법으로 다중 선형 회귀 구현 코드

공부한 시간(x_1)	2	4	6	8
과외 수업 횟수(x_2)	0	4	2	3
성적(y)	81	93	91	97

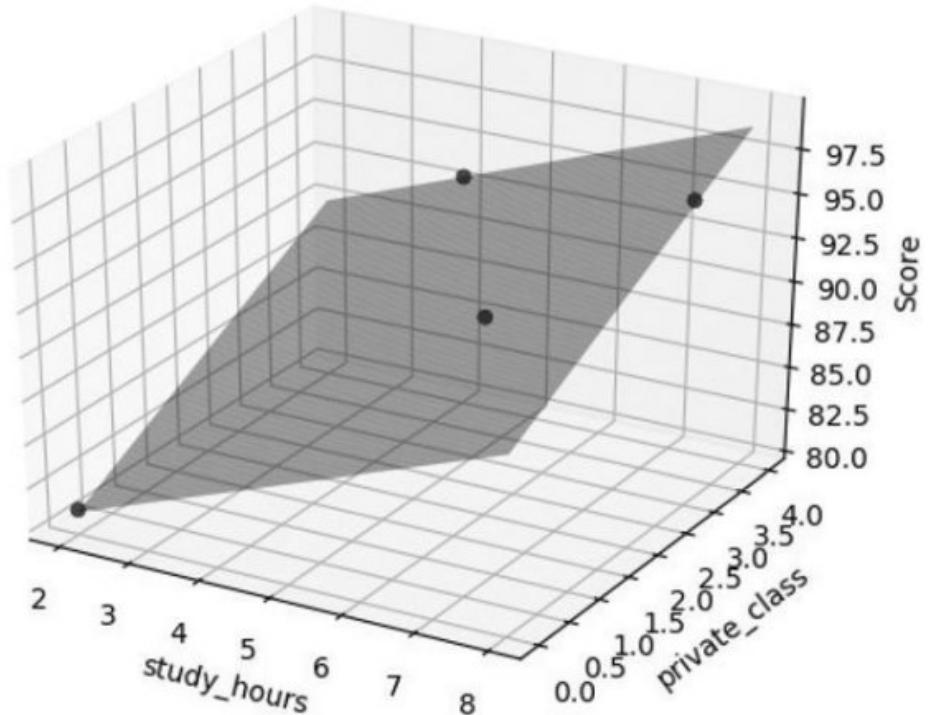
$$y = a_1x_1 + a_2x_2 + b$$

```
import tensorflow as tf
# x1, x2, y의 데이터 값
data = [[2, 0, 81], [4, 4, 93], [6, 2, 91], [8, 3, 97]]
x1 = [x_row1[0] for x_row1 in data]
x2 = [x_row2[1] for x_row2 in data] # 새로 추가되는 값
y_data = [y_row[2] for y_row in data]

# 기울기 a와 y 절편 b의 값을 임의로 정한다.
# 단, 기울기의 범위는 0 ~ 10 사이이며, y 절편은 0 ~ 100 사이에서 변하게 한다.
# 새로운 방정식
# 텐서플로 RMSE 함수
# 학습률 값
# RMSE 값을 최소로 하는 값 찾기
# 학습이 진행되는 부분
```

딥러닝 동작원리

- 경사 하강법으로 다중 선형 회귀 구현 코드

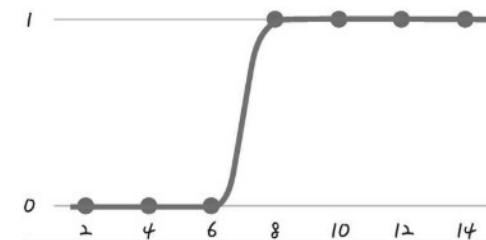
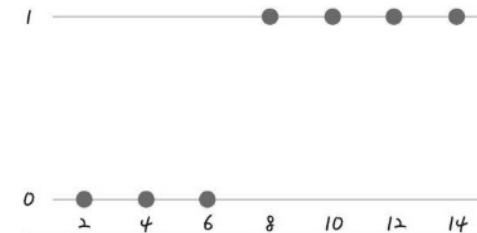


딥러닝 동작원리

➤ 로지스틱 회귀(logistic regression)

- 로지스틱 회귀는 선형 회귀와 마찬가지로 적절한 선을 그려가는 과정입니다. 다만, 직선이 아니라, 참(1)과 거짓(0) 사이를 구분하는 S자 형태의 선을 그어 주는 작업입니다.

공부한 시간	2	4	6	8	10	12	14
합격 여부	불합격	불합격	불합격	합격	합격	합격	합격

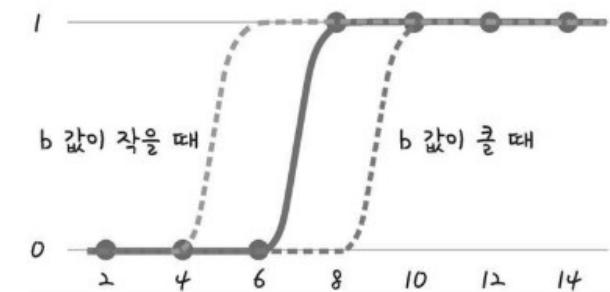
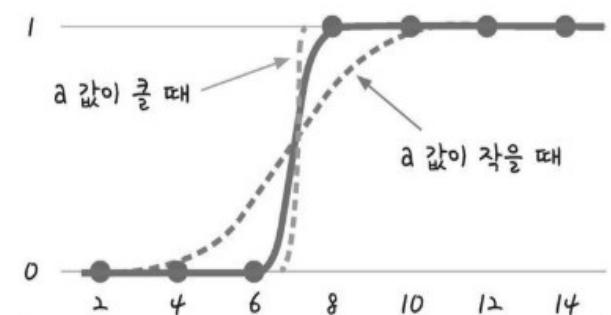


딥러닝 동작원리

▶ 시그모이드 함수(sigmoid function)

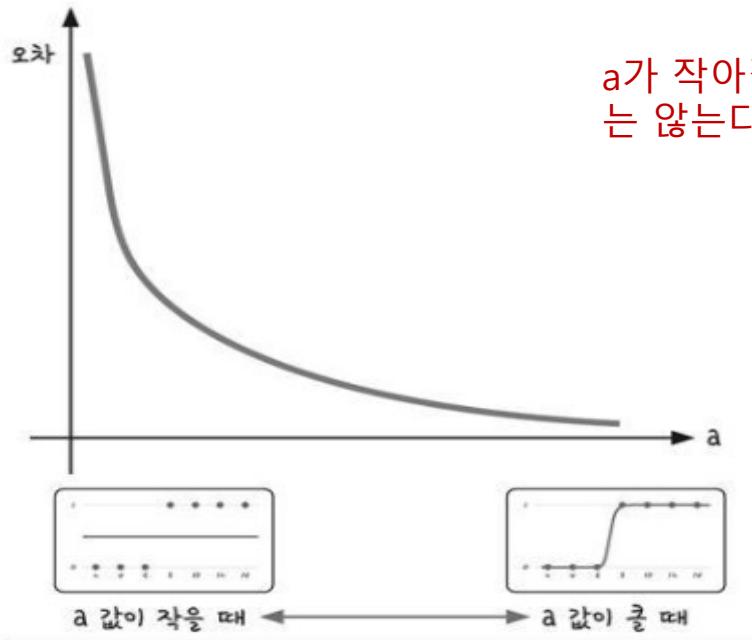
- S자 형태로 그래프가 그려지는 함수
- e는 자연 상수라고 불리는 무리수로 값은 2.71828...입니다.
- a는 그래프의 경사도를 결정합니다.
- a 값이 커지면 경사가 커지고 a 값이 작아지면 경사가 작아집니다
- b는 그래프의 좌우 이동을 의미합니다.
- b 값이 크고 작아짐에 따라 그래프가 이동합니다.

$$y = \frac{1}{1 + e^{(ax+b)}}$$

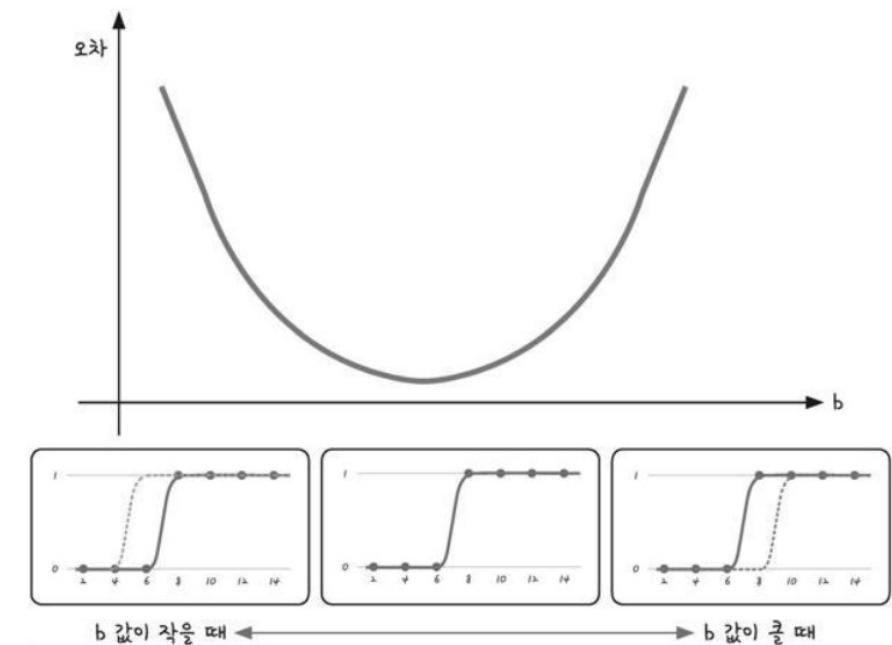


딥러닝 동작원리

▶ 시그모이드 함수(sigmoid function)



a가 작아질수록 오차는 무한대로 커지지만, a가 커진다고 해서 오차가 무한대로 커지지는 않는다.



b 값이 너무 작아지거나 커지면 오차도 무한대로 커진다.
이차 함수 그래프로 표현할 수 있습니다.

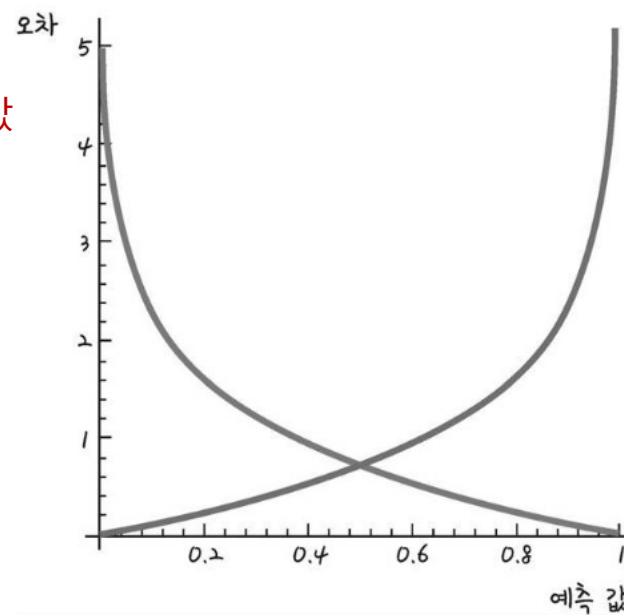
딥러닝 동작원리

➤ 오차공식

- 시그모이드 함수에서 a와 b의 값은 경사 하강법(오차를 구한 다음 오차가 작은 쪽으로 이동시키는 방법)으로 구합니다
- 시그모이드 함수의 특징은 y 값이 0과 1 사이입니다.
- 실제 값이 1일 때 예측 값이 0에 가까워지면 오차가 커져야 합니다.
- 실제 값이 0일 때 예측 값이 1에 가까워지는 경우에도 오차는 커져야 합니다.

예측 값이 1일 때 오차가 0이고, 예측 값이 0에 가까울수록 오차는 커지는 로그 함수 그래프

$-\log h$



예측 값이 0일 때 오차가 없고, 1에 가까워질수록 오차가 매우 커지는 로그 함수
그래프
 $-\log (1 - h)$

딥러닝 동작원리

➤ 로그 함수

- y 의 실제 값이 1일 때 $-\log h$ 그래프를 쓰고, 0일 때 $-\log(1 - h)$ 그래프를 써야 합니다.

$$-\underbrace{\{y \log h + (1 - y) \log(1 - h)\}}_{\text{A} + \text{B}}$$

- 실제 값 y 가 1이면 B 부분이 없어집니다. 반대로 0이면 A 부분이 없어집니다.

$$\text{오차} = -\text{평균}(y \log h + (1 - y) \log(1 - h))$$

딥러닝 동작원리

➤ 로지스틱 회귀 구현 코드

```
import tensorflow as tf
import numpy as np

# x, y의 데이터 값
data = [[2, 0], [4, 0], [6, 0], [8, 1], [10, 1], [12, 1], [14,1]]
x_data = [x_row[0] for x_row in data]
y_data = [y_row[1] for y_row in data]

# a와 b의 값을 임의로 정한다.

# y 시그모이드 함수의 방정식을 세운다.

# loss를 구하는 함수

# 학습률 값

# loss를 최소로 하는 값 찾기

# 학습
```

딥러닝 동작원리

➤ 여러 입력 값을 갖는 로지스틱 회귀 코드 구현

- `tf.placeholder('데이터형', '행렬의 차원', '이름')` 는 입력 값을 저장하는데 사용합니다..
- $a_1x_1 + a_2x_2$ 는 행렬곱을 이용해 $[a_1, a] * [x_1, x_2]$ 로도 표현할 수 있습니다
- 텐서플로에서는 `matmul()` 함수를 이용해 행렬곱을 적용합니다
- 텐서플로 내장 `sigmoid()` 함수 - 시그모이드를 계산

```
import tensorflow as tf
import numpy as np

# 실행할 때마다 같은 결과를 출력하기 위한 seed 값 설정
seed = 0
np.random.seed(seed)
tf.set_random_seed(seed)

# x, y의 데이터 값
x_data = np.array([[2, 3],[4, 3],[6, 4],[8, 6],[10, 7],[12,8],[14, 9]])
y_data = np.array([0, 0, 0, 1, 1, 1,1]).reshape(7, 1)

# 입력 값을 플레이스 홀더에 저장
```

딥러닝 동작원리

➤ 여러 입력 값을 갖는 로지스틱 회귀 코드 구현

```
# 기울기 a와 바이어스 b의 값을 임의로 정함  
# [2,1] 의미: 들어오는 값은 2개, 나가는 값은 1개  
  
# y 시그모이드 함수의 방정식을 세움  
# 오차를 구하는 함수  
# 학습률 값  
# 오차를 최소로 하는 값 찾기  
  
# 학습
```

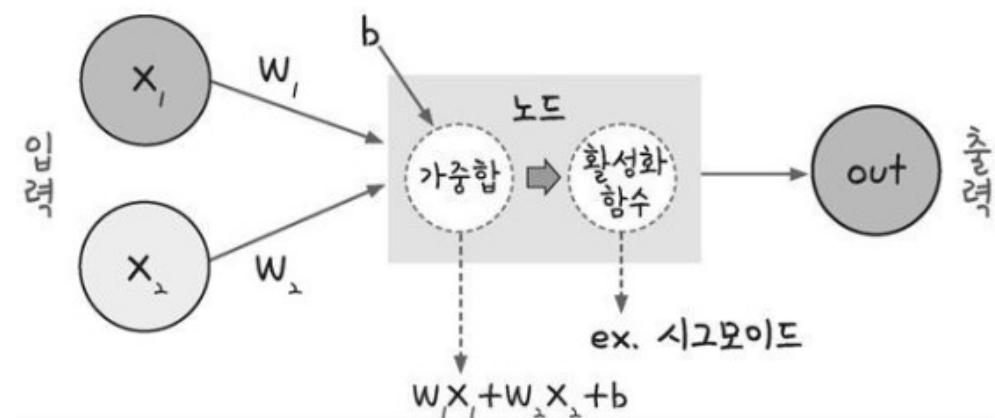
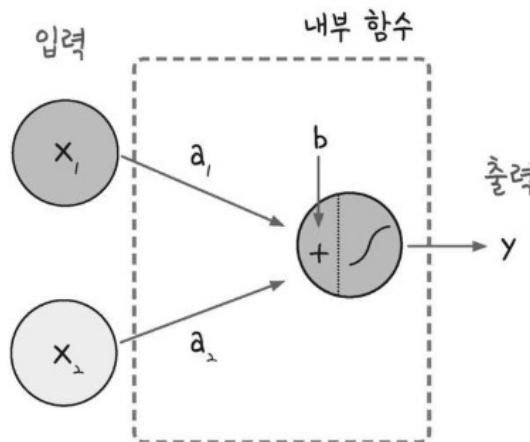
```
#여러 입력 값을 갖는 로지스틱 회귀 – 새로운 데이터에 대한 예측값 생성  
# [7, 6]은 각각 공부한 시간과 과외 수업 횟수
```

딥러닝 동작원리

▶ 퍼셉트론(perceptron)

- 뉴런과 뉴런이 서로 새로운 연결을 만들기도 하고 필요에 따라 위치를 바꾸는 것처럼, 여러 층의 퍼셉트론을 서로 연결시키고 복잡하게 조합하여 주어진 입력 값에 대한 판단을 하게 하는 것
- 퍼셉트론은 **입력 값과 활성화 함수를 사용해 출력 값을 다음으로 넘기는 가장 작은 신경망 단위**
- N개의 이진수가 하나의 뉴런을 통과해서 가중합 0보다 크면 활성화되는 가장 간단한 신경망 구조
- 초평면(hyperplane)으로 구분되는 두 개의 공간을 분리시키는 역할을 한다
- AND 게이트, OR 게이트를 만들 수 있다.

$$y = a_1x_1 + a_2x_2 + b$$



딥러닝 동작원리

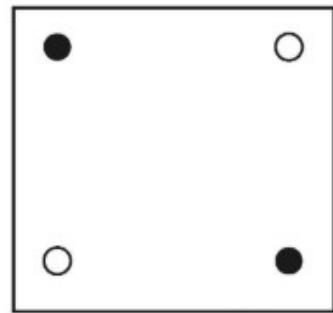
➤ 퍼셉트론(perceptron)

- 가중합 - 입력 값(x)과 가중치(w)의 곱을 모두 더한 값에 바이어스(b)를 더한 값
- 가중합의 결과를 놓고 1 또는 0을 출력해서 다음으로 보냅니다.
- 활성화 함수(activation function) - 0과 1을 판단하는 함수

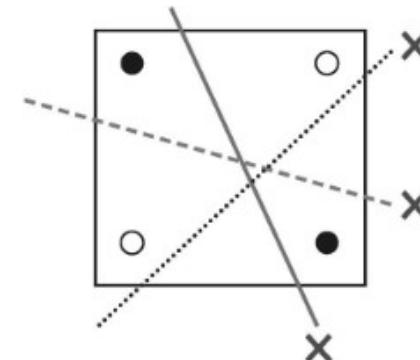
$$y = w \cdot x + b \quad (w\text{는 가중치, } b\text{는 바이어스})$$

딥러닝 동작원리

➤ XOR(exclusive OR) 문제



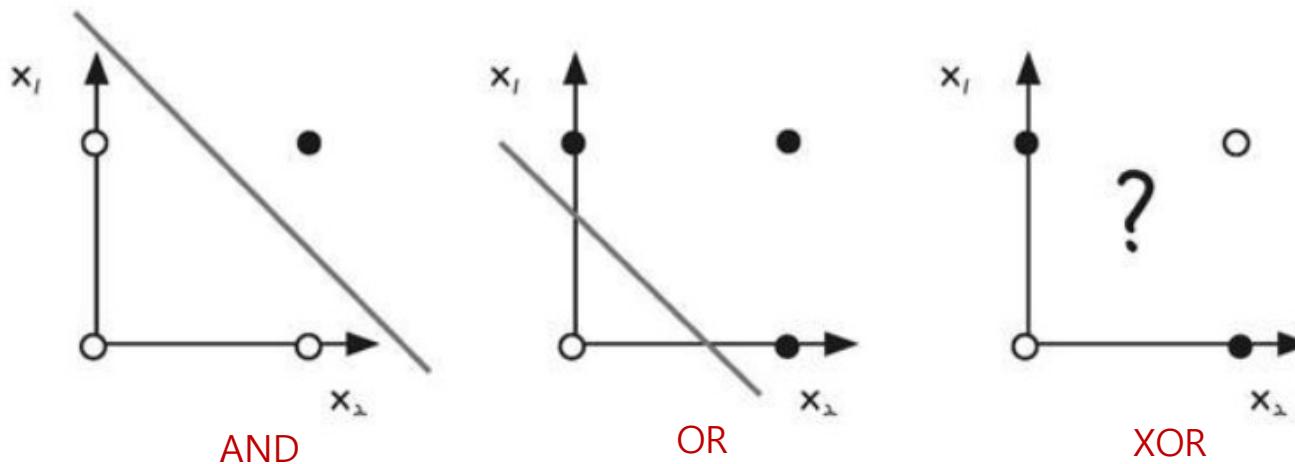
직선의 한쪽 편에는 검은점만 있고, 다른 한쪽에는 흰점만 있게끔 선을 그을 수 있을까요?.



