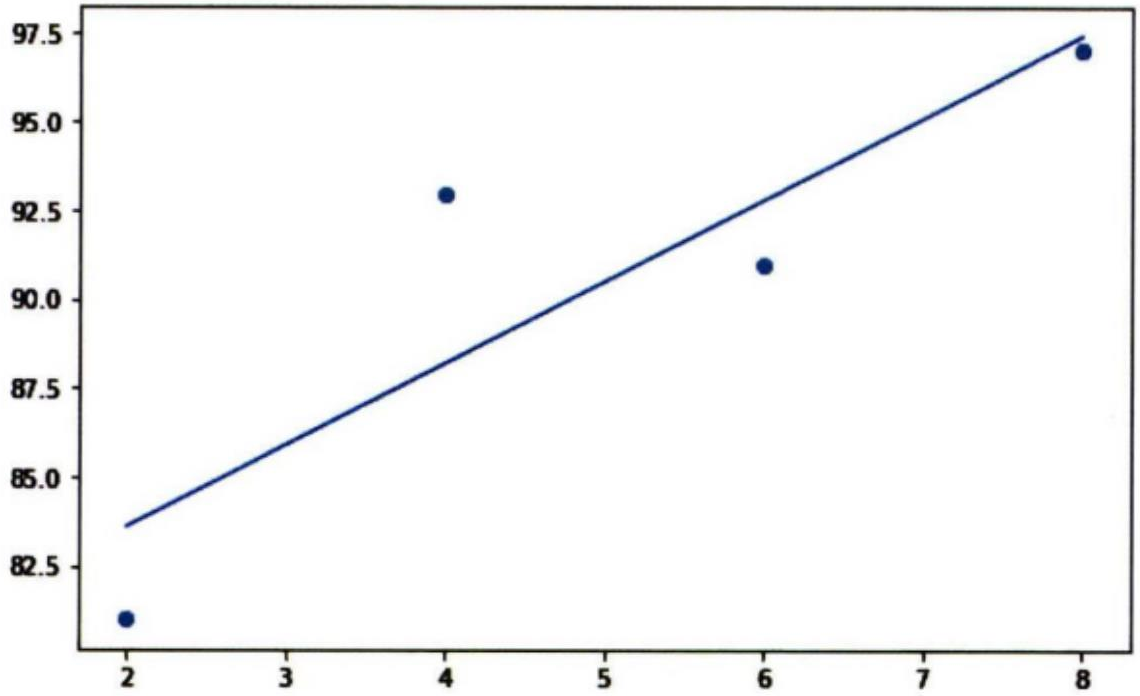
PREVIEW

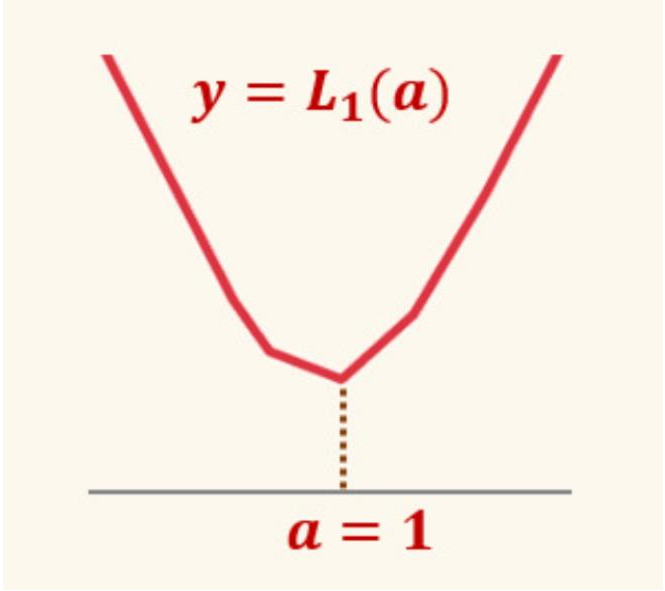
19:20-19:40	Pre-class quiz
19:40-19:50	기울기-오차 그래프의 형태가 이차곡선이 되는 이유
19:50-20:00	경사하강법
20:00-20:10	쉬는시간
20:10-20:20	에포크
20:20-20:30	다중선형회귀

기울기-오차 그래프

기울기-오차 그래프

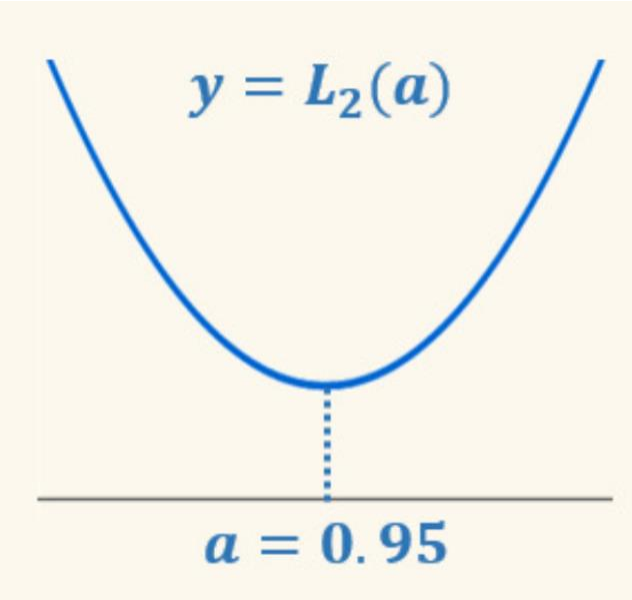


기울기-오차 그래프



$$\frac{1}{N}\sum_{i=1}^{n}|\hat{y}_i - y_i|$$