딥러닝 동작원리

➤ XOR(exclusive OR) 문제

- XOR(exclusive OR) - 둘 중 하나만 1일 때 1이 출력
- 결괏값이 0이면 흰점으로, 1이면 검은점으로 각각 그래프로 좌표 평면에 표현



XOR 진리표

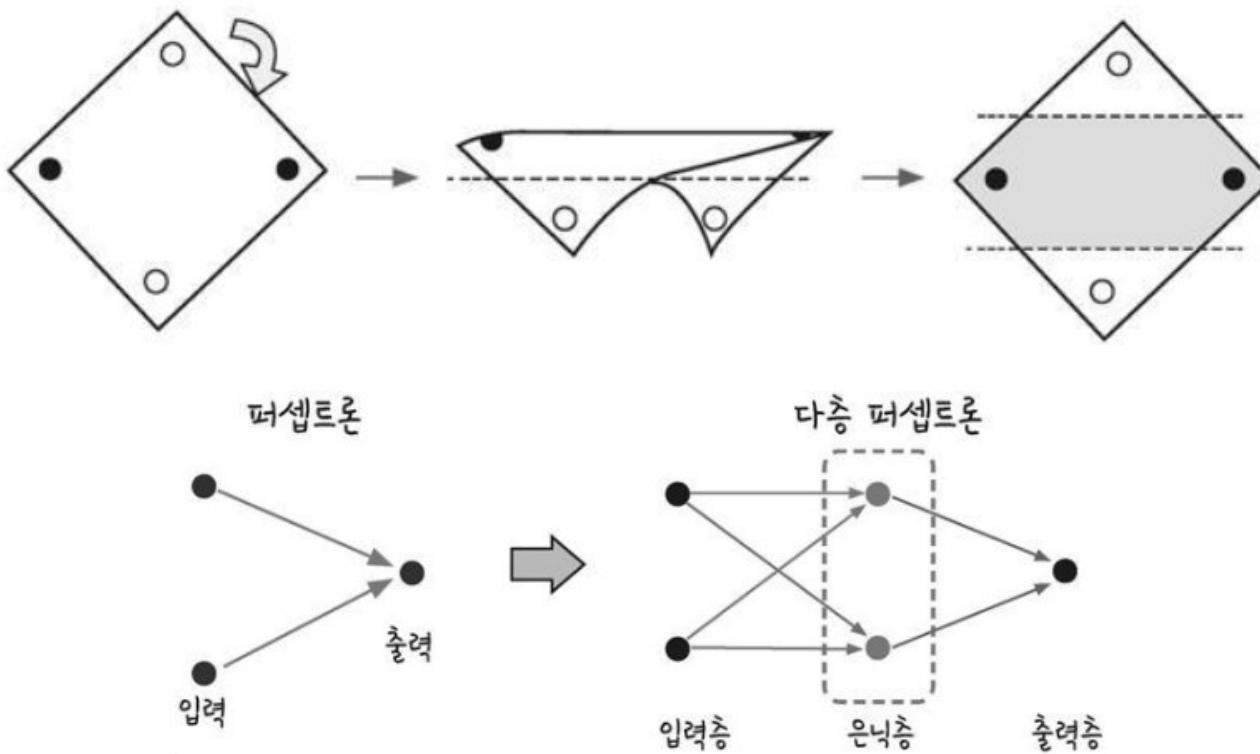
x_1	x_2	결괏값
0	0	0
0	1	1
1	0	1
1	1	0

AND와 OR 게이트는 직선을 그어 결괏값이 1인 값(검은점)을 구별할 수 있습니다.
XOR의 경우 선을 그어 구분할 수 없습니다.

딥러닝 동작원리

➤ XOR(exclusive OR) 문제

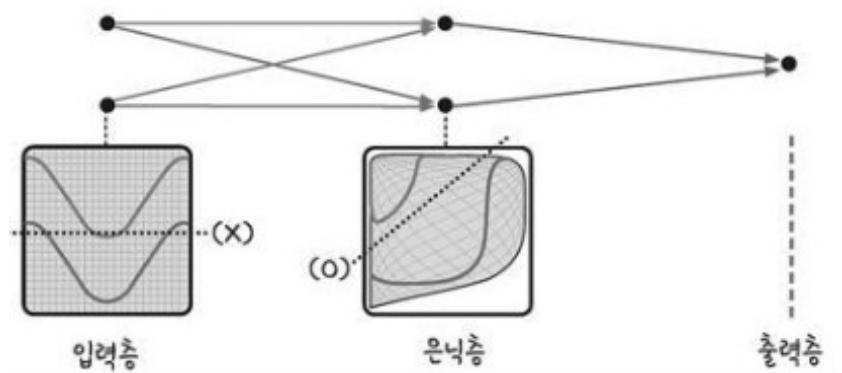
- 좌표 평면 자체에 변화를 주는 것
- XOR 문제를 해결하기 위해서 우리는 두 개의 퍼셉트론을 한 번에 계산할 수 있어야 합니다.
- 은닉층(hidden layer)



딥러닝 동작원리

▶ 퍼셉트론(perceptron)

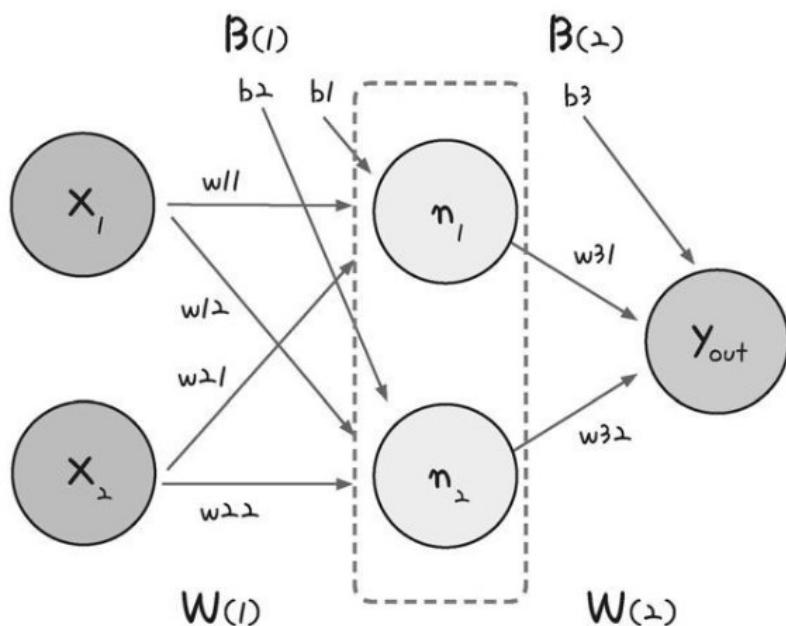
- 입력 값(input)을 놓고 파란색(위)과 빨간색(아래)의 영역을 구분한다고 할 때, 왼쪽은 어떤 직선으로도 이를 해결할 수 없습니다.
- 은닉층을 만들어 공간을 왜곡하면 두 영역을 가로지르는 선이 직선으로 바뀝니다
- 은닉층이 좌표 평면을 왜곡시키는 결과를 가져옵니다.



딥러닝 동작원리

▶ 다층 퍼셉트론(perceptron)

- 은닉층으로 퍼셉트론이 각각 자신의 가중치(w)와 바이어스(b) 값을 보내고
- 은닉층에서 모인 값이 한 번 더 시그모이드 함수(기호로 σ 라고 표시합니다)를 이용해 최종 값으로 결과를 보냅니다.
- 은닉층에 모이는 중간 정거장을 노드(node)라고 하며, (n1과 n2)
- 은닉층의 노드 n1과 n2의 값은 각각 단일 퍼셉트론의 값과 같습니다.



$$n_1 = \sigma(x_1 w_{11} + x_2 w_{21} + b_1)$$

$$n_2 = \sigma(x_1 w_{12} + x_2 w_{22} + b_2)$$

- 두식의 결괏값이 출력층으로 보내집니다.
- 출력층에서는 시그모이드 함수를 통해 y 값이 정해집니다.

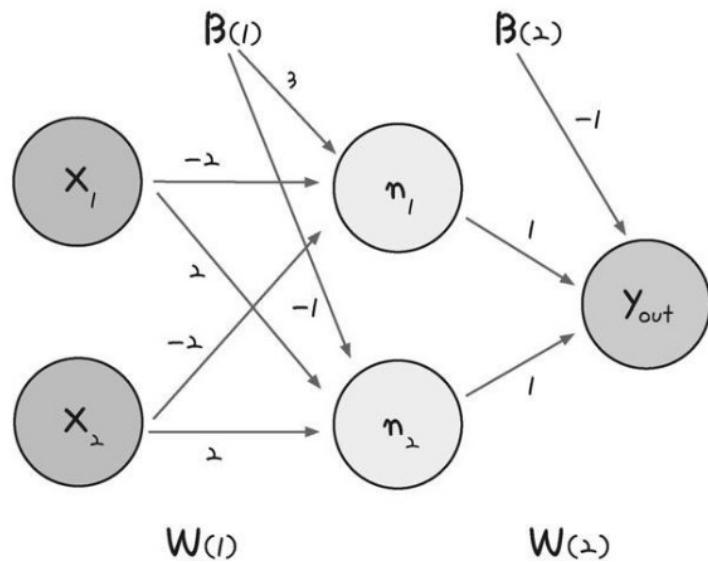
$$y_{out} = \sigma(n_1 w_{31} + n_2 w_{32} + b_3)$$

딥러닝 동작원리

➤ XOR 문제의 해결

$$W(1) = \begin{bmatrix} -2 & 2 \\ -2 & 2 \end{bmatrix} \quad B(1) = \begin{bmatrix} 3 \\ -1 \end{bmatrix}$$

$$W(2) = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad B(2) = [-1]$$



x_1	x_2	n_1	n_2	y_{out}	우리가 원하는 값
0	0	$\sigma(0 * (-2) + 0 * (-2) + 3) = 1$	$\sigma(0 * 2 + 0 * 2 - 1) = 0$	$\sigma(1 * 1 + 0 * 1 - 1) = 0$	0
0	1	$\sigma(0 * (-2) + 1 * (-2) + 3) = 1$	$\sigma(0 * 2 + 1 * 2 - 1) = 1$	$\sigma(1 * 1 + 1 * 1 - 1) = 1$	1
1	0	$\sigma(1 * (-2) + 0 * (-2) + 3) = 1$	$\sigma(1 * 2 + 0 * 2 - 1) = 1$	$\sigma(1 * 1 + 1 * 1 - 1) = 1$	1
1	1	$\sigma(1 * (-2) + 1 * (-2) + 3) = 0$	$\sigma(1 * 2 + 1 * 2 - 1) = 1$	$\sigma(0 * 1 + 1 * 1 - 1) = 0$	0

딥러닝 동작원리

- XOR 문제의 해결 코드 구현

```
import numpy as np

# 가중치와 바이어스

# 퍼셉트론

# NAND 게이트

# OR 게이트

# AND 게이트

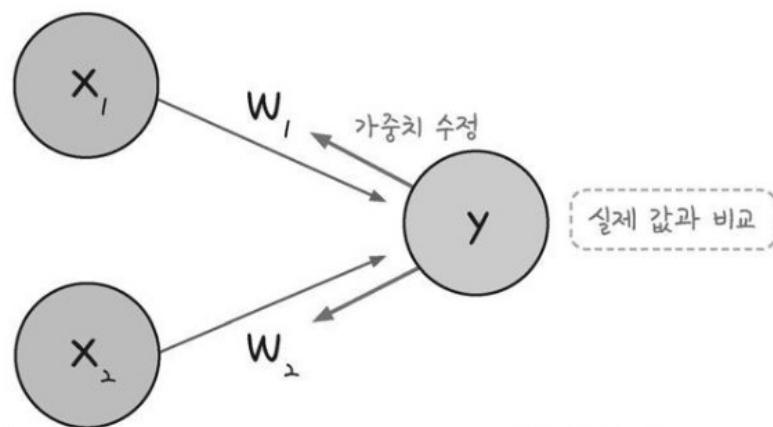
# XOR 게이트

# x1, x2 값을 번갈아 대입해 가며 최종값 출력
```

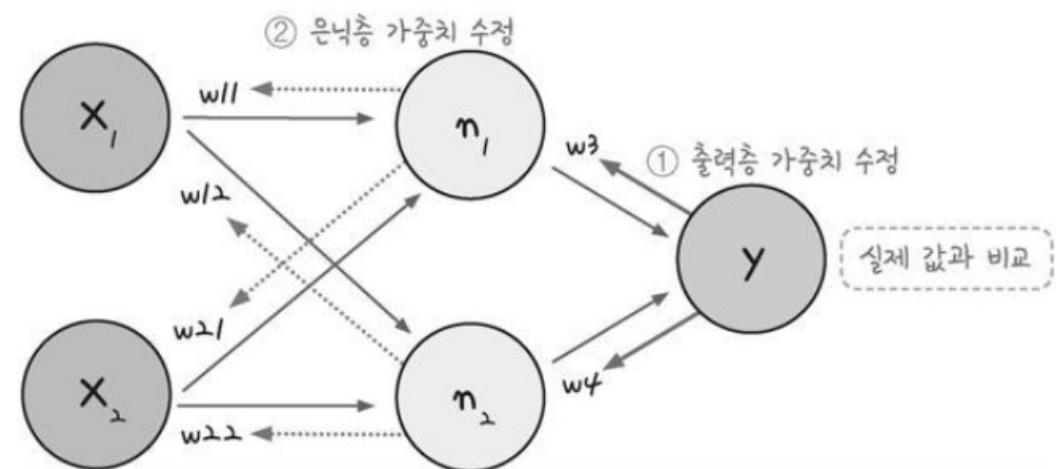
딥러닝 동작원리

➤ 오차 역전파

- 신경망 내부의 가중치는 오차 역전파 방법을 사용해 수정합니다.
- 임의의 가중치를 선언하고 최소 제곱법을 이용해 오차를 구한 뒤 이 오차가 최소인 지점으로 계속해서 조금씩 이동시킵니다.
- 오차가 최소가 되는 점(미분했을 때 기울기가 0이 되는 지점)이 우리가 알고자 하는 답입니다



단일 퍼셉트론에서의 오차 수정



다층 퍼셉트론에서의 오차 수정

딥러닝 동작원리

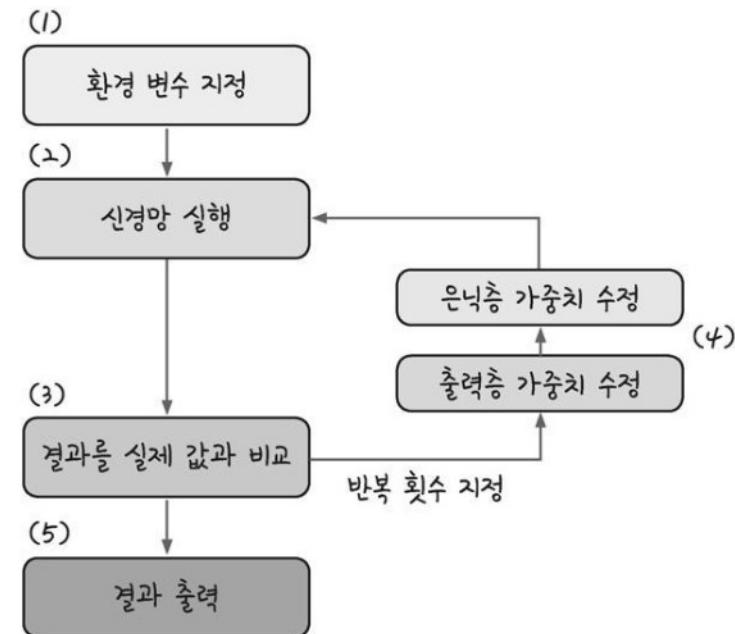
➤ 오차 역전파 (back propagation)

- 다층 퍼셉트론에서의 최적화 과정
- 가중치에서 기울기를 빼도 값의 변화가 없을 때까지 계속해서 가중치 수정 작업을 반복하는 것
- 출력층으로부터 하나씩 앞으로 되돌아가며 각 층의 가중치를 수정하는 방법
- 가중치를 수정하려면 미분 값, 즉 기울기가 필요합니다

새 가중치는 현 가중치에서 '가중치에 대한 기울기'를 뺀 값!

$$W(t+1) = Wt - \frac{\partial \text{오차}}{\partial W}$$

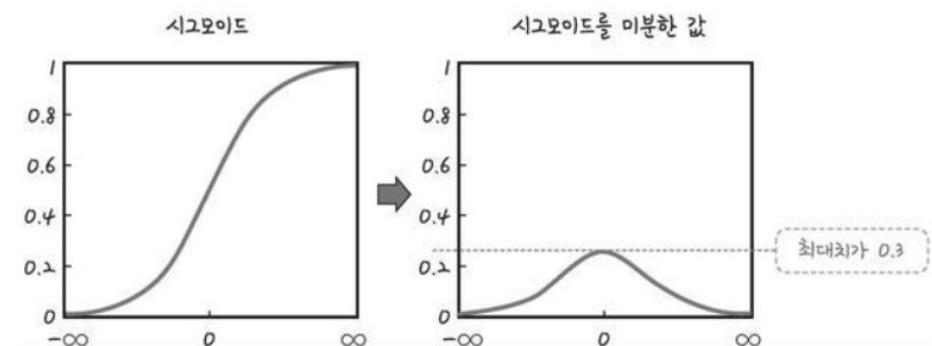
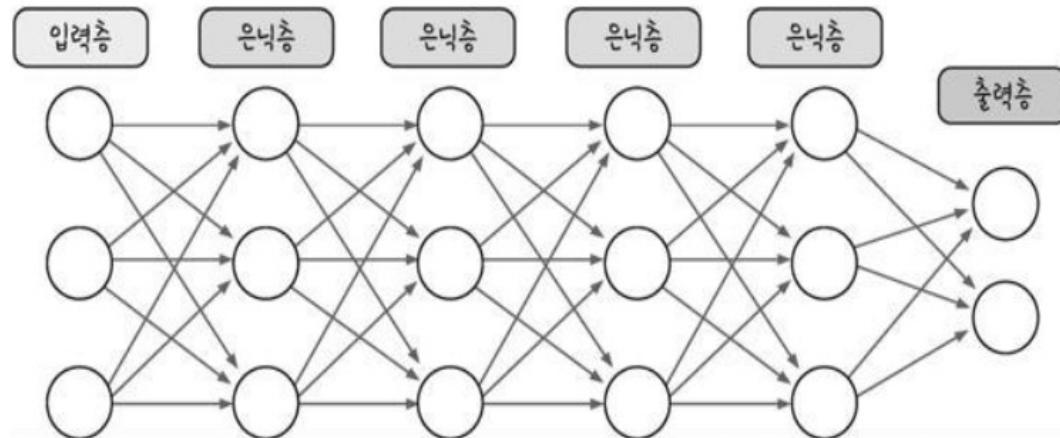
편미분 기호 ∂ (partial, 파셜)



딥러닝 동작원리

▶ 활성화 함수 - 기울기 소실 문제 해결

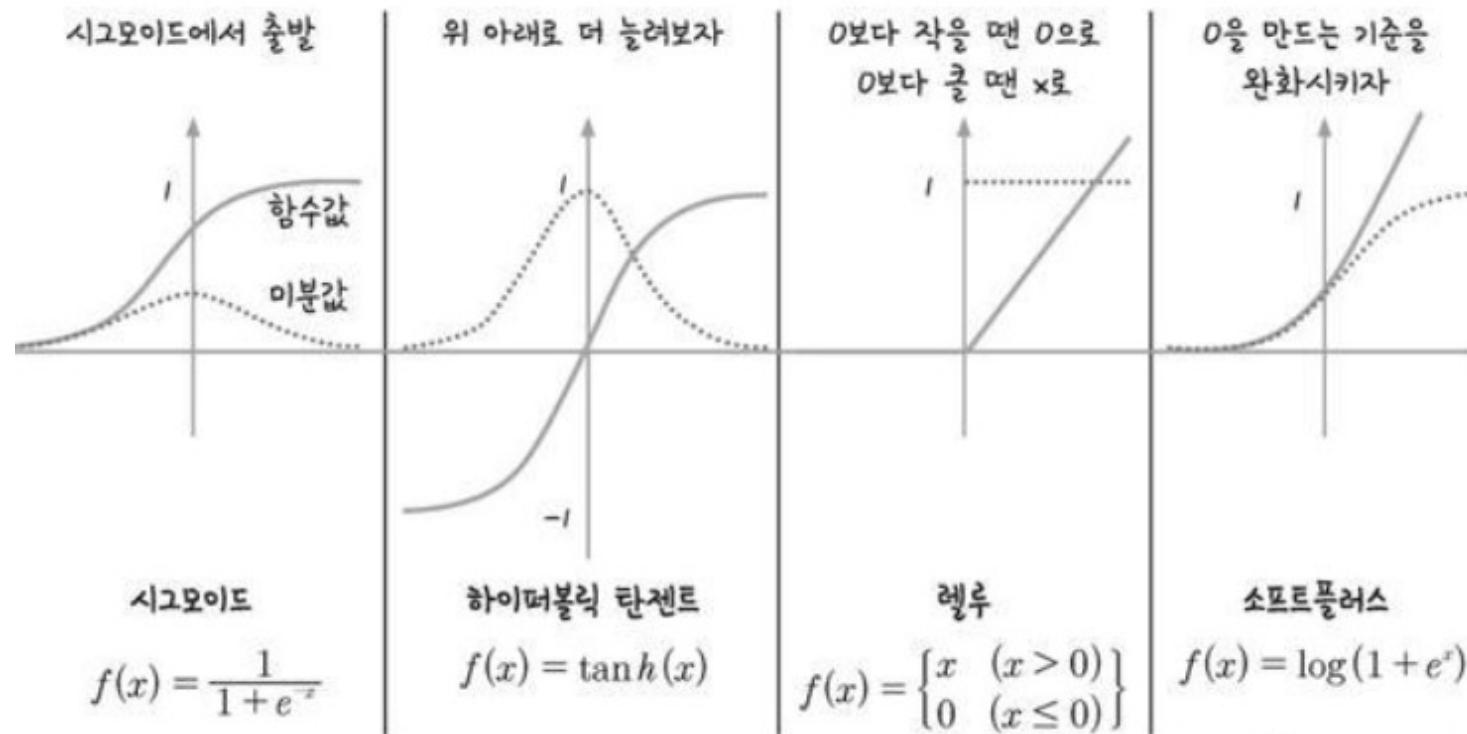
- 활성화 함수 시그모이드는 층이 늘어나면서 기울기가 중간에 0이 되어버리는 기울기 소실(vanishing gradient) 문제가 발생합니다.
- 시그모이드를 미분하면 최대치가 0.3입니다. 1보다 작으므로 계속 곱하다 보면 0에 가까워집니다.
- 시그모이드는 층을 거쳐 갈수록 기울기가 사라져 가중치를 수정하기가 어려워집니다.



딥러닝 동작원리

▶ 활성화 함수 - 기울기 소실 문제 해결

- 기울기가 사라지는 문제를 해결하기 위해 활성화 함수를 시그모이드가 아닌 여러 함수로 대체하기 시작

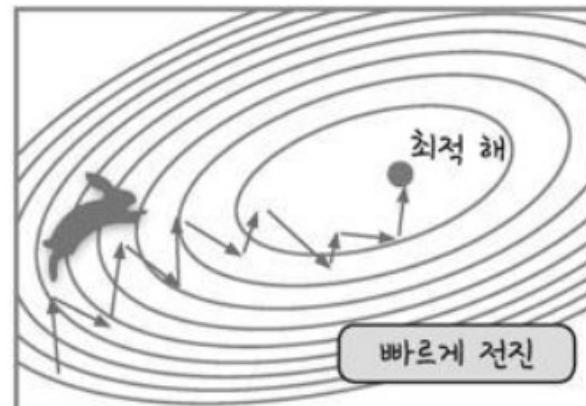
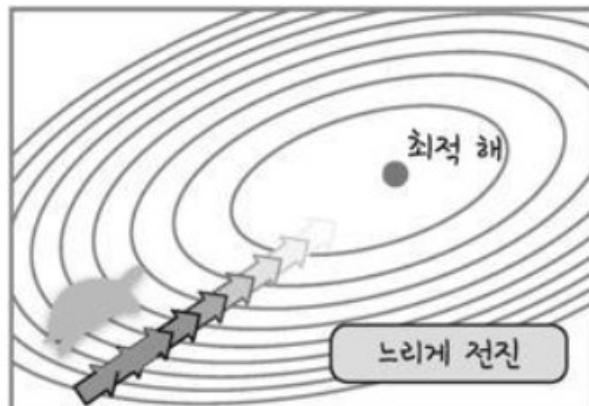


- 하이퍼볼릭 탄젠트(tanh) - 시그모이드 함수의 범위를 -1 에서 1 로 확장, 미분한 값의 범위가 함께 확장 되는 효과를 가져왔습니다. 여전히 1 보다 작은 값이 존재하므로 기울기 소실 문제는 사라지지 않습니다.
- 렐루(ReLU) - x 가 0 보다 작을 때는 모든 값을 0 으로 처리하고, 0 보다 큰 값은 x 를 그대로 사용하는 방법 x 가 0 보다 크기만 하면 미분 값이 1 이 됩니다. 따라서 여러 은닉층을 거치며 곱해지더라도 맨 처음 층까지 사라지지 않고 남아 있을 수 있습니다.

딥러닝 동작원리

➤ 확률적 경사 하강법(SGD)

- 경사 하강법은 정확하게 가중치를 찾아가지만, 한 번 업데이트할 때마다 전체 데이터를 미분해야 하므로 계산량이 매우 많다는 단점이 있습니다.
- 확률적 경사 하강법은 전체 데이터를 사용하는 것이 아니라, 랜덤하게 추출한 일부 데이터를 사용합니다. 일부 데이터를 사용하므로 더 빨리, 자주 업데이트를 하는 것이 가능해졌습니다.
- 확률적 경사 하강법은 중간 결과의 진폭이 크고 불안정해 보일 수도 있습니다.
- 속도가 빠르면서도 최적 해에 근사한 값을 찾아낸다는 장점 때문에 경사 하강법의 대안으로 사용되고 있습니다.



딥러닝 동작원리

➤ 모멘텀(momentum)

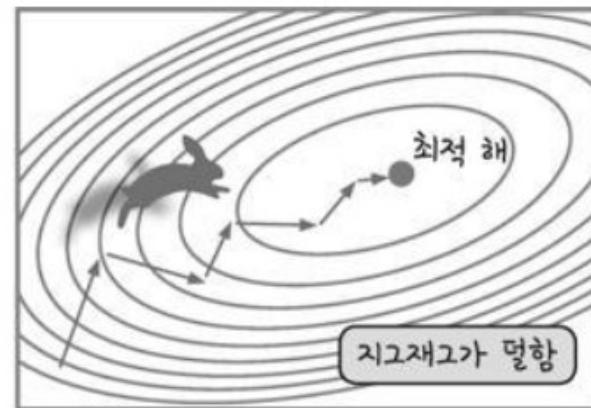


경사 하강법과 마찬가지로 매번 기울기를 구하지만, 이를 통해 **오차를 수정하기 전 바로 앞 수정 값과 방향(+, -)을 참고하여 같은 방향으로 일정한 비율만 수정되게 하는 방법**

수정 방향이 양수(+) 방향으로 한 번, 음수(-) 방향으로 한 번 지그재그로 일어나는 현상이 줄어들고, 이전 이동 값을 고려하여 일정 비율만큼만 다음 값을 결정하므로 관성의 효과를 낼 수 있습니다.



확률적 경사 하강법



모멘텀을 적용한 확률적 경사 하강법

딥러닝 동작원리

➤ 고급 경사 하강법의 케라스 사용법

고급 경사 하강법	개요	효과	케라스 사용법
확률적 경사 하강법(SGD)	랜덤하게 추출한 일부 데이터를 사용해 더 빨리, 자주 업데이트를 하게 하는 것	속도 개선	keras.optimizers.SGD(lr = 0.1) 케라스 최적화 함수를 이용
모멘텀(Momentum)	관성의 방향을 고려해 진동과 폭을 줄이는 효과	정확도 개선	keras.optimizers.SGD(lr = 0.1, momentum= 0.9) 모멘텀 계수를 추가
네스테로프 모멘텀(NAG)	변수의 업데이트가 잦으면 학습률을 적게 하여 이동 보폭을 조절하는 방법	보폭 크기 개선	keras.optimizers.Adagrad(lr = 0.01, epsilon = 1e - 6) 아다그라드 함수를 사용
알엠에스프롭(RMSProp)	아다그라드의 보폭 민감도를 보완한 방법	보폭 크기 개선	keras.optimizers.RMSprop(lr = 0.001, rho = 0.9, epsilon = 1e - 08, decay = 0.0) 알엠에스프롭 함수를 사용합니다.
아담(Adam)	모멘텀과 알엠에스프롭 방법을 합친 방법	정확도와 보폭 크기 개선	keras.optimizers.Adam(lr = 0.001, beta_1 = 0.9, beta_2 = 0.999, epsilon = 1e - 08, decay = 0.0) 아담 함수를 사용합니다.

딥러닝 동작원리

➤ 폐암 수술 환자의 생존율 예측하기 실습

```
# 딥러닝을 구동하는 데 필요한 케라스 함수를 불러옵니다.  
from keras.models import Sequential  
from keras.layers import Dense  
  
# 필요한 라이브러리를 불러옵니다.  
import numpy  
import tensorflow as tf  
  
# 실행할 때마다 같은 결과를 출력하기 위해 설정하는 부분입니다.  
seed = 0  
numpy.random.seed(seed)  
tf.set_random_seed(seed)  
  
# 준비된 수술 환자 데이터를 불러들입니다.  
Data_set = numpy.loadtxt("./dataset/ThoracicSurgery.csv",delimiter=',')  
  
# 환자의 기록과 수술 결과를 X와 Y로 구분하여 저장합니다.  
X = Data_set[:,0:17]  
Y = Data_set[:,17]
```

딥러닝 동작원리

➤ 폐암 수술 환자의 생존율 예측하기 실습

```
# 딥러닝 구조를 결정합니다(모델을 설정하고 실행).
#딥러닝의 구조를 짜고 층을 설정
# 첫 번째 은닉층에 input_dim을 적어 줌으로써 첫 번째 Dense가 은닉층 + 입력층의 역할을 겸합니다.
# 데이터에서 17개의 값을 받아 은닉층의 30개 노드로 보낸다
#activation : 출력층으로 전달할 때 사용할 활성화 함수
#출력층의 노드 수는 1개, 최종 출력 값에 사용될 활성화 함수

# 딥러닝을 실행합니다. (오차 함수 : 평균 제곱 오차 함수 사용)

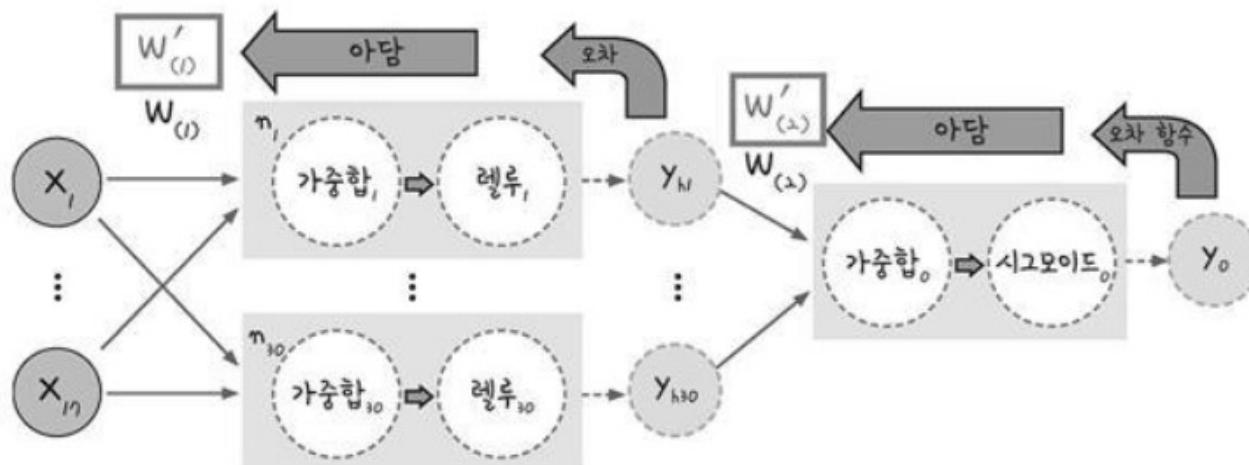
# 결과를 출력합니다.
```

- 평균 제곱 오차는 수렴하기까지 속도가 많이 걸린다는 단점이 있다
- 교차 엔트로피는 출력 값에 로그를 취해서, 오차가 커지면 수렴 속도가 빨라지고 오차가 작아지면 속도가 감소하게끔 만듭니다.
- metrics 함수는 모델이 컴파일될 때 모델 수행 결과를 나타내게끔 설정하는 부분입니다. 정확도를 측정하기 위해 사용되는 테스트 샘플을 학습 과정에서 제외시킴으로써 과적합 문제(over fitting, 특정 데이터에서는 잘 작동하나 다른 데이터에서는 잘 작동하지 않는 문제)를 방지하는 기능을 담고 있습니다.

딥러닝 동작원리

▶ 폐암 수술 환자의 생존율 예측하기 실습

- 학습 프로세스가 모든 샘플에 대해 한 번 실행되는 것을 1 epoch('에포크'라고 읽습니다)라고 합니다.
- epochs=1000은 각 샘플이 처음부터 끝까지 1,000번 재사용될 때까지 실행을 반복하라는 뜻입니다.
- batch_size는 샘플을 한 번에 몇 개씩 처리할지를 정하는 부분입니다.
- batch_size=10은 전체 470개의 샘플을 10개씩 끊어서 집어넣으라는 뜻이 됩니다.
- batch_size가 너무 크면 학습 속도가 느려지고, 너무 작으면 각 실행 값의 편차가 생겨서 전체 결괏값이 불안정해질 수 있습니다.



딥러닝 동작원리

➤ 오차 함수

평균 제곱 계열	mean_squared_error	평균 제곱 오차 $\text{mean}(\text{square}(yt - yo))$
	mean_absolute_error	평균 절대 오차(실제 값과 예측 값 차이의 절댓값 평균) $\text{mean}(\text{abs}(yt - yo))$
	mean_absolute_percentage_error	평균 절대 백분율 오차(절댓값 오차를 절댓값으로 나눈 후 평균) $\text{mean}(\text{abs}(yt - yo)/\text{abs}(yt))$ (단, 분모 ≠ 0)
	mean_squared_logarithmic_error	평균 제곱 로그 오차(실제 값과 예측 값에 로그를 적용한 값의 차 이를 제곱한 값의 평균) $\text{mean}(\text{square}((\log(yo) + 1) - (\log(yt) + 1)))$
교차 엔트로피 계열 분류 문제에서 많이 사용	categorical_crossentropy	범주형 교차 엔트로피(일반적인 분류)
	binary_crossentropy	이항 교차 엔트로피(두 개의 클래스 중에서 예측할 때) 예측 값이 참과 거짓 둘 중 하나인 형식일 때 사용

딥러닝 동작원리

➤ 피마 인디언 데이터 분석 실습

- 피마 인디언은 1950년대까지만 해도 비만인 사람이 단 한 명도 없는 민족이었습니다.
- 생존하기 위해 영양분을 체내에 저장하는 뛰어난 능력을 물려받은 인디언들이 미국의 기름진 패스트푸드 문화를 만나면서
- 지금은 전체 부족의 60%가 당뇨, 80%가 비만으로 고통받고 있습니다.

샘플	속성						클래스
	정보 1	정보 2	정보 3	...	정보 8	당뇨병 여부	
1번째 인디언	6	148	72	...	50	1	
2번째 인디언	1	85	66	...	31	0	
3번째 인디언	8	183	64	...	32	1	
...	
768번째 인디언	1	93	70	...	23	0	

샘플 수: 768

- 속성: 8
 - 정보 1 (pregnant): 과거 임신 횟수
 - 정보 2 (plasma): 포도당 부하 검사 2시간 후 공복 혈당 농도(mm Hg)
 - 정보 3 (pressure): 확장기 혈압(mm Hg)
 - 정보 4 (thickness): 삼두근 피부 주름 두께(mm)
 - 정보 5 (insulin): 혈청 인슐린(2-hour, mu U/ml)
 - 정보 6 (BMI): 체질량 지수(BMI, weight in kg/(height in m)²)
 - 정보 7 (pedigree): 당뇨병 가족력
 - 정보 8 (age): 나이
- 클래스: 당뇨(1), 당뇨 아님(0)

딥러닝 동작원리

➤ 피마 인디언 데이터 분석 실습

```
from keras.models import Sequential
from keras.layers import Dense
import numpy
import tensorflow as tf

seed = 0
numpy.random.seed(seed)          # seed 값 생성
tf.set_random_seed(seed)

dataset = numpy.loadtxt("./dataset/pima-indians-diabetes.csv", delimiter=",")      # 데이터 로드
X = dataset[:,0:8]
Y = dataset[:,8]

# 모델의 설정

# 모델 컴파일
# 모델 실행
# 결과 출력
```

딥러닝 동작원리

➤ 다중 분류 분석 실습

- 아이리스 데이터의 샘플, 속성, 클래스 구분

샘플	속성				클래스
	정보 1	정보 2	정보 3	정보 4	
1번째 아이리스	5.1	3.5	4.0	0.2	Iris-setosa
2번째 아이리스	4.9	3.0	1.4	0.2	Iris-setosa
3번째 아이리스	4.7	3.2	1.3	0.3	Iris-setosa
...
150번째 아이리스	5.9	3.0	5.1	1.8	Iris-virginica

샘플 수: 150

- 속성 수: 4
 - 정보 1: 꽃받침 길이 (sepal length, 단위: cm)
 - 정보 2: 꽃받침 넓이 (sepal width, 단위: cm)
 - 정보 3: 꽃잎 길이 (petal length, 단위: cm)
 - 정보 4: 꽃잎 넓이 (petal width, 단위: cm)
- 클래스: Iris-setosa, Iris-versicolor, Iris-virginica

딥러닝 동작원리

➤ 다중 분류 분석 실습

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
df = pd.read_csv('../dataset/iris.csv', names = ["sepal_length", "sepal_width", "petal_length", "petal_width",
"species"])
print(df.head())

sns.pairplot(df, hue='species') #속성별 연관성 파악
plt.show()

dataset = df.values
X = dataset[:,0:4].astype(float)
Y_obj = dataset[:,4]
```

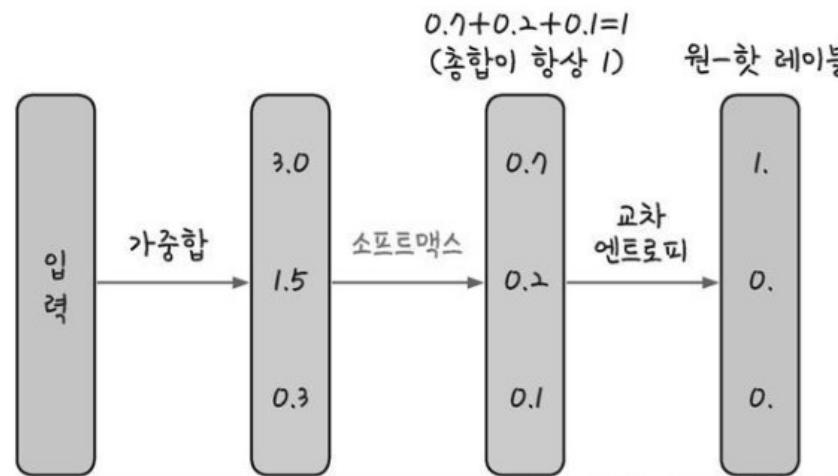
딥러닝 동작원리

➤ 다중 분류 분석 실습

- 소프트맥스(softmax) - 총합이 1인 형태로 바꿔서 계산해 주는 함수

합계가 1인 형태로 변환하면 큰 값이 두드러지게 나타나고 작은 값은 더 작아집니다.

이 값이 교차 엔트로피를 지나 [1., 0., 0.]으로 변화하게 되면 우리가 원하는 원-핫 인코딩 값, 즉 하나만 1이고 나머지는 모두 0인 형태로 전환시킬 수 있습니다.



딥러닝 동작원리

➤ 다중 분류 분석 실습

```
from keras.models import Sequential
from keras.layers.core import Dense
import numpy
import tensorflow as tf

seed = 0
numpy.random.seed(seed) # seed 값 설정
tf.set_random_seed(seed)

model = Sequential() # 모델의 설정

#최종 출력 값이 3개 중 하나여야 하므로 출력층에 해당하는 Dense의 노드 수를 3으로 설정

# 모델 컴파일(다중 분류에 적절한 오차 함수인 categorical_crossentropy를 사용, 최적화 함수로 adam 사용)

# 모델 실행(한 번에 입력되는 값은 1개, 전체 샘플이 50회 반복될 때까지 실험을 진행

# 결과 출력
```

딥러닝 동작원리

➤ 초음파 광물 예측 분석 실습

Range Index: 208 entries, 0 to 207			
Data columns (total 61 columns):			
0	208	non-null	float64
1	208	non-null	float64
2	208	non-null	float64
3	208	non-null	float64
4	208	non-null	float64
5	208	non-null	float64
...
58	208	non-null	float64
59	208	non-null	float64
60	208	non-null	object

Dtypes: float64(60), object(1)
memory usage: 99.2+ KB

	0	1	2	3	...	59	60
0	0.02	0.0371	0.0428	0.0207	...	0.0032	R
1	0.0453	0.0523	0.0843	0.0689	...	0.0044	R
2	0.0262	0.0582	0.1099	0.1083	...	0.0078	R
3	0.01	0.0171	0.0623	0.0205	...	0.0117	R
4	0.0762	0.0666	0.0481	0.0394	...	0.0094	R

딥러닝 동작원리

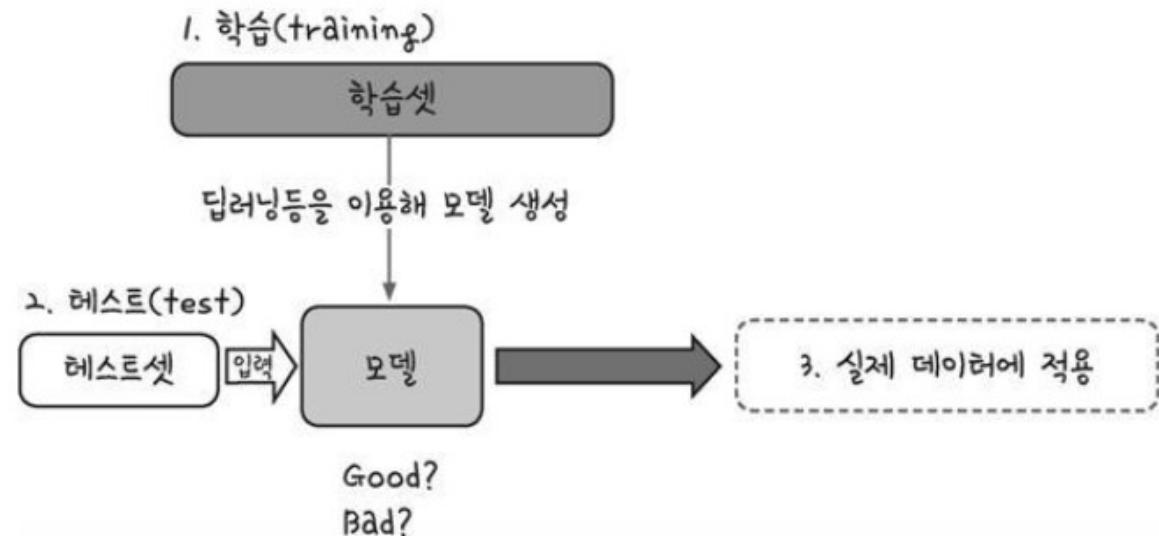
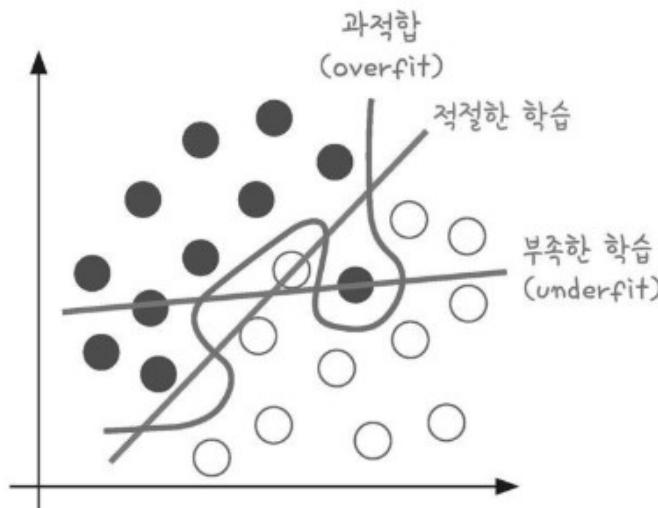
➤ 초음파 광물 예측 분석 실습

```
from keras.models import Sequential
from keras.layers.core import Dense
from sklearn.preprocessing import LabelEncoder
import pandas as pd
import numpy
import tensorflow as tf
seed = 0
numpy.random.seed(seed)      # seed 값 설정
tf.set_random_seed(seed)
df = pd.read_csv('../dataset/sonar.csv', header=None)      # 데이터 입력
dataset = df.values
X = dataset[:,0:60]
Y_obj = dataset[:,60]
```

딥러닝 동작원리

➤ 과적합 피하기

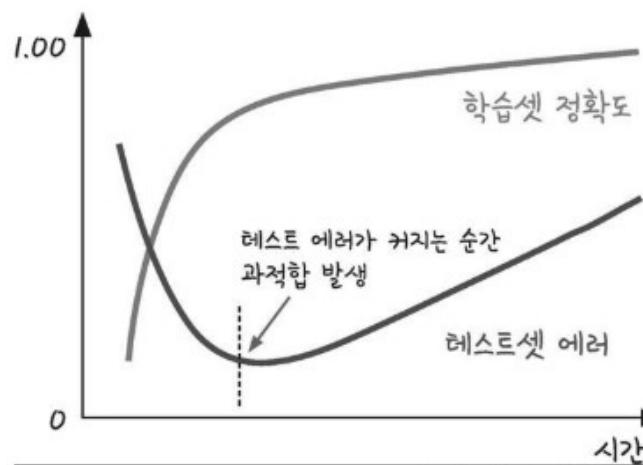
- 과적합(over fitting) - 모델이 학습 데이터셋 안에서는 일정 수준 이상의 예측 정확도를 보이지만, 새로운 데이터에 적용하면 잘 맞지 않는 것
- 과적합은 층이 너무 많거나 변수가 복잡해서 발생하기도 하고 테스트셋과 학습셋이 중복될 때 생기기도 합니다
- 과적합을 방지 방법 - 학습을 하는 데이터셋과 이를 테스트할 데이터셋을 완전히 구분한 다음 학습과 동시에 테스트를 병행하며 진행



딥러닝 동작원리

➤ 과적합 피하기

- 머신러닝의 최종 목적 - 과거의 데이터를 토대로 새로운 데이터를 예측하는 것
새로운 데이터에 사용할 모델을 만드는 것
- 학습셋만 가지고 평가할 때, 층을 더하거나 에포크(epoch) 값을 높여 실행 횟수를 늘리면 정확도가 계속해서 올라갈 수 있습니다.
- 학습 데이터셋만으로 평가한 예측 성공률이 테스트셋에서도 그대로 나타나지는 않습니다.
- 학습이 깊어져서 학습셋 내부에서의 성공률은 높아져도 테스트셋에서는 효과가 없다면 과적합이 일어나고 있는 것입니다.



온닉총 수의 변화	학습셋의 예측률	테스트셋의 예측률
0	79.3	73.1
2	96.2	85.7
3	98.1	87.6
6	99.4	89.3
12	99.8	90.4
24	100	89.2

딥러닝 동작원리

➤ 초음파 광물 예측 분석(학습셋과 테스트셋 구분) 실습

```
from keras.models import Sequential
from keras.layers.core import Dense
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy
import tensorflow as tf

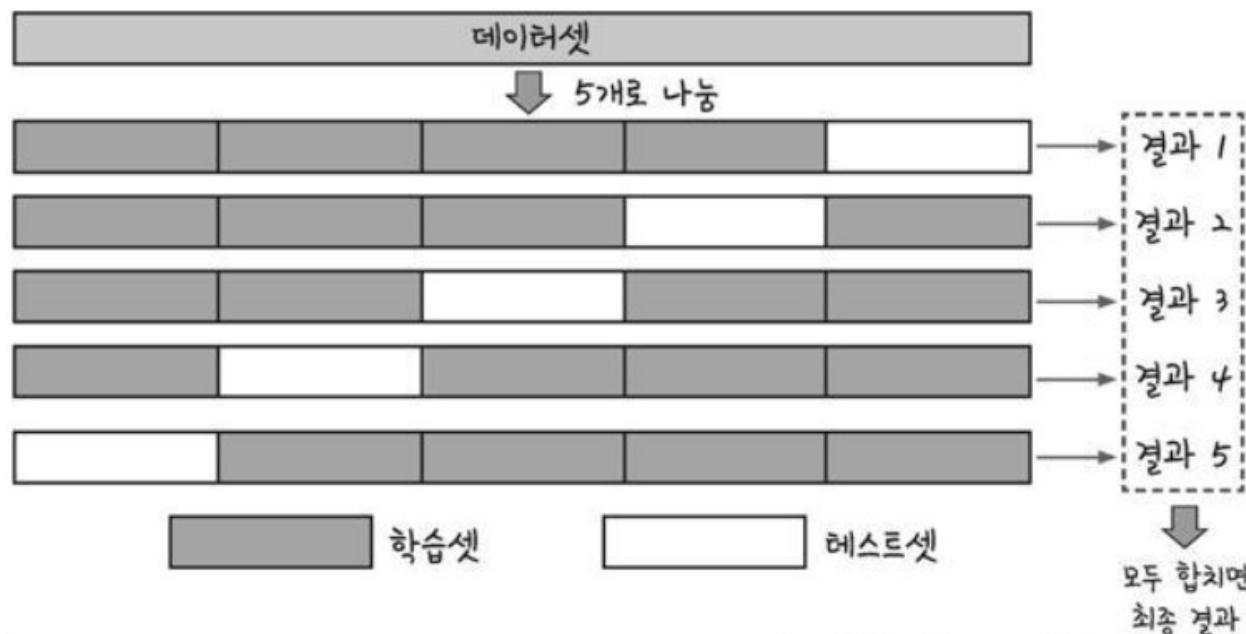
seed = 0
numpy.random.seed(seed) # seed 값 설정
tf.set_random_seed(seed)

df = pd.read_csv('../dataset/sonar.csv', header=None)
dataset = df.values
X = dataset[:,0:60]
Y_obj = dataset[:,60]
```

딥러닝 동작원리

➤ k겹 교차 검증

- k겹 교차 검증(k-fold cross validation) - k겹 교차 검증이란 데이터셋을 여러 개로 나누어 하나씩 테스트 셋으로 사용하고 나머지를 모두 합해서 학습셋으로 사용하는 방법
- 데이터가 충분치 않은 경우, 데이터의 100%를 테스트셋으로 사용할 수 있습니다.



딥러닝 동작원리

➤ 초음파 광물 예측 분석 - k겹 교차 검증

10개의 파일로 쪼개 테스트하는 10-fold cross validation을 실시하도록 n_fold의 값을 10으로 설정한 뒤 StratifiedKFold() 함수에 적용했습니다. 그런 다음 모델을 만들고 실행하는 부분을 for 구문으로 묶어 n_fold만큼 반복되게 합니다.

```
from sklearn.model_selection import StratifiedKFold
```

```
n_fold = 10
```

```
skf = StratifiedKFold(n_splits=nfold, shuffle=True, randomstate=seed)
```

```
accuracy = [] # 빈 accuracy 배열
```

```
for train, test in skf.split(X, Y): # 모델의 설정, 컴파일, 실행
```

```
    model = Sequential()
```

```
    model.add(Dense(24, input_dim=60, activation='relu'))
```

```
    model.add(Dense(10, activation='relu'))
```

```
    model.add(Dense(1, activation='sigmoid'))
```

```
    model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
```

```
    model.fit(X[train], Y[train], epochs=100, batch_size=5)
```

```
    k_accuracy = "%." + str(n_fold) + "f" % (model.evaluate(X[test], Y[test])[1])
```

```
    accuracy.append(k_accuracy)
```

```
# 결과 출력
```

```
print("n %.f fold accuracy:" % n_fold, accuracy)
```

딥러닝 동작원리

➤ 와인의 종류 예측 분석

- 총 6497개의 샘플
- 13개의 속성

0	주석산 농도	7	밀도
1	아세트산 농도	8	pH
2	구연산 농도	9	황산칼륨 농도
3	잔류 당분 농도	10	알코올 도수
4	염화나트륨 농도	11	와인의 맛(0~10등급)
5	유리 아황산 농도	12	class (1: 레드와인, 0: 화이트와인)
6	총 아황산 농도		

딥러닝 동작원리

➤ 와인의 종류 예측 분석

```
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import ModelCheckpoint, EarlyStopping
import pandas as pd
import numpy
import tensorflow as tf
import matplotlib.pyplot as plt

seed = 0
numpy.random.seed(seed) # seed 값 설정
tf.set_random_seed(seed)

df_pre = pd.read_csv('../dataset/wine.csv', header=None)
df = df_pre.sample(frac=1) #rac = 1 지정은 원본 데이터의 100%를 불러오라는 의미
dataset = df.values
X = dataset[:,0:12]
Y = dataset[:,12]
```

딥러닝 동작원리

➤ 와인의 종류 예측 분석 – 모델 업데이트

```
#에포크(epoch)마다 모델의 정확도를 함께 기록하면서 저장
```

```
from keras.callbacks import ModelCheckpoint
```

```
.....
```

```
# 모델 컴파일
```

```
# 모델 저장 폴더 설정
```

```
#테스트 오차는 케라스 내부에서 val_loss, 학습 정확도는 acc, 테스트셋 정확도는 val_acc, 학습셋 오차는 loss로 각각 기록됩니다
```

```
# 모델 저장 조건 설정
```

```
# 모델 실행 및 저장
```

딥러닝 동작원리

➤ 와인의 종류 예측 분석 – 그래프 표현

```
import matplotlib.pyplot as plt  
.....  
  
#sample() 함수를 이용하여 전체 샘플 중 15%만 불러오게 하고, 배치 크기는 500으로 늘려 한 번 딥러닝을 가동할  
때 더 많이 입력되게끔 했습니다. 불러온 샘플 중 33%는 분리하여 테스트셋으로 사용하였습니다.  
# 모델 실행 및 저장  
  
# y_vloss에 테스트셋(33%)으로 실험 결과의 오차 값을 저장  
# y_acc에 학습셋(67%)으로 측정한 정확도의 값을 저장  
  
# x 값을 지정하고 정확도를 파란색으로, 오차를 빨간색으로 표시  
# 학습이 진행될수록 학습셋의 정확도는 올라가지만 과적합으로 인해 테스트셋의 실험 결과는 점점 나빠지게 됩니다.  
# 케라스에는 학습이 진행되어도 테스트셋 오차가 줄지 않으면 학습을 멈추게 하는 함수가 있습니다.  
  
# 자동 중단 설정(EarlyStopping()) 함수에 모니터할 값과 테스트 오차가 좋아지지 않아도 몇 번까지 기다릴지를 정  
합니다.)  
# 모델 실행  
# 결과 출력
```

딥러닝 동작원리

➤ 선형 회귀 적용- 보스턴 집값 예측 분석

- 1978년, 집값에 가장 큰 영향을 미치는 것이 '깨끗한 공기'라는 연구 결과가 하버드대학교 도시개발학과에서 발표
- 집값의 변동에 영향을 미치는 여러 가지 요인을 모아서 환경과 집값의 변동을 보여주는 데이터셋 - 머신러닝의 선형 회귀를 테스트하는 가장 유명한 데이터로 쓰이고 있음
- 수치를 예측하는 문제 (선형 회귀 문제)
- 머신러닝 혹은 딥러닝을 위해 주어진 데이터의 답을 구하는 문제 (여러 개 중에 정답을 맞히거나, 가격, 성적 같은 수치를 맞히는 것)
- 506 entries, 14 columns (13개의 속성과 1개의 클래스로 이루어졌음)

딥러닝 동작원리

➤ 선형 회귀 적용- 보스턴 집값 예측 분석

컬럼(속성명)	설명
CRIM	인구 1인당 범죄 발생 수
ZN	25,000평방 피트 이상의 주거 구역 비중
INDUS	소매업 외 상업이 차지하는 면적 비율
CHAS	찰스강 위치 변수(1: 강 주변, 0: 이외)
NOX	일산화질소 농도
RM	집의 평균 방 수
AGE	1940년 이전에 지어진 비율
DIS	5가지 보스턴 시 고용 시설까지의 거리
RAD	순환고속도로의 접근 용이성
TAX	\$10,000당 부동산 세율 총계
PTRATIO	지역별 학생과 교사 비율
B	지역별 흑인 비율
LSTAT	급여가 낮은 직업에 종사하는 인구 비율(%)
	가격(단위: \$1,000)

딥러닝 동작원리

➤ 선형 회귀 적용- 보스턴 집값 예측 분석

- 선형 회귀 데이터는 마지막에 참과 거짓을 구분할 필요가 없기 때문에 출력층에 활성화 함수를 지정할 필요도 없습니다.
- 모델의 학습이 어느 정도 되었는지 확인하기 위해 예측 값과 실제 값을 비교하는 부분을 추가합니다.

```
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
import numpy
import pandas as pd
import tensorflow as tf

seed = 0
numpy.random.seed(seed) # seed 값 설정
tf.set_random_seed(seed)

df = pd.read_csv("../dataset/housing.csv", delim_whitespace=True, header=None)
dataset = df.values
X = dataset[:,0:13]
Y = dataset[:,13]
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=seed)
```

딥러닝 동작원리

➤ MNIST 손글씨 인식하기

- MNIST 데이터셋은 미국 국립표준기술원(NIST)이 고등학생과 인구조사국 직원 등이 쓴 손글씨를 이용해 만든 데이터로 구성
- 70,000개의 글자 이미지에 각각 0부터 9까지 이름표를 붙인 데이터셋
- 딥러닝을 이용 손글씨 이미지 분류, 정확률 예측 실습



- 이미지 데이터를 X
- 이미지에 0~9까지 붙인 이름표를 Y_class
- 70,000개의 이미지 중 60,000개를 학습용, 10,000개를 테스트용으로 구분

딥러닝 동작원리

➤ MNIST 손글씨 인식하기

- 이미지는 가로 $28 \times$ 세로 $28 =$ 총 784개의 픽셀로 이루어져 있습니다.
 - 각 픽셀은 밝기 정도에 따라 0부터 255까지의 등급을 매깁니다.
 - 흰색 배경이 0이라면 글씨가 들어간 곳은 1~255까지 숫자 중 하나로 채워져 긴 행렬로 이루어진 하나의 집합으로 변환됩니다.

```
for x in X_train[0]:  
    for i in x:  
        sys.stdout.write('%d\t' % i)  
    sys.stdout.write('\n')
```

딥러닝 동작원리

➤ MNIST 손글씨 인식하기: 데이터 전처리

- $28 \times 28 = 784$ 개의 속성을 이용해 0~9까지 10개 클래스 중 하나를 맞히는 분류 문제가 됩니다.
- Reshape() 함수를 사용하여 가로 28, 세로 28의 2차원 배열을 784개의 1차원 배열로 바꿔 주어야 합니다.
- 케라스는 데이터를 0에서 1 사이의 값으로 변환해야 구동할 때 최적의 성능을 보이므로 0~255 사이의 값으로 이루어진 값을 0~1 사이의 값으로 값을 255로 나누어 데이터 정규화(normalization) 합니다.
- Y_class를 np_utils.to_categorical(클래스, 클래스의 개수)를 사용하여 원-핫 인코딩 합니다.

```
from keras.datasets import mnist
from keras.utils import np_utils
import numpy
import sys
import tensorflow as tf

seed = 0
numpy.random.seed(seed) # seed 값 설정
tf.set_random_seed(seed)

# MNIST 데이터셋 불러오기
(X_train, Y_class_train), (X_test, Y_class_test) = mnist.load_data()
print("학습셋 이미지 수 : %d 개" % (X_train.shape[0]))
print("테스트셋 이미지 수 : %d 개" % (X_test.shape[0]))
```

딥러닝 동작원리

➤ MNIST 손글씨 인식하기: 딥러닝 실행

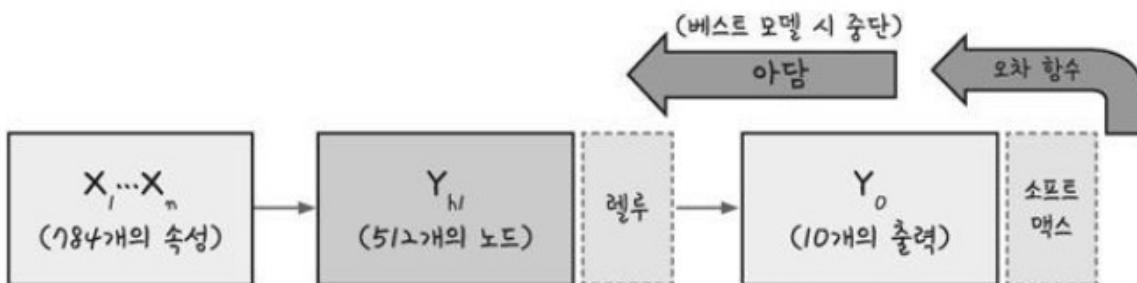
- 딥러닝을 실행하고자 프레임을 설정(총 784개의 속성, 10개의 클래스)
- 입력 값(input_dim)이 784개, 은닉층이 512개 , 출력이 10개인 모델
- 활성화 함수로 은닉층에서는 relu를, 출력층에서는 softmax를 사용
- 모델의 성과를 저장하고 모델의 최적화 단계에서 학습을 자동 중단하게끔 설정
- 샘플링 200개, 30번 학습 설정
- 테스트셋으로 최종 모델의 성과를 측정, 출력
- 학습셋의 정확도 대신 학습셋의 오차를 그래프로 표현 (학습셋의 오차= 1- 학습셋의 정확도)

```
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import ModelCheckpoint,EarlyStopping
import matplotlib.pyplot as plt
Import os

# 모델 프레임 설정
model = Sequential()
model.add(Dense(512, input_dim=784, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

딥러닝 동작원리

- MNIST 손글씨 인식하기: 딥러닝 실행



딥러닝 동작원리

➤ 컨볼루션 신경망(Convolutional Neural Network, CNN)

- 기본 딥러닝 프레임에 이미지 인식 분야에서 강력한 성능을 보이는 신경망
- 입력된 이미지에서 다시 한번 특징을 추출하기 위해 마스크(필터, 윈도 또는 커널이라고도 함)를 도입하는 기법
- 입력층과 출력층 사이의 중간층(은폐층)에 합성곱과 풀링층을 배치한 것
- 이미지를 흐리게 만들거나 경계를 강조하는 작업을 함
- 합성곱층과 풀링층에서는 해상도를 낮추거나 샘플링하는 처리를 계속 반복함

1	0	1	0
0	1	1	0
0	0	1	1
0	0	1	0

입력된 이미지

x1	x0
x0	x1

2x2 마스크 (가중치)

1x1	0x0	1	0
0x0	1x1	1	0
0	0	1	1
0	0	1	0

마스크 (가중치) 적용 새로운 값 추출

$$(1 \times 1) + (0 \times 0) + (0 \times 0) + (1 \times 1) = 2$$

딥러닝 동작원리

➤ 컨볼루션 신경망(Convolutional Neural Network, CNN)

$I \times I$	0×0	I	0
0×0	$I \times I$	I	0
0	0	I	I
0	0	I	0

I	$0 \times I$	$I \times 0$	0
0	$I \times 0$	$I \times I$	0
0	0	I	I
0	0	I	0

I	0	$I \times I$	0×0
0	I	$I \times 0$	$0 \times I$
0	0	I	I
0	0	I	0

I	0	I	0
$0 \times I$	$I \times 0$	I	0
0×0	$0 \times I$	I	I
0	0	I	0

I	0	I	0
0	$I \times I$	$I \times 0$	0
0	0×0	$I \times I$	I
0	0	I	0

I	0	I	0
0	I	$I \times I$	0
0	0×0	$I \times 0$	$I \times I$
0	0	I	0

I	0	I	0
0	I	I	0
$0 \times I$	0×0	I	I
0×0	$0 \times I$	I	0

I	0	I	0
0	I	I	0
0	$0 \times I$	$I \times 0$	I
0	0×0	$I \times I$	0

I	0	I	0
0	I	I	0
0	$0 \times I$	$I \times 0$	I
0	0×0	$I \times I$	0

마스크를 한 칸씩 옮겨 모두 적용 결과

I	I	I
0	I	I
0	I	I

딥러닝 동작원리

▶ 컨볼루션 신경망(Convolutional Neural Network, CNN)

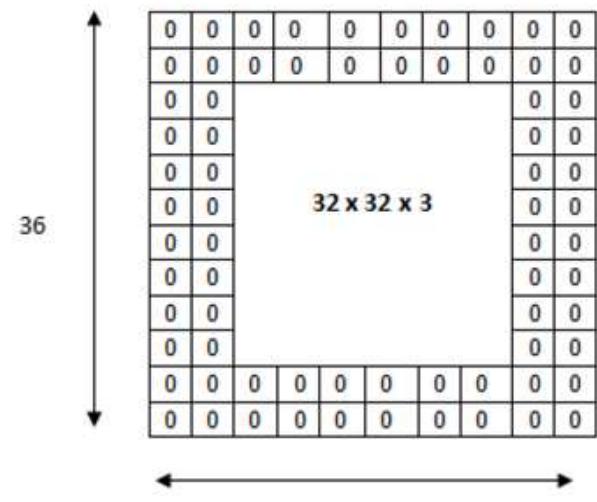
- 5x5 원본 이미지가 3x3의 1 stride 값은 3x3으로 크기가 작아진다
- CNN 네트워크는 하나의 필터 레이어가 아니라 여러 단계에 걸쳐서 계속 필터를 연속적으로 적용하여 특징을 추출하는 최적화를 수행하는데, 필터 적용 후 결과 값이 작아지게 되면 처음에 비해서 특징이 많이 유실 될 수가 있다. 필터를 거쳐감에 따라서 충분히 특징이 추출되기 전에, 결과 값이 작아지면 특징이 유실된다. 이를 방지하기 위한 방법으로 padding 이라는 기법이 있는데, padding은 결과 값이 작아지는 것을 방지하기 위해서 입력값 주위로 0 값 을 넣어서 입력 값의 크기를 인위적으로 키워서, 결과값이 작아지는 것을 방지 하는 기법이다.
- 패딩은 결과 값을 작아지는 것을 막아서 특징이 유실되는 것을 막는 것 뿐 아니라, 오버피팅도 방지
- 패딩은 원본 데이터에 0 값을 넣어서 원래의 특징을 희석 시켜 버리고, 이것을 기반으로 머신러닝 모델이 트레이닝 값에만 정확하게 맞아 들어가는 오버피팅 현상을 방지한다

1	1	1	0	0
0	1	1	1	0
0	0	1 _{x1}	1 _{x0}	1 _{x1}
0	0	1 _{x0}	1 _{x1}	0 _{x0}
0	1	1 _{x1}	0 _{x0}	0 _{x1}

Image

4	3	4
2	4	3
2	3	4

Convolved
Feature

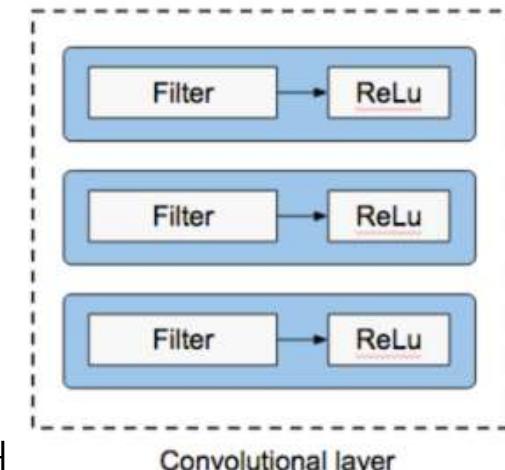


32x32x3 데이터에 폭이 2인 padding을 적용

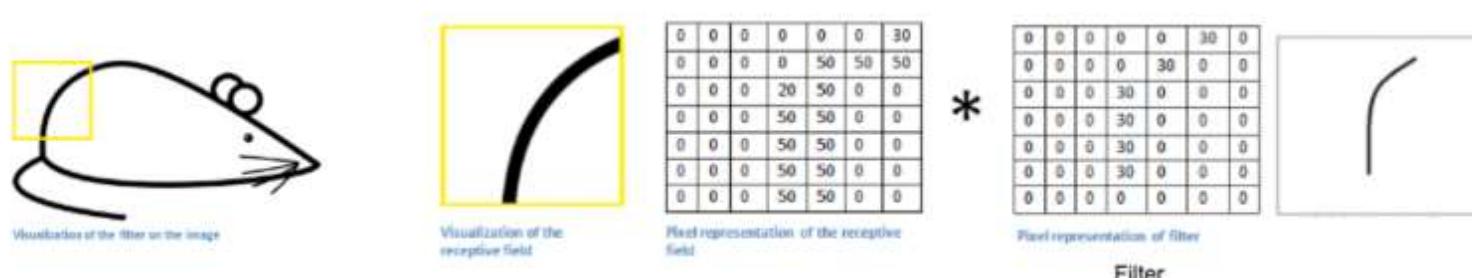
딥러닝 동작원리

➤ 컨볼루션 신경망(Convolutional Neural Network, CNN)

- 컨볼루셔널 계층을 통해서 입력 받는 이미지에 대한 특징을 추출
- 특징을 추출하는 기능을 하는 필터(Filter)와 필터의 값을 비선형 값으로 바꾸어 주는 Activation함수로 구성



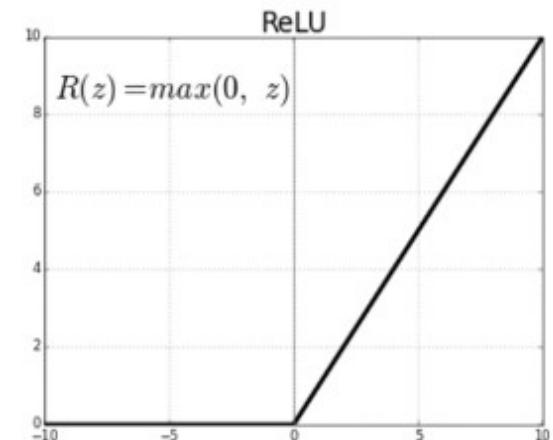
- 필터는 입력받은 데이터에서 그 특성을 가지고 있으면 결과 값이 큰값이 나오고, 특성을 가지고 있지 않으면 결과 값이 0에 가까운 값이 나오게 돼서 데이터가 그 특성을 가지고 있는지 없는지 여부를 알 수 있게 해준다.
- 입력값에는 여러가지 특징이 있기 때문에 하나의 필터가 아닌 여러개의 다중 필터를 같이 적용하게 된다
- Filter : 특징이 데이터에 있는지 없는지를 검출 (잘라낸 이미지와 필터를 곱하면 매우 큰 값이 나온다.)



딥러닝 동작원리

➤ 컨볼루션 신경망(Convolutional Neural Network, CNN)

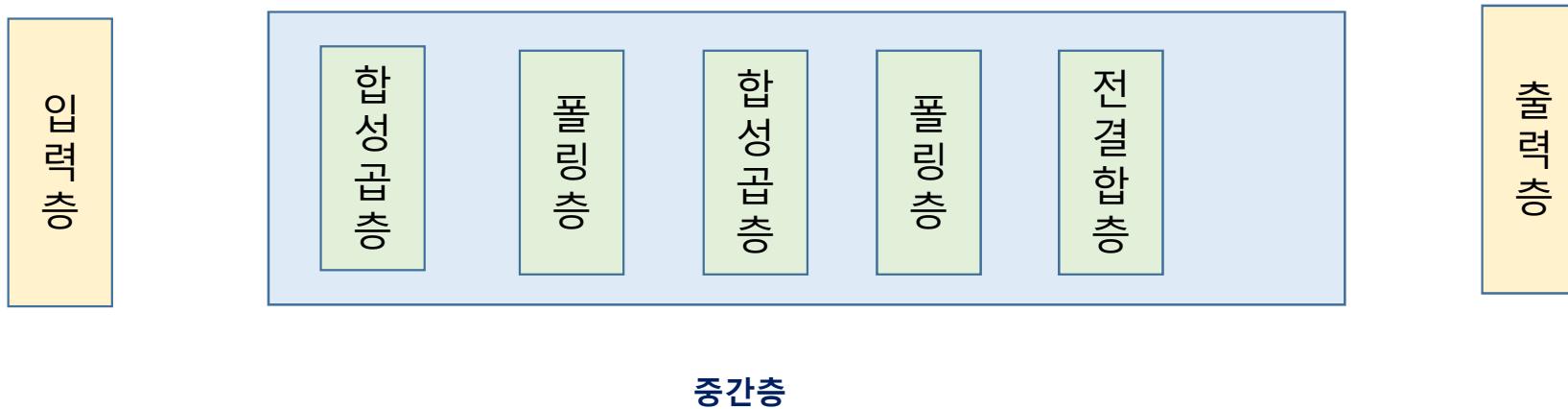
- 필터들을 통해서 Feature map이 추출되었으면, 이 Feature map에 Activation function을 적용
- Activation function 적용은 필터를 통해서 추출한 값이 양적인 값으로 나오기 때문에, 그 특징이 “있다 없다”的 비선형 값으로 바꿔 주는 과정이다
- 뉴럴 네트워크나 CNN (CNN도 뉴럴 네트워크이다.) Activation 함수로 ReLU 함수를 주요 사용한다.



딥러닝 동작원리

➤ 컨볼루션 신경망(Convolutional Neural Network, CNN)

- 합성곱층은 이미지의 특징을 추출할 때 사용
- 입력 x 의 일부분을 잘라내고, 가중치 필터 W 를 적용해 특징 맵 c 를 만들어 낼 때 사용
- 입력 x 의 일부분을 조금씩 자르면서 평활화와 윤곽선 검출 처리를 하며, 특징 맵 c 를 추출
- 합성곱층의 역할은 주변의 값과 필터를 사용해 중앙에 있는 값을 변화시키는 것
- 평활화(Equalization) – 명암의 분포가 균일하지 못한 이미지에 적용해 분포를 균일하게 만들어주는 것

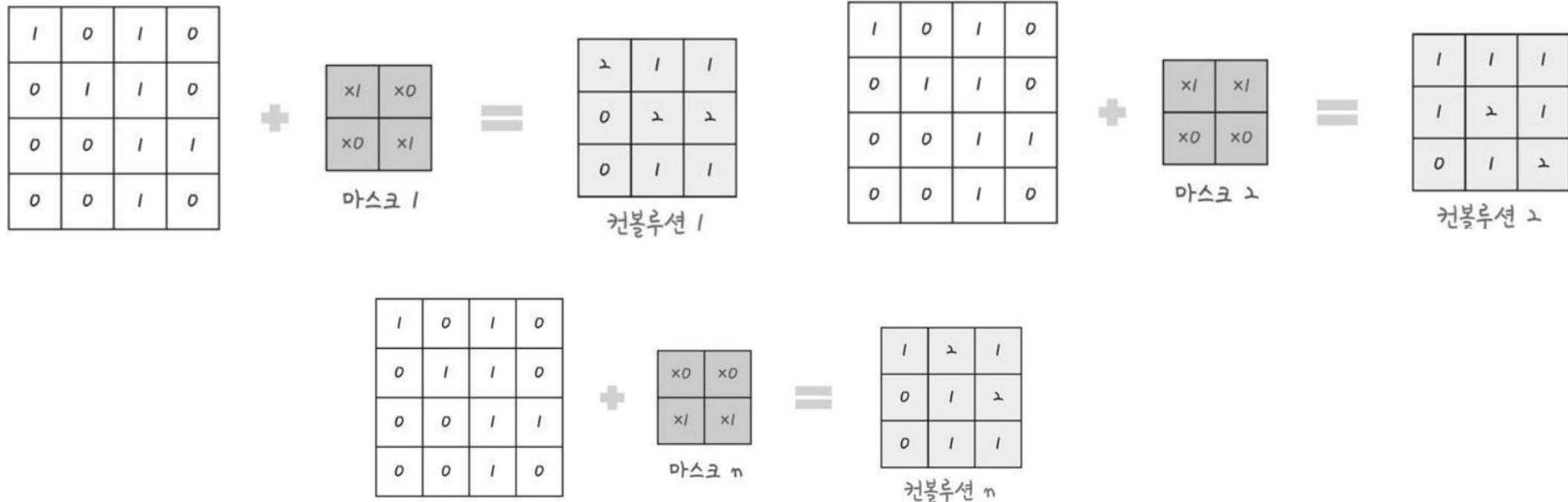


- **풀링층**은 합성곱층으로 얻은 특징맵 c 를 축소하는 층 – max pooling, average pooling
- **전결합층**은 각 층의 유닛을 결합

딥러닝 동작원리

➤ 컨볼루션 신경망(Convolutional Neural Network, CNN)

- 컨볼루션(합성곱) : 새롭게 만들어진 층을
- 마스크를 여러 개 만들 경우 여러 개의 컨볼루션이 만들어집니다.



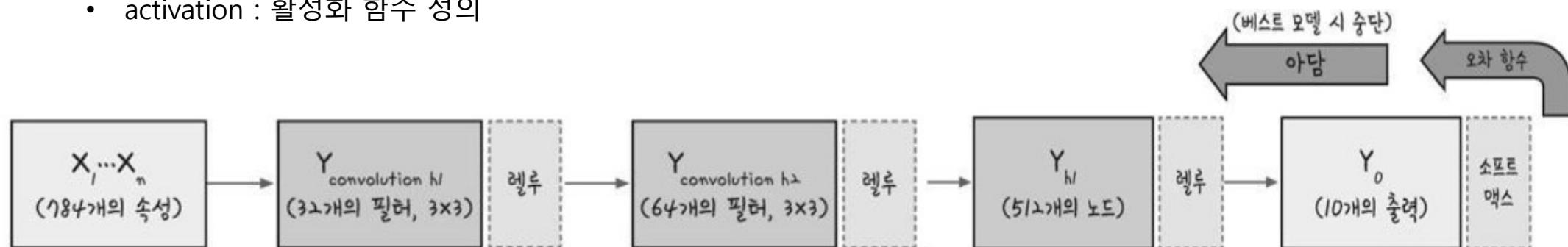
딥러닝 동작원리

➤ 컨볼루션 신경망(Convolutional Neural Network, CNN)

- 케라스에서 컨볼루션 층을 추가하는 함수는 **Conv2D()**입니다.

```
model.add(Conv2D(32, kernel_size=(3, 3), input_shape=(28, 28, 1), activation='relu'))
```

- 첫 번째 인자 : 마스크 적용 개수
- kernel_size : 마스크(커널)의 크기. kernel_size=(행, 열) 형식
- input_shape : 맨 처음 층의 입력 뉴런 수
- activation : 활성화 함수 정의



딥러닝 동작원리

➤ 컨볼루션 신경망(Convolutional Neural Network, CNN) - 맥스 풀링(max pooling)

- 풀링(pooling), 서브 샘플링(sub sampling) : 컨볼루션 층을 통해 이미지 특징을 도출 결과가 여전히 크고 복잡하면 다시 한번 축소하는 과정
- 맥스 풀링은 정해진 구역 안에서 가장 큰 값만 다음 층으로 넘기고 나머지는 버립니다.
- MaxPooling2D()

1	0	1	0
0	4	2	0
0	1	6	1
0	0	1	0

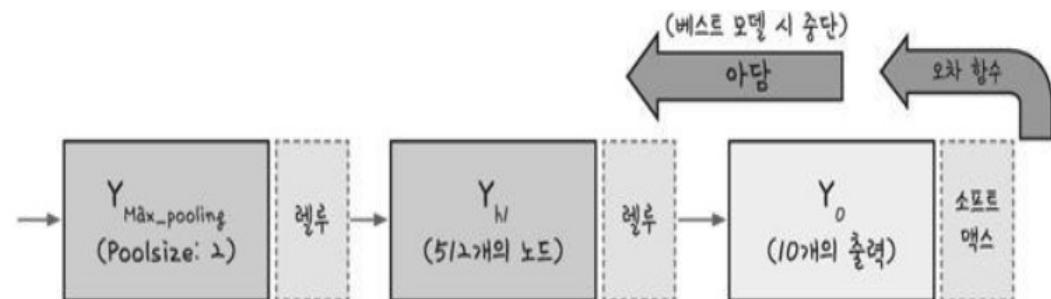
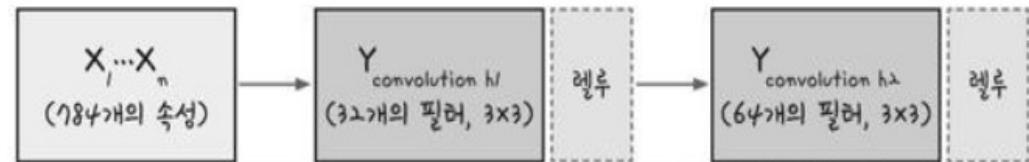
이미지

1	0	1	0
0	4	2	0
0	1	6	1
0	0	1	0

구역을 나눕니다.

4	2
1	6

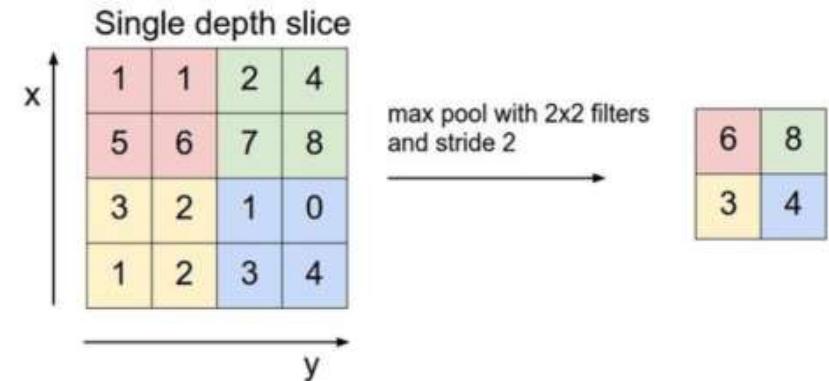
각 구역에서 가장 큰 값을 추출합니다



딥러닝 동작원리

➤ 컨볼루션 신경망(Convolutional Neural Network, CNN)

- 풀링 (Sub sampling or Pooling) : 추출된 Activation map을 인위로 줄이는 작업
컨볼루셔널 계층을 통해서 어느정도 특징이 추출 되었으면, 이 모든 특징을 가지고 판단을 할 필요가 없다.
- max pooling, average pooling, L2-norm pooling 등 (max pooling이라는 기법이 많이 사용된다.)
- **Max pooling (맥스 풀링)** : Activation map을 MxN의 크기로 잘라낸 후, 그 안에서 가장 큰 값을 뽑아내는 방법
-

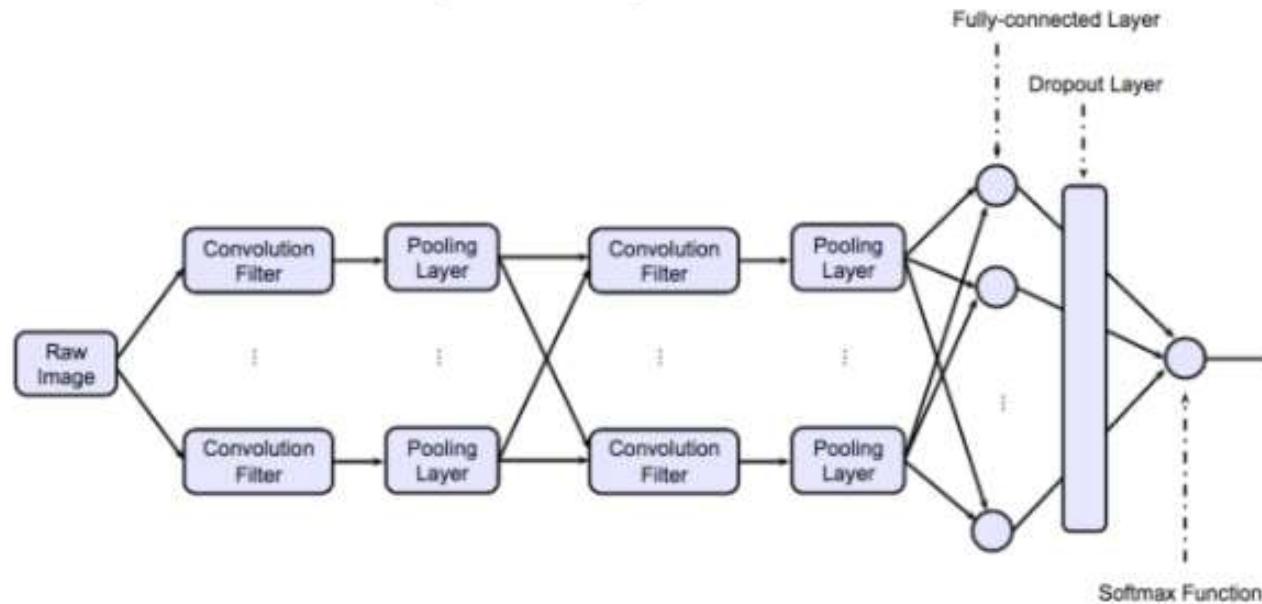


- 맥스 풀링은 특징의 값이 큰 값이 다른 특징들을 대표한다는 개념을 기반으로 하고 있다.
- sampling 을 통해서 얻을 수 있는 장점 :
전체 데이터의 사이즈가 줄어들기 때문에 연산에 들어가는 컴퓨팅 리소스가 적어지고
데이터의 크기를 줄이면서 소실이 발생하기 때문에, 오버피팅을 방지할 수 있다.

딥러닝 동작원리

➤ 컨볼루션 신경망(Convolutional Neural Network, CNN)

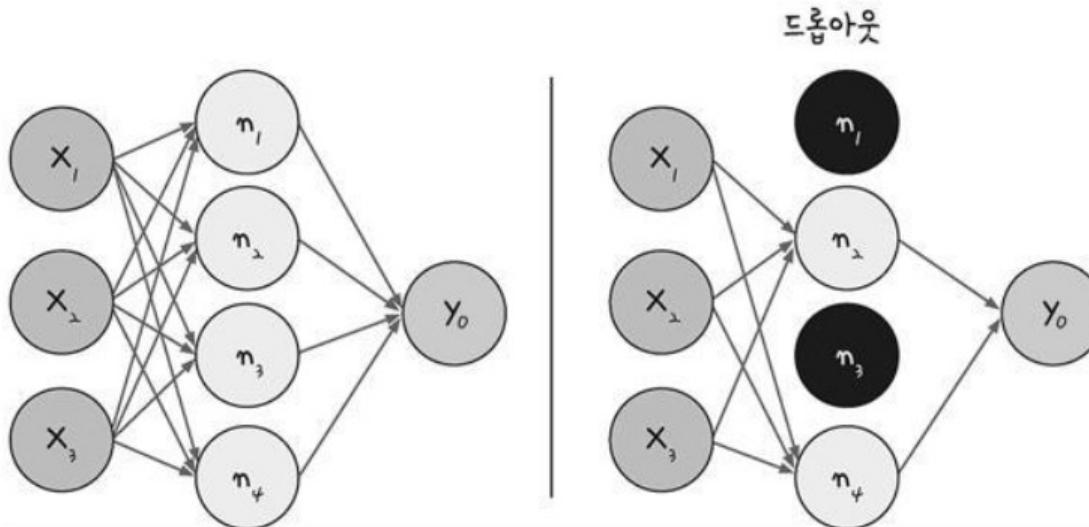
- **컨볼루셔널 레이어** : 컨볼루셔널 필터와 액티베이션 함수 (ReLU) 그리고 풀링 레이어를 반복적으로 조합하여 특징을 추출한다.
- **Fully connected Layer** : 컨볼루셔널 계층에서 특징이 추출이 되었으면 이 추출된 특징 값을 기존의 뉴럴 네트워크 (인공 신경 지능망)에 넣어서 분류를 한다.



딥러닝 동작원리

➤ 컨볼루션 신경망(Convolutional Neural Network, CNN) – 과적합 피하기

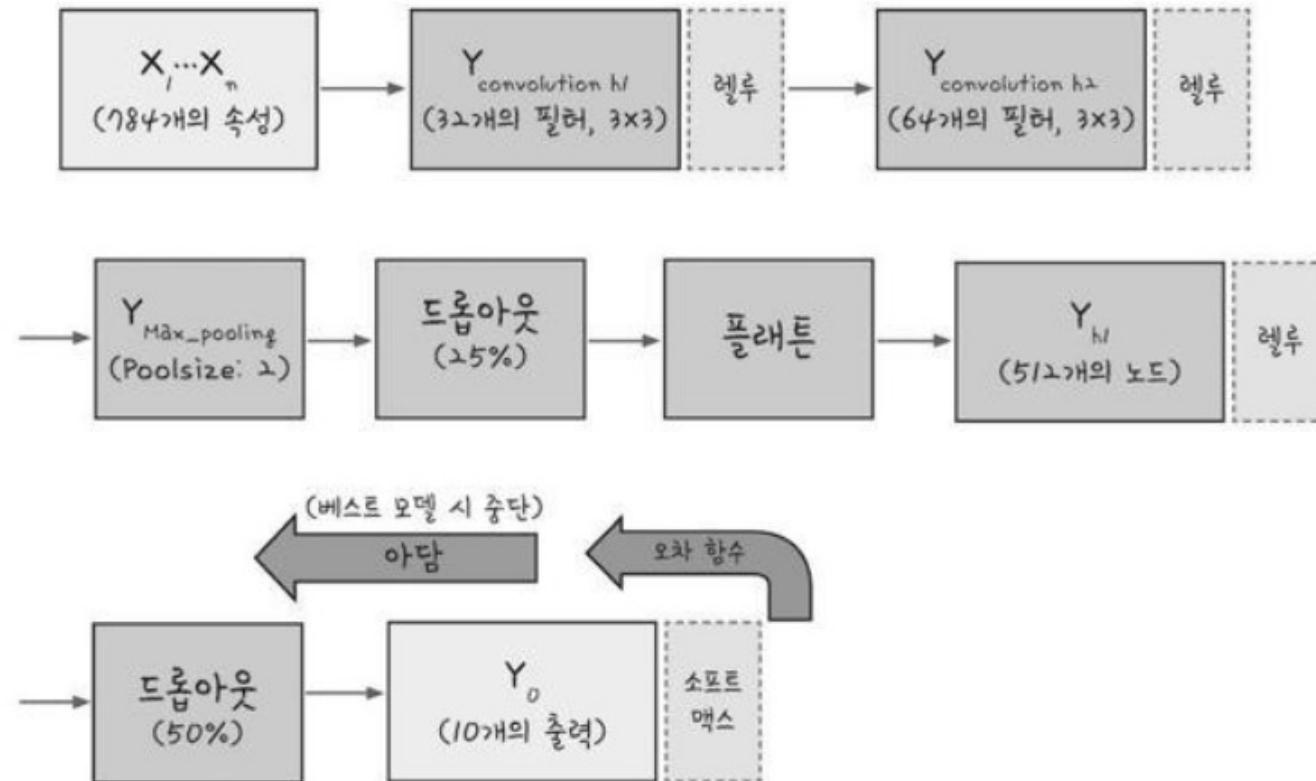
- 딥러닝 학습을 실행할 때 가장 중요한 것은 과적합을 얼마나 효과적으로 피해가는지에 달려 있습니다.
- **드롭아웃(drop out)** 기법 - 은닉층에 배치된 노드 중 일부를 임의로 꺼주는 것입니다.
- 랜덤하게 노드를 끔으로써 학습 데이터에 지나치게 치우쳐서 학습되는 과적합을 방지할 수 있습니다.



```
model.add(Dropout(0.25)) #25%의 노드를 끔
```

딥러닝 동작원리

➤ 컨볼루션 신경망(Convolutional Neural Network, CNN) – 과적합 피하기



드롭아웃과 풀래튼 추가하기

딥러닝 동작원리

➤ 컨볼루션 신경망(Convolutional Neural Network, CNN)

```
from keras.datasets import mnist
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
from keras.callbacks import ModelCheckpoint, EarlyStopping
import matplotlib.pyplot as plt
import numpy
import os
import tensorflow as tf

# seed 값 설정
seed = 0
numpy.random.seed(seed)
tf.set_random_seed(seed)

# 데이터 불러오기
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1).astype('float32') / 255
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float32') / 255
Y_train = np_utils.to_categorical(Y_train)
Y_test = np_utils.to_categorical(Y_test)
```

딥러닝 동작원리

- 각 층에서 뉴런(neuron)의 개수에는 제약이 없다.
- summary() 함수로 자신이 생성한 모델의 레이어, 출력 모양, 파라미터 개수 등을 체크할 수 있다.
- model.summary()
- 레이어의 설정 및 레이어의 층 구조에 따라서 모델의 성능이 달라지는 데, 변경해볼 수 있는 요소를 하이퍼파라미터라고 하고, 이를 하이퍼파라미터 튜닝이라고 합니다.
- 마지막 출력층은 하이퍼파라미터가 없습니다.
- fit() 함수 로그에는 기본적으로 손실값과 정확도가 표시됩니다.
- 이진분류 모델에서는 정확도값 하나만 보더라도 학습이 제대로 되고 있는지 알 수 있지만, 다중클래스분류 문제에서는 클래스별로 학습이 제대로 되고 있는지 확인하기 위해서는 정확도값 하나로는 부족함이 많습니다.
- **메트릭(metric)** : 평가 기준. compile() 함수의 metrics 인자로 여러 개의 평가 기준을 지정할 수 있습니다.
- 평가 기준에는 모델의 학습에는 영향을 미치지 않지만, 학습 과정 중에 제대로 학습되고 있는지 살펴볼 수 있습니다.
- 케라스에서는 다중클래스분류 문제에서 평가기준을 'accuracy'로 지정했을 경우 내부적으로 categorical_accuracy() 함수를 이용하여 정확도가 계산됩니다.
- evaluate() 함수는 손실값 및 메트릭 값을 반환
- 컨볼루션 레이어인 Conv2D에서는 필터의 수 "32"와 필터의 사이즈 "(3,3)"가 성능에 영향을 미칩니다
- Dense 레이어의 출력 뉴런 수인 "20"과 Dropout 레이어의 드랍비율인 "0.2"가 하이퍼파라미터가 됩니다
- 은닉층을 여러 번 쌓을 수도 있기에 몇 번 반복하느냐도 모델 구성에도 중요합니다.
-

머신러닝

➤ 텍스트 분석

- 텍스트 마이닝(Text Mining)
- 비정형 데이터에서 의미 있는 정보를 추출하는 것에 좀 더 중점을 두고 기술이 발전
- **텍스트 분류(Text Classification)** – Text Categorization, 문서가 특정 분류 또는 카테고리에 속하는 것을 예측하는 기법
예] 특정 신문 기사 내용이 연애/정치/사회 문화 중 어떤 카테고리에 속하는지 자동으로 분류하거나 스팸 메일 검출 같은 프로그램
지도학습을 적용
- **감정분석(Sentiment Analysis)** – 텍스트에서 나타나는 감정/판단/믿음/의견/기분등의 주관적인 요소를 분석하는 기법
소셜 미디어 감정 분석, 영화나 제품에 대한 긍정 또는 리뷰, 여론조사 의견분석 등의 다양한 영역에서 활용됩니다.
Text Analytics에서 가장 활발하게 사용되고 있는 분야
지도학습 방법, 비지도학습을 이용해 적용
- **텍스트 요약(Summarization)** – 텍스트내에서 중요한 주제나 중심 사상을 추출하는 기법
토픽 모델링(Topic Modeling)
- **텍스트 군집화(Clustering)와 유사도 측정** – 비슷한 유형의 문서에 대해 군집화를 수행하는 기법
텍스트 분류를 비지도학습으로 수행하는 방법의 일환으로 사용될 수 있습니다.
유사도 측정은 문서들간의 유사도를 측정해 비슷한 문서끼리 모을 수 있는 방법

머신러닝

➤ NLP

- National Language Processing
- 머신이 인간의 언어를 이해하고 해석하는데 더 중점을 두고 기술이 발전해
- 언어를 해석하기 위한 기계 번역
- 자동으로 질문을 해석하고 다블 해주는 질의응답 시스템 등의 영역
- 텍스트 분석을 향상시키는 기반 기술

머신러닝

➤ 텍스트 분석

- 텍스트를 머신러닝에 적용하기 위해서는 비정형 텍스트 데이터를 어떻게 피처 형태로 추출하고 추출된 피처에 의미 있는 값을 부여하는가 하는 것이 매우 중요
- 피처 벡터화(Feature Vectorization) , 피처 추출(Feature Extraction) – 텍스트를 word 기반의 다수의 피처로 추출하고 이 피처에 단어 빈도수와 같은 숫자 값을 부여
BOW(Bag of Words), Word2Vec
- **텍스트 분석 수행 프로세스**
 1. 텍스트 전처리 : 텍스트를 피처로 만들기 전에 미리 클렌징, 대/소문자 변경, 특수문자 삭제 등의 클렌징 작업, 단어(Word) 등의 토큰화 작업, 의미 없는 단어(Stop word) 제거 작업, 어근 추출(Stemming/Lemmatization)등의 텍스트 정규화 작업을 수행하는 것
 2. 피처 벡터화/추출 : 사전 준비 작업으로 가공된 텍스트에서 피처를 추출하고 여기에 벡터 값을 할당
BOW(Count기반 벡터화, TF-IDF 기반 벡터화)와 Word2Vect
 3. ML 모델 수립 및 학습/예측/평가 : 피처 벡터화된 데이터 세트에 ML 모델을 적용해 학습/예측 및 평가를 수행

머신러닝

➤ 파이썬 기반의 NLP, 텍스트 분석 패키지

- **NLTK(Natural Language Toolkit for Python)** : 파이썬의 가장 대표적인 NLP 패키지, 방대한 데이터 세트와 서브 모듈을 가지고 있으며 NLP의 거의 모든 영역을 커버하고 있습니다. 많은 NLP 패키지가 NLTK의 영향을 받아 작성되고 있습니다. 수행 속도 측면에서 부족한 부분이 있어서 실제 대량의 데이터 기반에서는 제대로 활용되지 못하고 있습니다.
- **Gensim** : 토픽 모델링 분야에서 가장 두각을 나타내는 패키지. 오래전부터 토픽 모델링을 쉽게 구현할 수 있는 기능을 제공해 왔으며, Word2Vec 구현 등의 다양한 신기능도 제공, SpaCy와 함께 가장 많이 사용되는 NLP 패키지
- **SpaCy** : 뛰어난 수행 성능으로 최근 가장 주목을 받는 NLP 패키지

머신러닝

➤ 텍스트 정규화

- 텍스트를 머신러닝 알고리즘이나 NLP 애플리케이션에 입력 데이터로 사용하기 위해 클렌징, 정제, 토큰화, 어근화 등의 다양한 텍스트 데이터의 사전 작업을 수행하는 것
- **클렌징(Cleansing)** : 텍스트에서 분석에 방해가 되는 불필요한 문자, 기호 등을 사전에 제거하는 작업
예] HTML, XML 태그나 특정 기호 등을 사전에 제거
- **토큰화(Tokenization)** : 문자 토큰화, 단어 토큰화
문장 토큰화(sentence tokenization) : 문장의 마침표(.), 개행문자(₩n) 등 문장의 마지막을 뜻하는 기호에 따라 분리 각 문장이 가지는 시맨틱적인 의미가 중요한 요소로 사용될 때 사용
sent_tokenize를 이용해 토큰화 수행
`nltk.download('punkt')`는 마침표, 개행 문자등의 데이터 세트를 다운로드 받습니다.
단어 토큰화(Word Tokenization) : 문장을 단어로 토큰화, 단어의 순서가 중요하지 않은 경우 사용
`word_tokenize()`
n-gram : 연속된 n개의 단어를 하나의 토큰화 단위로 분리, n개 단어 크기 윈도우를 만들어 문장의 처음부터 오른쪽으로 움직이면서 토큰화를 수행
- **필터링/Stop Word 제거/철자 수정**
문장을 구성하는 필수 문법 요소지만 문맥적으로 큰 의미가 없는 단어 제거
NLTK에서는 다양한 언어의 스톱 워드를 제공
- **Stemming** : 원형 단어로 변환 시 일반적인 방법을 적용하거나 더 단순화된 방법을 적용해 원래 단어에서 일부 철자가 훼손된 어근 단어를 추출하는 경향이 있습니다.
Porter, Lancaster, Snowball Stemmer
- **Lemmatization** : 품사와 같은 문법적인 요소와 더 의미적인 부분을 감안해 정확한 철자로 된 어근 단어를 찾아줍니다.
LancasterStemmer
Lemmatization이 Stemming보다 변환에 더 오랜 시간을 필요로 합니다.

머신러닝

➤ 텍스트 정규화

- Bag of Words 모델은 문서가 가지는 모든 단어(Words)를 문맥이나 순서를 무시하고 일괄적으로 단어에 대해 빈도 값을 부여해 피처 값을 추출하는 모델입니다.
- 문서 내 모든 단어를 한꺼번에 봉투(Bag)안에 넣은 뒤에 흔들어서 섞는다는 의미로 Bag of Words(BOW)모델이라고 합니다.
- BOW 모델의 장점은 쉽고 빠른 구축에 있습니다.
- 단순히 단어의 발생 횟수에 기반하고 있지만, 예상보다 문서의 특징을 잘 나타낼 수 있는 모델이어서 전통적으로 여러 분야에서 활용도가 높습니다.
- BOW는 단어의 순서를 고려하지 않기 때문에 문장 내에서 단어의 문맥적인 의미가 무시됩니다.
- BOW로 피처 벡터화를 수행하면 희소 행렬 형태의 데이터 세트가 만들어지기 쉽습니다.
- 많은 문서에서 단어를 추출하면 매우 많은 단어가 컬럼으로 만들어집니다.
- 희소 행렬은 일반적으로 ML알고리즘의 수행 시간과 예측 성능을 떨어뜨리기 때문에 희소 행렬을 위한 특별한 기법이 마련돼 있습니다

머신러닝

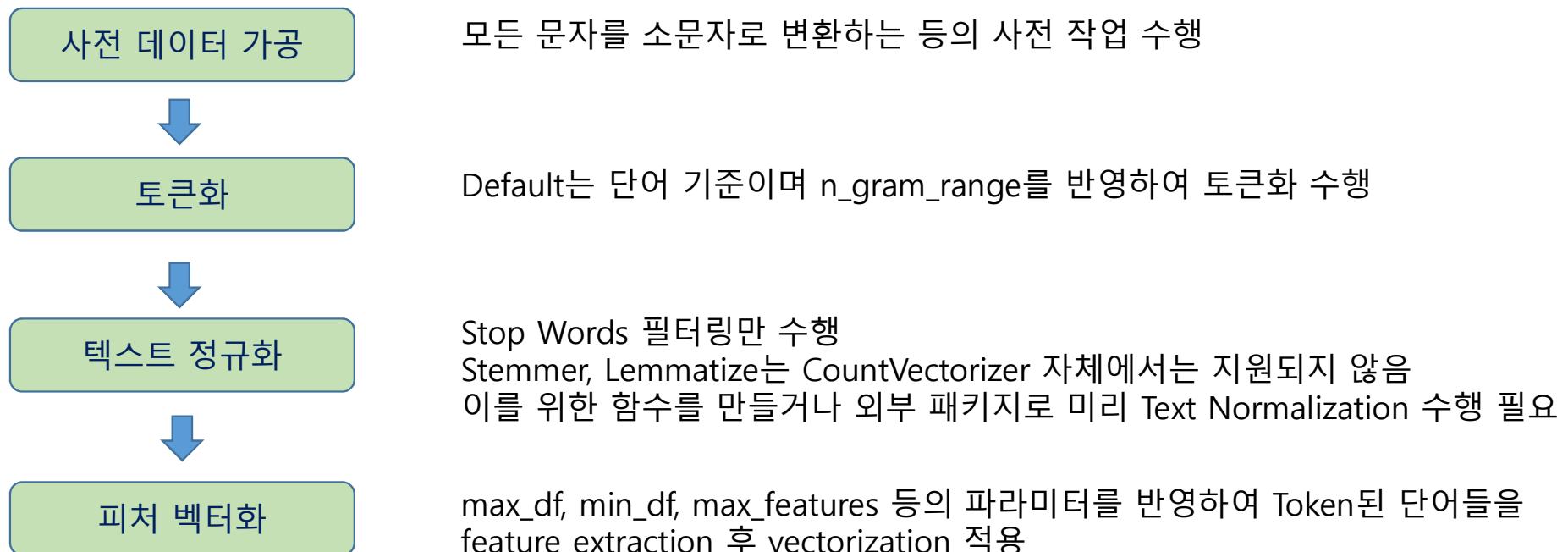
➤ 텍스트 정규화

- 텍스트를 특정 의미를 가지는 숫자형 값인 벡터 값으로 변환하는 피처 벡터화 방법 :
- 각 문서의 텍스트를 단어로 추출해 피처로 할당
- 각 단어의 발생 빈도와 같은 값을 피처에 값으로 부여
- 각 문서를 단어 피처의 발생 빈도 값으로 구성된 벡터로 만듬
- 카운트 기반의 벡터화 : 카운트 값이 높을수록 중요한 단어로 인식
- TF-IDF(Term Frequency - Inverse Document Frequency)기반의 벡터화 : 개별 문서에서 자주 나타나는 단어에 높은 가중치를 주되, 모든 문서에서 전반적으로 자주 나타나는 단어에 대해서는 페널티를 주는 방식으로 값을 부여
- 문서마다 텍스트가 길고 문서의 개수가 많은 경우 카운트 방식보다 TF-IDF 방식을 사용하는 것이 더 좋은 예측 성능을 보장할 수 있습니다.
- 사이킷런의 **CountVectorizer** - Feature 벡터화와 소문자 일괄 변환, 토큰화, stop word 필터링등 전처리를 함께 수행
max-df - 전체 문서에 걸쳐서 너무 높은 빈도수를 가지는 단어 피처를 제외하기 위한 파라미터
min-df - 전체 문서에 걸쳐서 너무 낮은 빈도수를 가지는 단어 피처를 제외하기 위한 파라미터
max-features - 추출하는 피처의 개수를 제한하며 정수로 값을 지정
stop_words - 추출에서 제외될 단어
n_gram_range - 단어 순서를 보정하기 위한 n_gram 범위 설정
analyzer - 피처 추출을 수행한 단위를 지정
token_pattern - 토큰화를 수행하는 정규 표현식 패턴을 지정
tokenizer - 토큰화를 별도의 커스텀 함수로 이용시 적용
- 사이킷런의 **TfidfVectorizer**

머신러닝

➤ 텍스트 정규화

- 사이킷런의 **CountVectorizer** 를 이용한 피처 벡터화



머신러닝

➤ 텍스트 정규화

- COO(Coordinate)형식은 0이 아닌 데이터만 별도의 데이터 배열(Array)에 저장하고, 그 데이터가 가르키는 행과 열의 위치를 별도의 배열로 저장하는 방식
- 사이파이의 sparse패키지는 희소 행렬 변환을 위한 다양한 모듈을 제공합니다.
- CSR(Compressed Sparse Row) 형식은 COO형식이 행과 열의 위치를 나타내기 위해서 반복적인 위치 데이터를 사용해야 하는 문제점을 해결한 방식
- 행 위치 배열 내에 있는 고유한 값의 시작 위치만 다시 별도의 위치 배열로 가지는 변환 방식
- 사이파이의 csr_matrix 클래스를 이용하여 변환

머신러닝

➤ 텍스트 분류 실습 – 20 뉴스 그룹 분류

- 사이킷런은 fetch_20newsgroup() API를 이용해 뉴스그룹의 분류를 수행해 볼 수 있는 예제 데이터를 제공

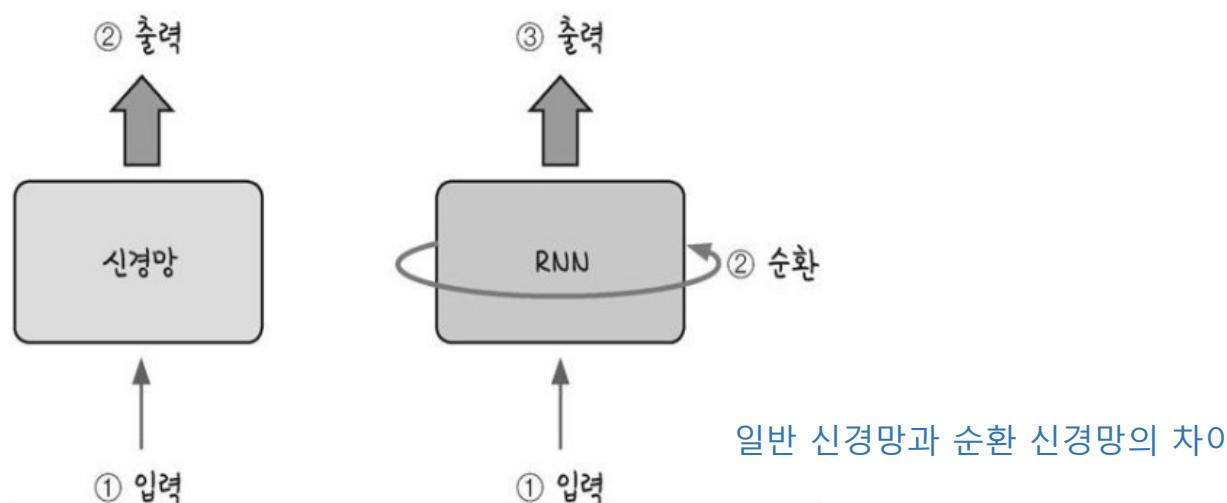
머신러닝

➤ 감성 분석

- 문서의 주관적인 감성/의견/감정/기분 등을 파악하기 위한 방법
- 쇼셜 미디어, 여론조사, 온라인 리뷰, 피드백 등 다양한 분야에서 활용
- 문서 내 텍스트가 나타내는 여러 가지 주관적인 단어와 문맥을 기반으로 감성 수치를 계산하는 방법을 이용
- 비지도 학습은 Lexicon이라는 일종의 감정 어휘 사전을 이용합니다. Lexicon은 감정 분석을 위한 용어와 문맥에 대한 다양한 정보를 가지고 있으며, 이를 이용해 문서의 긍정적, 부정적 감성 여부를 판단합니다.

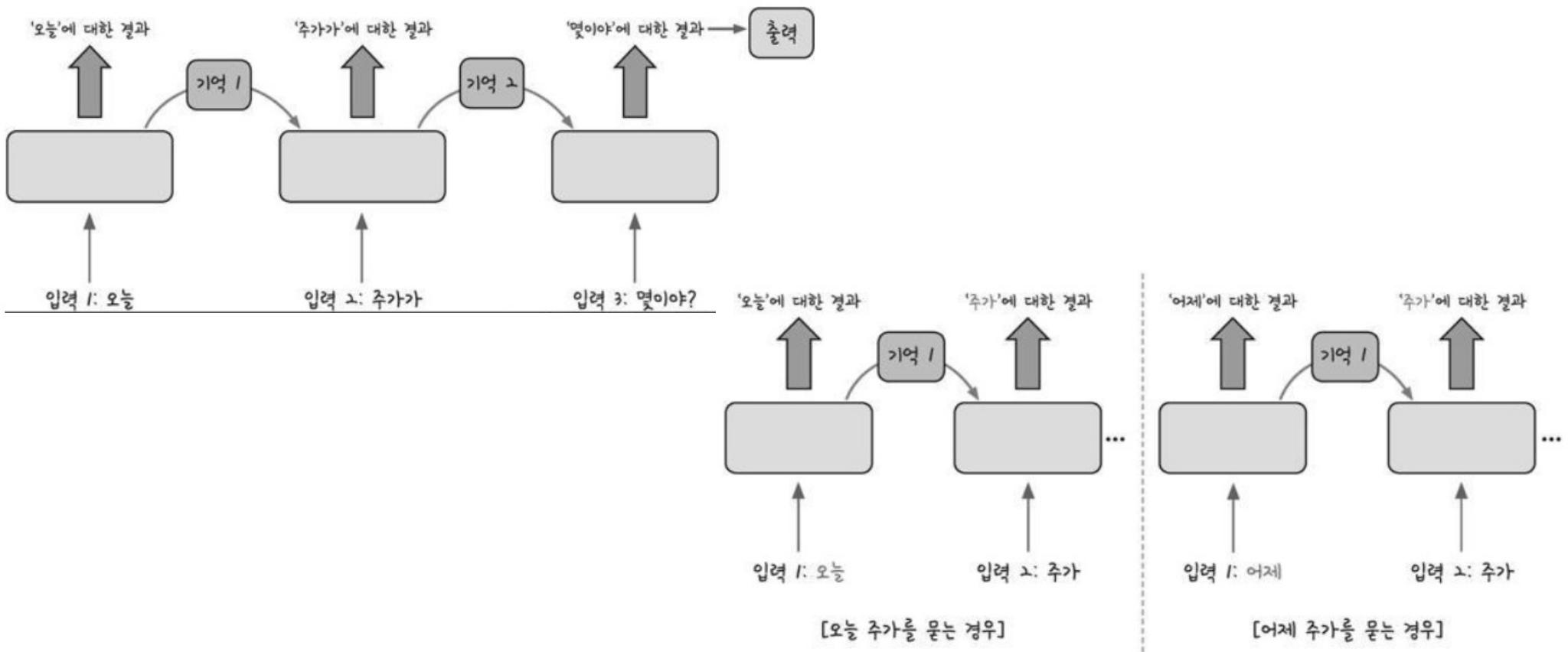
순환 신경망(Recurrent Neural Network, RNN)

- 인공지능이 문장을 듣고 이해한다는 것은 과거에 입력된 데이터와 나중에 입력된 데이터 사이의 관계를 고려해서 학습하는 것입니다.
- 입력된 문장의 의미를 파악하는 것은 곧 모든 단어를 종합하여 하나의 카테고리로 분류하는 작업입니다.
- 대화형 인공지능 - 문장을 듣고 무엇을 의미하는지를 알아야 서비스를 제공
- 문장 학습 - 여러 개의 단어로 이루어진 문장의 각 단어가 정해진 순서대로 입력되어야 의미를 전달 할 수 있으므로 입력된 데이터 사이의 관계를 고려해야 합니다.
- 순환 신경망(Recurrent Neural Network, RNN)**은 여러 개의 데이터가 순서대로 입력되었을 때 앞서 입력 받은 데이터를 잠시 기억해 놓는 방법입니다.
- 모든 입력 값에 기억된 데이터가 얼마나 중요한지를 판단하여 별도의 가중치를 줘서 다음 층으로 데이터로 넘어갑니다.



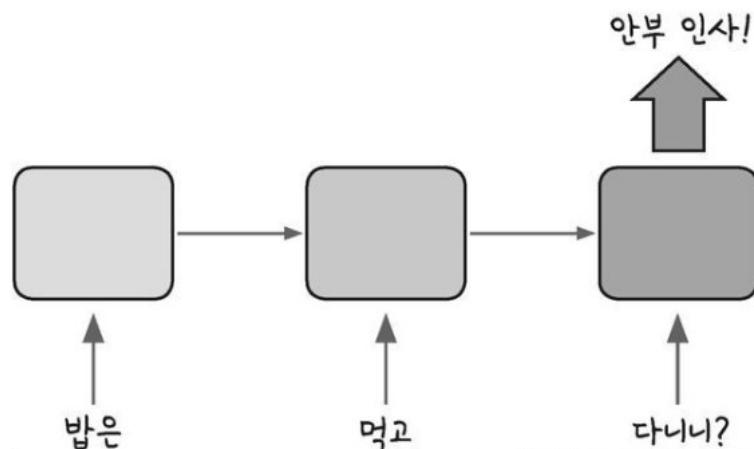
순환 신경망(Recurrent Neural Network, RNN)

- RNN 처리 방식 : 순환 부분에서 단어를 하나 처리할 때마다 단어마다 기억하여 다음 입력 값의 출력을 결정합니다.
- 순환이 되는 가운데 앞서 나온 입력에 대한 결과가 뒤에 나오는 입력 값에 영향을 주는 것을 알 수 있습니다. 이렇게 해야지만, 비슷한 두 문장이 입력되었을 때 그 차이를 구별하여 출력 값에 반영할 수가 있습니다.



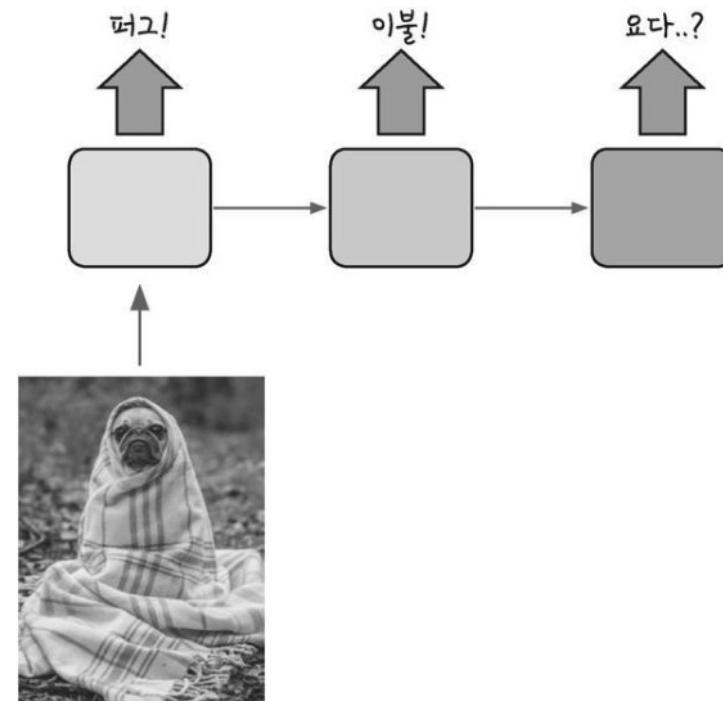
순환 신경망(Recurrent Neural Network, RNN)

- RNN 방식의 장점은 입력 값과 출력 값을 어떻게 설정하느냐에 따라 여러 가지 상황에서 이를 적용할 수 있다는 것입니다.
- RNN의 입력과 출력은 우리가 네트워크에게 시키고 싶은 것이 무엇이냐에 따라 얼마든지 달라질 수 있습니다.



다수 입력 단일 출력(문장을 읽고 뜻을 파악할 때 활용)

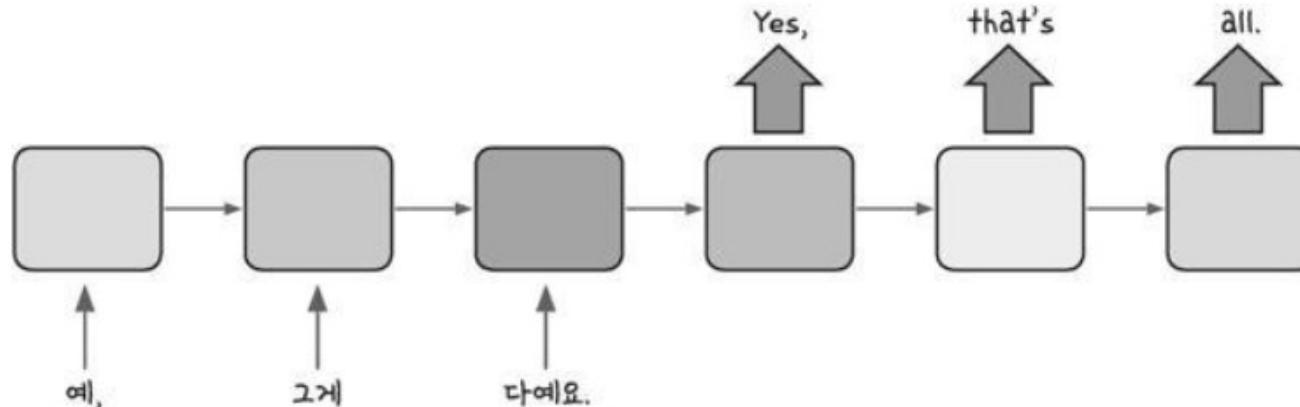
시퀀스 입력 & 고정크기 출력. 예) 문장을 입력해서 긍정 부정 정도를 출력하는 감성 분석기



단일 입력 다수 출력(사진의 캡션을 만들 때 활용)

고정크기 입력 & 시퀀스 출력. 예) 이미지를 입력해서 이미지에 대한 설명을 문장으로 출력하는 이미지 캡션 생성

순환 신경망(Recurrent Neural Network, RNN)



다수 입력 다수 출력(문장을 번역할 때 활용)

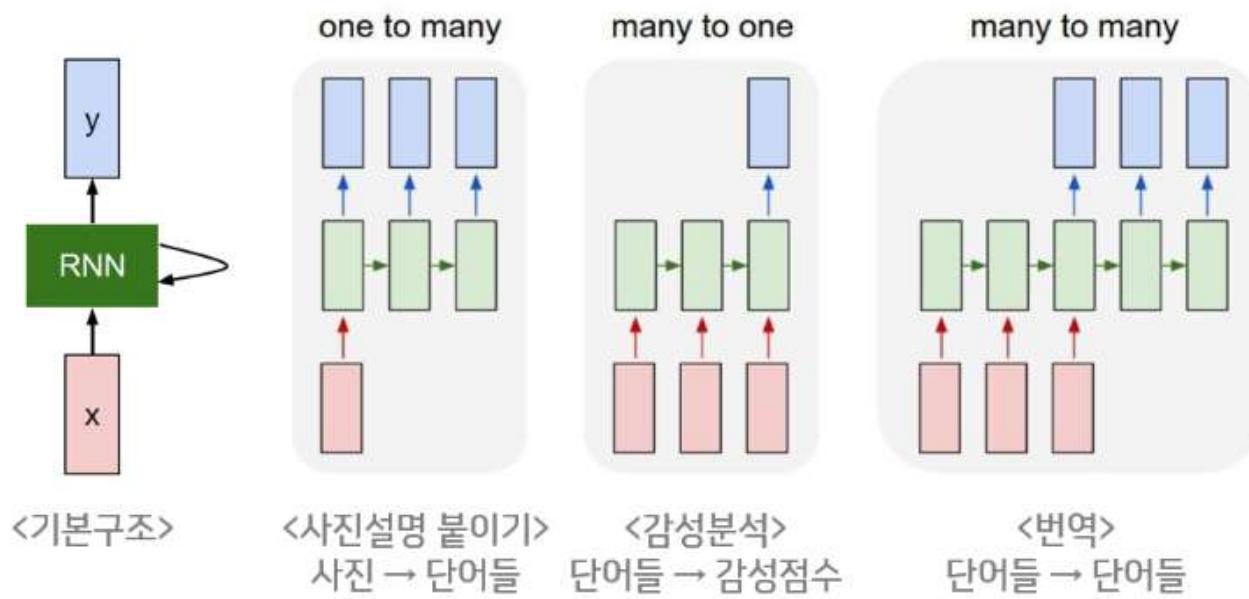
시퀀스 입력 & 시퀀스 출력. 예) 영어를 한국으로 번역하는 자동 번역기

- 동기화된 시퀀스 입력 & 시퀀스 출력. 예) 문장에서 다음에 나올 단어를 예측하는 언어 모델
- 이미지를 처리하는 CNN(Convolutional Neural Network)과 RNN을 결합하여 이미지를 텍스트로 설명해주는 모델을 만드는 것이 가능합니다 (고정크기 입력 & 시퀀스 출력에 해당)
- 구글의 번역기와 네이버의 파파고는 RNN을 응용한 모델로 만들어졌습니다. RNN 기반 모델은 기존 통계 기반 모델의 비해 우수한 성능을 보입니다. (시퀀스 입력 & 시퀀스 출력 구조)

머신러닝

➤ Recurrent Neural Networks(RNN)

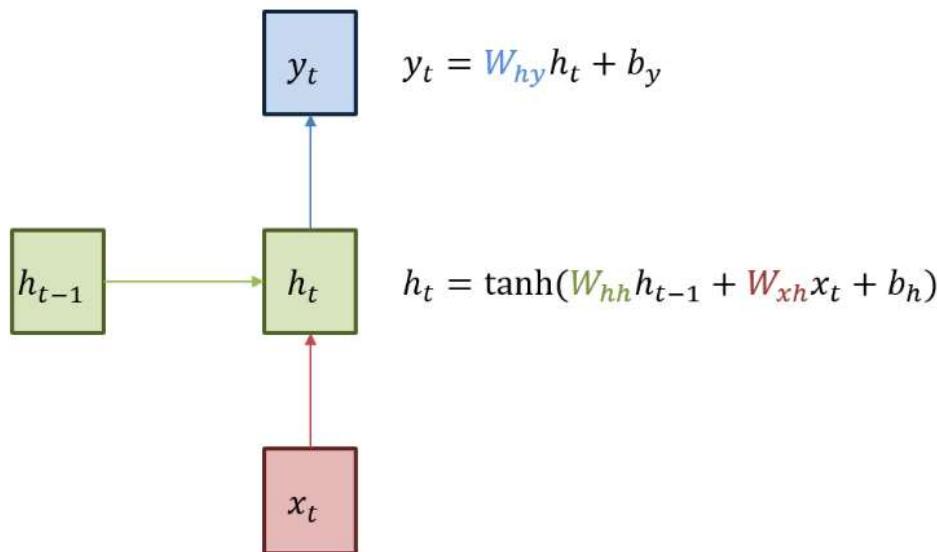
- RNN은 **하든(기억) 노드가 방향을 가진 엣지로 연결돼 순환구조를 이루는(directed cycle) 인공신경망**의 한 종류
- 음성, 문자 등 순차적으로 등장하는 데이터 처리에 적합한 모델
- 시퀀스 길이에 관계없이 입력과 출력을 받아들일 수 있는 네트워크 구조이기 때문에 필요에 따라 다양하고 유연하게 구조를 만들 수 있다
- 네트워크의 기억은 지금까지의 입력 데이터를 요약한 정보라고 볼 수 있습니다. 새로운 입력이 들어 올 때마다 네트워크는 자신의 기억을 조금씩 수정합니다. 결국 입력을 모두 처리하고 난 후 네트워크에게 남겨진 기억은 시퀀스 전체를 요약하는 정보가 됩니다.



머신러닝

➤ Recurrent Neural Networks(RNN)

- 히든 state(RNN 셀), input x, output y
- 현재 상태의 히든 state h_t 는 직전 시점의 히든 state h_{t-1} 를 받아 갱신됩니다.
- 현재 상태의 아웃풋 y_t 는 h_t 를 전달받아 갱신되는 구조
- 히든 state의 활성함수(activation function)은 비선형 함수인 하이퍼볼릭탄젠트(tanh)
- RNN은 관련 정보와 그 정보를 사용하는 지점 사이 거리가 멀 경우 역전파시 그래디언트가 점차 줄어 학습능력이 크게 저하되는 단점이 있습니다. (vanishing gradient problem)



RNN(Recurrent Neural Network) :
입력과 출력을 시퀀스 (Sequence) 단위로 처리하는
모델입니다.

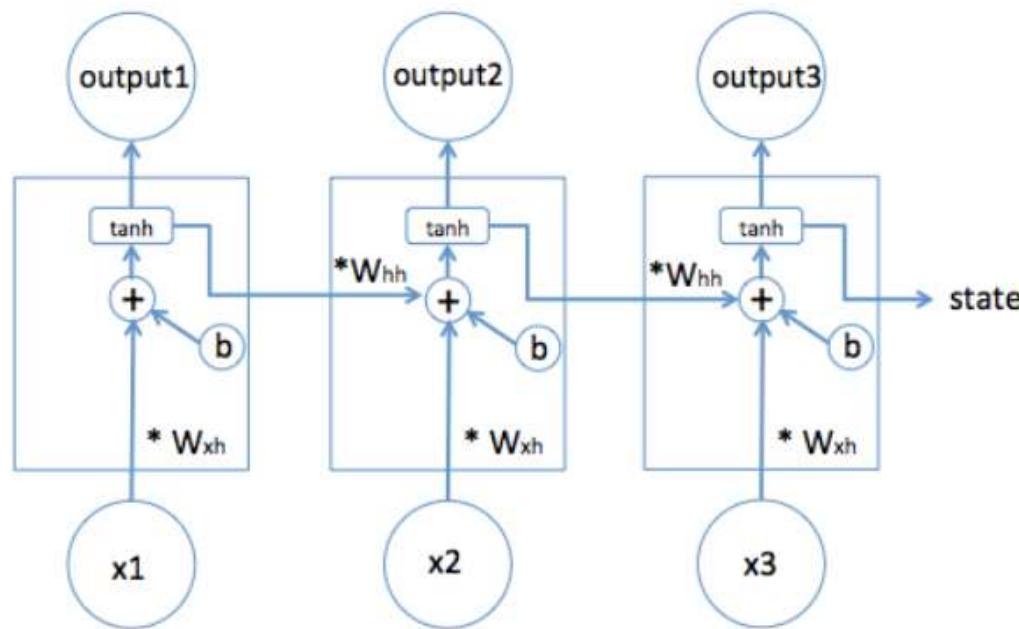
피드 포워드 신경망(Feed Forward Neural
Network) :
은닉층에서 활성화 함수를 지난 값은 오직 출력층
방향으로만 향함

RNN은 은닉층의 노드에서 활성화 함수를 통해 나온 결과값을 출력층 방향으로도 보내면서, 다시 은닉층 노드의 다음 계산의 입력으로 보내는 특징을 갖고 있습니다.

머신러닝

➤ Recurrent Neural Networks(RNN)

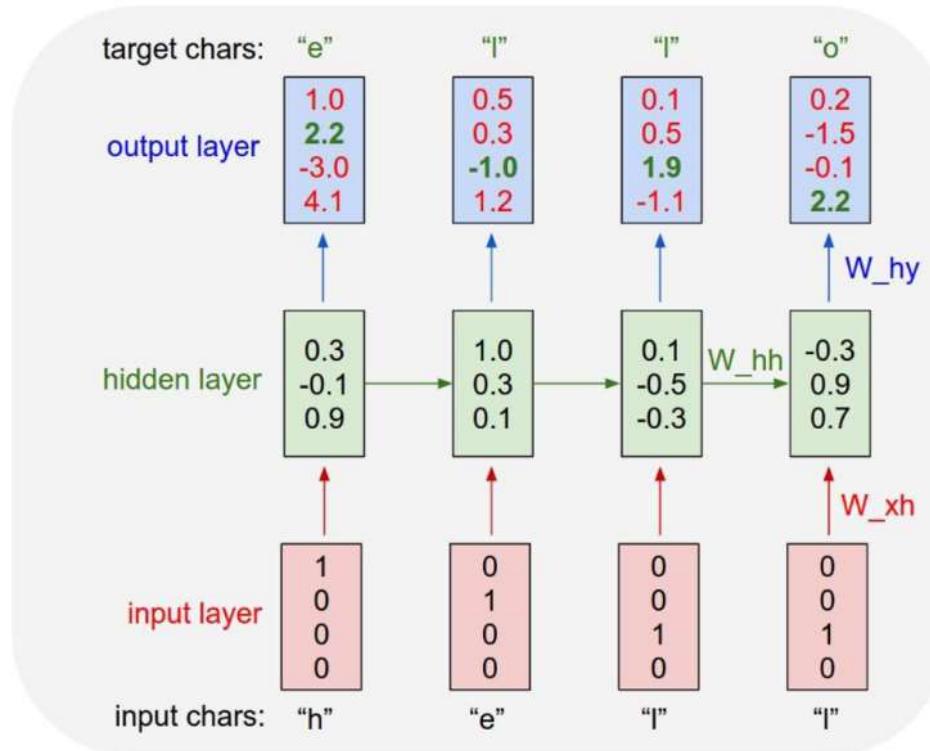
- RNN은 입력값(x), 출력값(output), 상태값(state), 가중치(W), 편향값(b), 그리고 활성화함수(tanh)로 구성



머신러닝

➤ Recurrent Neural Networks(RNN)

- 어떤 글자가 주어졌을 때 바로 다음 글자를 예측하는 character-level-mode
- RNN 모델에 'hell'을 넣으면 'o'를 반환하게 해 결과적으로는 'hello'를 출력
- 학습데이터의 글자는 'h', 'e', 'l', 'o' -> one-hot-vector로 변환 [1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]

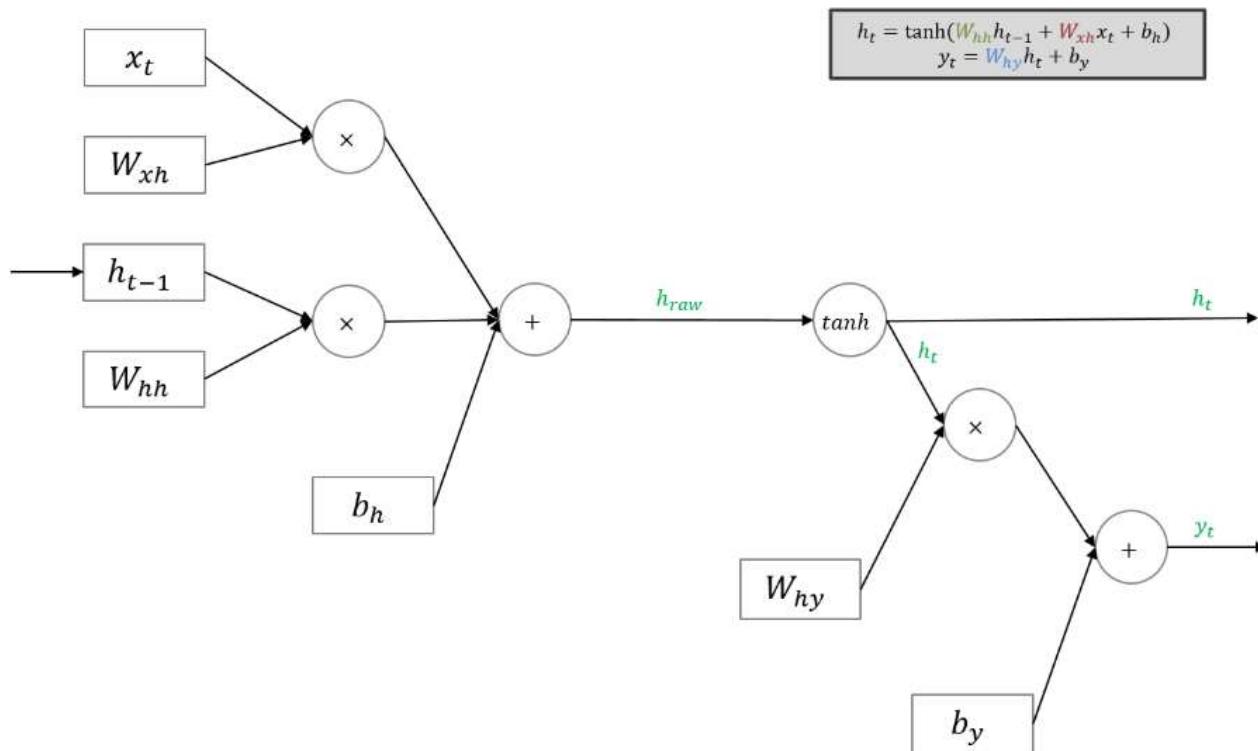


이전 hidden state가 없고
forward propagation 으로 수행

머신러닝

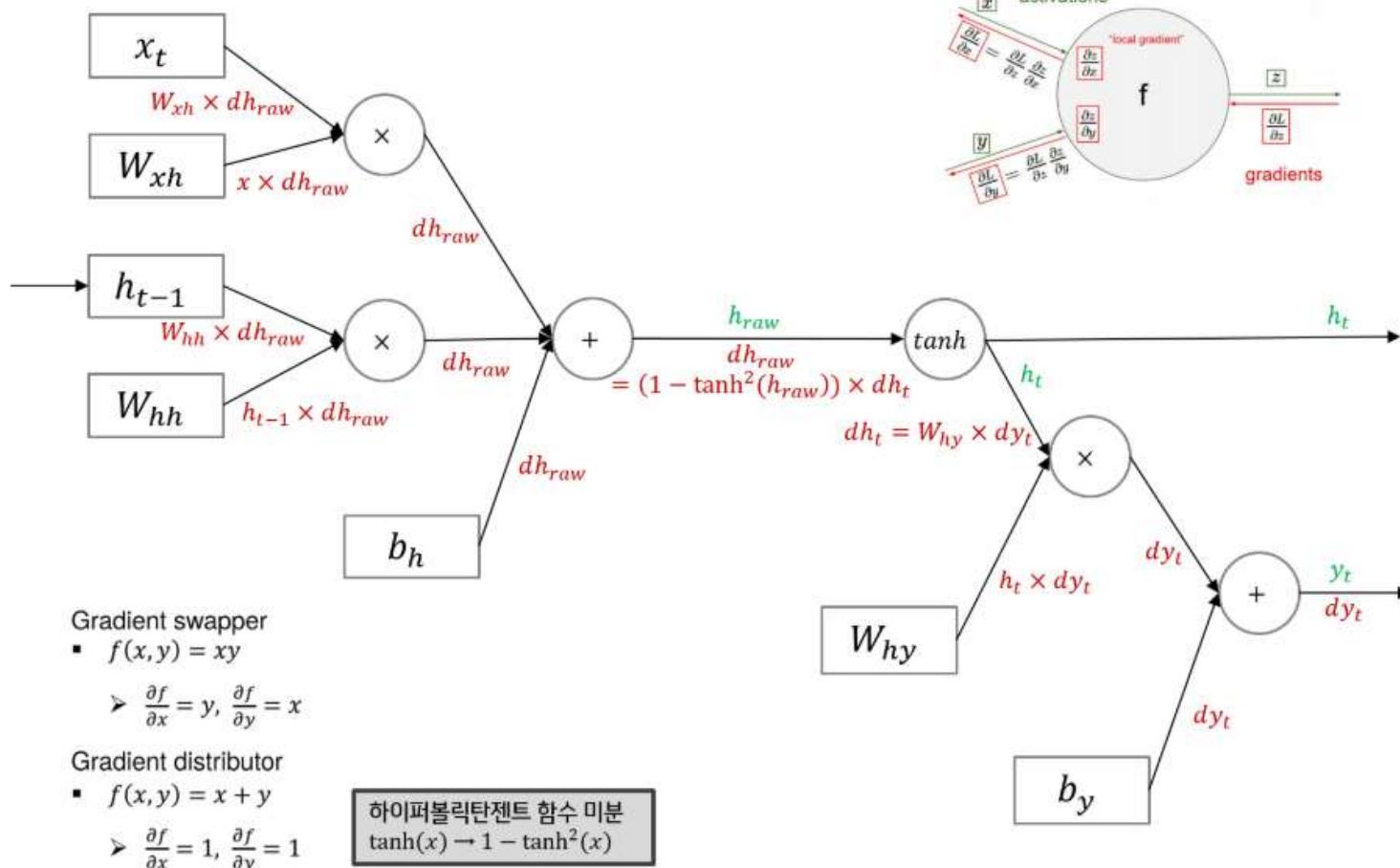
➤ Recurrent Neural Networks(RNN) 순전파

- 인풋 x 를 히든레이어 h 로 보내는 W_{xh} , 이전 히든레이어 h 에서 다음 히든레이어 h 로 보내는 W_{hh} , 히든레이어 h 에서 아웃풋 y 로 보내는 W_{hy} 가 parameter
- 모든 시점의 state에서 이 parameter는 동일하게 적용됩니다(shared weights).



머신러닝

➤ Recurrent Neural Networks(RNN) 역전파



Gradient swapper

- $f(x, y) = xy$

$$\triangleright \frac{\partial f}{\partial x} = y, \frac{\partial f}{\partial y} = x$$

Gradient distributor

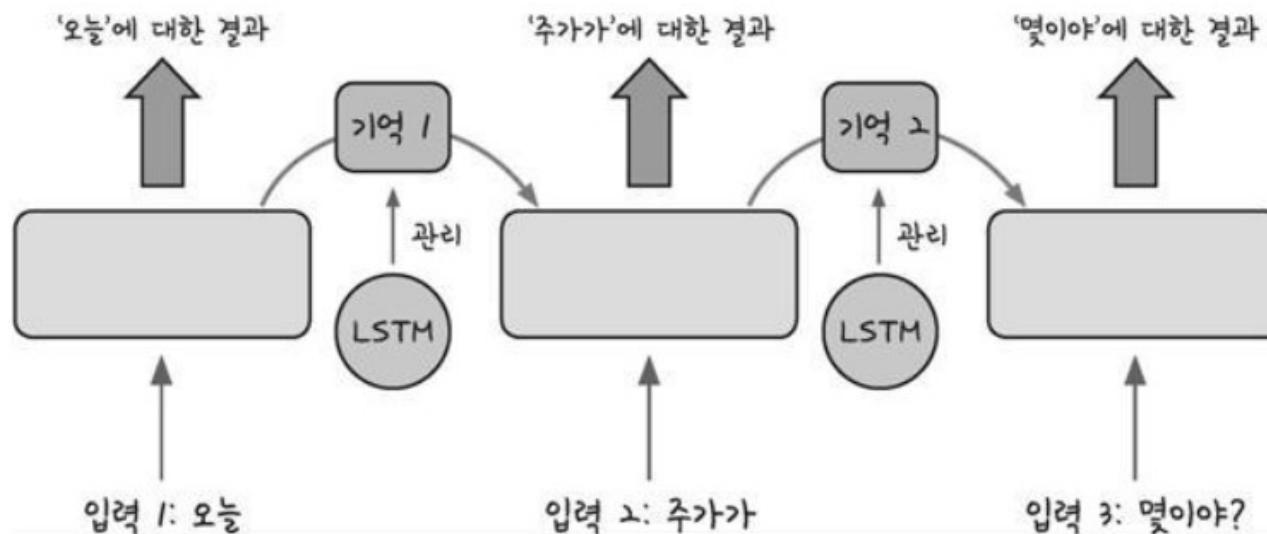
- $f(x, y) = x + y$

$$\triangleright \frac{\partial f}{\partial x} = 1, \frac{\partial f}{\partial y} = 1$$

하이퍼볼릭탄젠트 함수 미분
 $\tanh(x) \rightarrow 1 - \tanh^2(x)$

순환 신경망(Recurrent Neural Network, RNN)

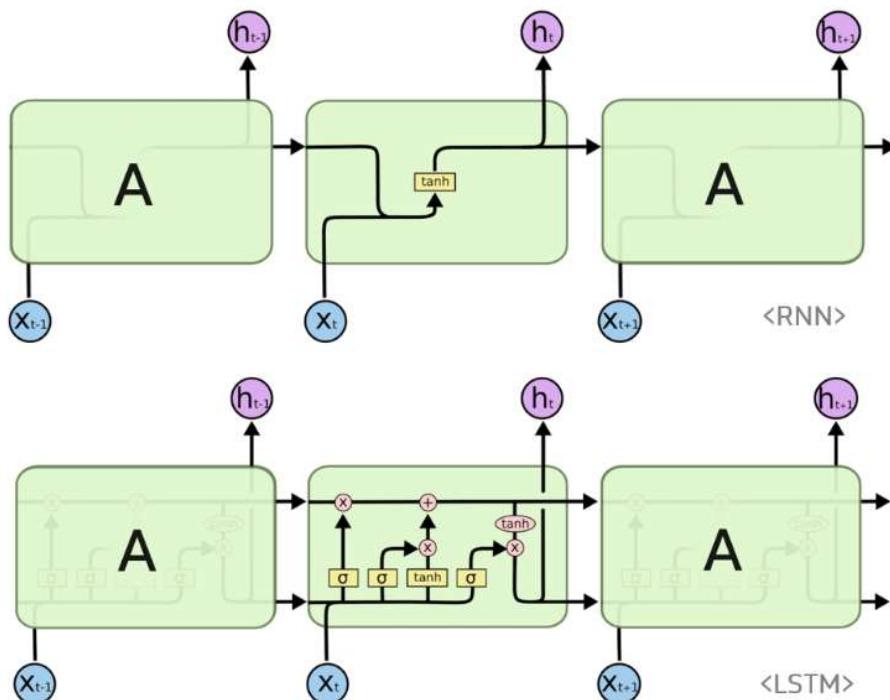
- **LSTM(Long Short Term Memory)** : 한 층 안에서 반복을 많이 해야 하는 RNN의 특성상 일반 신경망보다 기울기 소실 문제가 더 많이 발생하고 이를 해결하기 어렵다는 단점을 보완한 방법입니다.
- LSTM(Long Short Term Memory)은 반복되기 직전에 다음 층으로 기억된 값을 넘길지 안 넘길지를 관리하는 단계를 하나 더 추가하는 것입니다



머신러닝

➤ Long Short-Term Memory models(LSTM)

- LSTM은 RNN의 히든 state에 cell-state를 추가한 구조입니다
- cell state는 컨베이어 벨트 역할을 수행 (state가 꽤 오래 경과하더라도 그래디언트가 비교적 전파가 잘 되게 함)
- \odot 은 요소별 곱셈을 뜻하는 Hadamard product 연산자입니다.

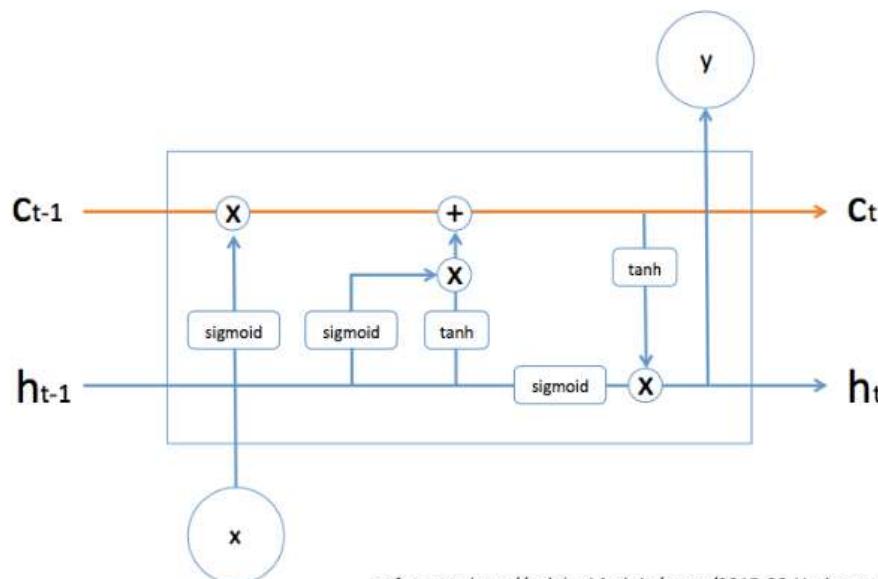


$$f_t = \sigma(W_{xh_f}x_t + W_{hh_f}h_{t-1} + b_{h_f})$$
$$i_t = \sigma(W_{xh_i}x_t + W_{hh_i}h_{t-1} + b_{h_i})$$
$$o_t = \sigma(W_{xh_o}x_t + W_{hh_o}h_{t-1} + b_{h_o})$$
$$g_t = \tanh(W_{xh_g}x_t + W_{hh_g}h_{t-1} + b_{h_g})$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = o_t \odot \tanh(c_t)$$

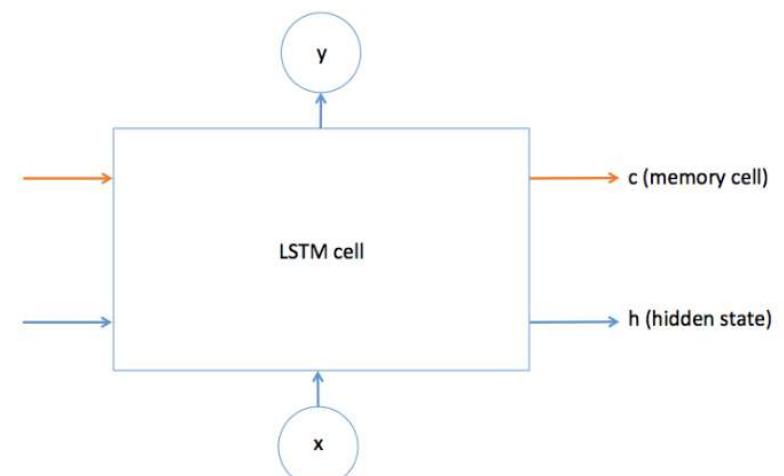
머신러닝

➤ Long Short-Term Memory models(LSTM)

- LSTM은 gradient vanishing 또는 gradient exploding과 같은 기존 RNN의 단점을 극복하고자 만들어진 조금 더 진화된 RNN 셀입니다.
- LSTM 셀 내부는 이전 정보를 지우거나 기억하기 위한 로직과 현재 정보를 기억하기 위한 로직이 구현되어 있습니다.
- 주황색 선은 메모리 셀이라고 부릅니다. 주황색 선상의 곱하기 기호에서, 0부터 1까지의 값인 시그모이드 출력값이 곱해지게 되어, 메모리 셀의 기존 정보를 어느정도까지 기억할 지 결정하게 됩니다.
- 주황색 선상의 더하기 기호는 새로운 정보를 메모리셀의 기존 정보에 더하는 로직입니다.
- h_t 선상의 곱하기 기호에서 메모리셀의 정보와 현재 정보가 함께 계산되어 상태값을 출력



reference: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>



머신러닝

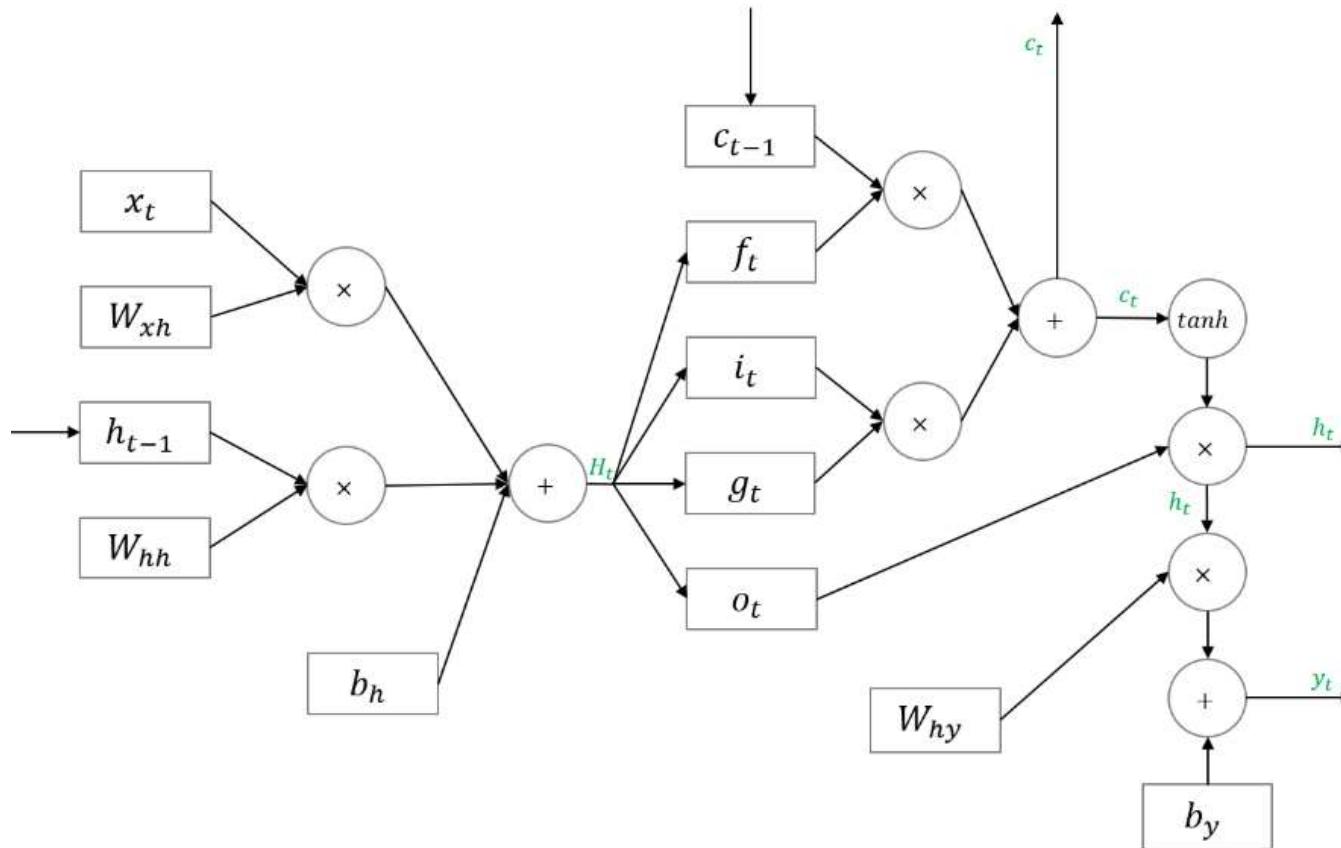
➤ Long Short-Term Memory models(LSTM)

- forget gate ft 는 '과거 정보를 잊기'를 위한 게이트입니다. $ht-1$ 과 xt 를 받아 시그모이드를 취해준 값이 바로 forget gate가 내보내는 값이 됩니다. 시그모이드 함수의 출력 범위는 0에서 1 사이이기 때문에 그 값이 0이라면 이전 상태의 정보는 잊고, 1이라면 이전 상태의 정보를 온전히 기억하게 됩니다.
- input gate $it \odot gt$ 는 '현재 정보를 기억하기' 위한 게이트입니다. $ht-1$ 과 xt 를 받아 시그모이드를 취하고, 또 같은 입력으로 하이퍼볼릭탄젠트를 취해준 다음 Hadamard product 연산을 한 값이 바로 input gate가 내보내는 값이 됩니다.
- it 의 범위는 0~1, gt 의 범위는 -1~1이기 때문에 각각 강도와 방향을 나타냅니다

머신러닝

➤ Long Short-Term Memory models(LSTM)의 순전파

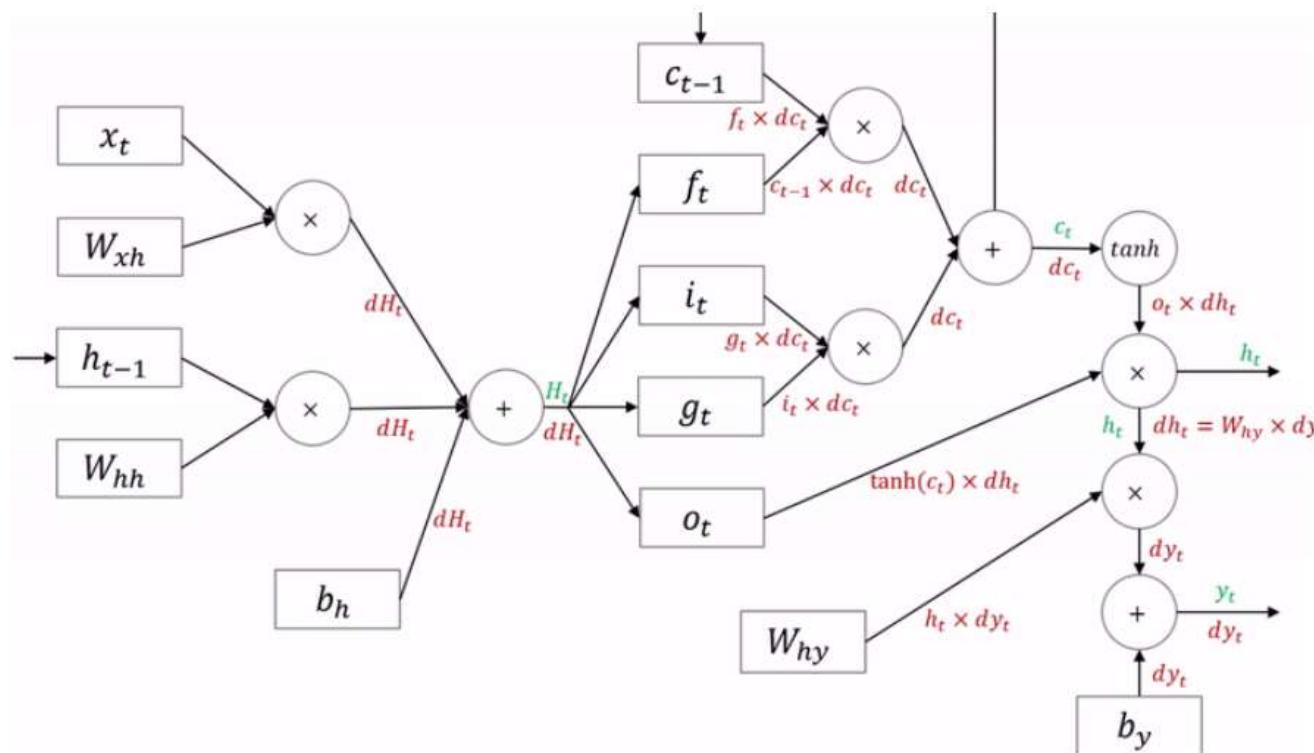
- 행 기준으로 4등분해 i, f, o, g 각각에 해당하는 활성함수를 적용하는 방식으로 i, f, o, g 를 계산합니다.



머신러닝

➤ Long Short-Term Memory models(LSTM)의 역전파

- dH_t 를 구하는 과정
- H_t 는 i_t, f_t, o_t, g_t 로 구성된 행렬입니다.
- 각각에 해당하는 그래디언트를 이를 합치면(merge) dH_t 를 만들 수 있다
- i, f, o 의 활성함수는 시그모이드이고, g 만 하이퍼볼릭탄젠트입니다.
- 각각의 활성함수에 대한 로컬 그래디언트를 구해 흘러들어온 그래디언트를 곱해주면 됩니다..



순환 신경망(Recurrent Neural Network, RNN)

➤ LSTM을 이용한 로이터 뉴스 카테고리 분류

- 입력된 문장의 의미를 파악하는 것은 곧 모든 단어를 종합하여 하나의 카테고리로 분류하는 작업이라고 할 수 있습니다.

중부 지방은 대체로 맑겠으나, 남부 지방은 구름이 많겠습니다. → 날씨

올 초부터 유동성의 힘으로 주가가 일정하게 상승했습니다. → 주식

이번 선거에서는 누가 이길 것 같아? → 정치

퍼셉트론의 한계를 극복한 신경망이 다시 뜨고 있대. → 딥러닝

- 로이터 뉴스 데이터 : 총 11,258개의 뉴스 기사가 46개의 카테고리로 나누어진 대용량 텍스트 데이터
- 딥러닝은 단어를 그대로 사용하지 않고 숫자로 변환한 다음 학습할 수 있습니다.
- 데이터 안에서 단어별 빈도에 따라 순서 번호를 붙였습니다.
- Embedding('불러온 단어의 총 개수', '기사당 단어 수') 층은 데이터 전처리 과정을 통해 입력된 값을 받아 다음 층이 알아들을 수 있는 형태로 변환하는 역할을 합니다.
- Embedding은 모델 설정 부분의 맨 처음에 있어야 합니다.
- LSTM(기사당 단어 수, 기타 옵션)은 RNN에서 기억 값에 대한 가중치를 제어합니다.
- LSTM의 활성화 함수로는 Tanh를 사용합니다.

순환 신경망(Recurrent Neural Network, RNN)

➤ LSTM을 이용한 로이터 뉴스 카테고리 분류

- Embedding 층은 데이터 전처리 과정을 통해 입력된 값을 받아 다음 층이 알아들을 수 있는 형태로 변환하는 역할을 합니다.
- Embedding('불러온 단어의 총 개수', '기사당 단어 수') 형식으로 사용하며, 모델 설정 부분의 맨 처음에 있어야 합니다. LSTM은 RNN에서 기억 값에 대한 가중치를 제어합니다.
- LSTM(기사당 단어 수, 기타 옵션)의 형태로 적용됩니다.
- LSTM의 활성화 함수로는 Tanh를 사용합니다.

```
from keras.datasets import reuters
from keras.models import Sequential
from keras.layers import Dense, LSTM, Embedding
from keras.preprocessing import sequence
from keras.utils import np_utils
import numpy
import tensorflow as tf
import matplotlib.pyplot as plt

seed = 0      # seed 값 설정
numpy.random.seed(seed)
tf.set_random_seed(seed)

# 불러온 데이터를 학습셋과 테스트셋으로 나누기
(X_train, Y_train), (X_test, Y_test) = reuters.load_data(num_words=1000, test_split=0.2)
```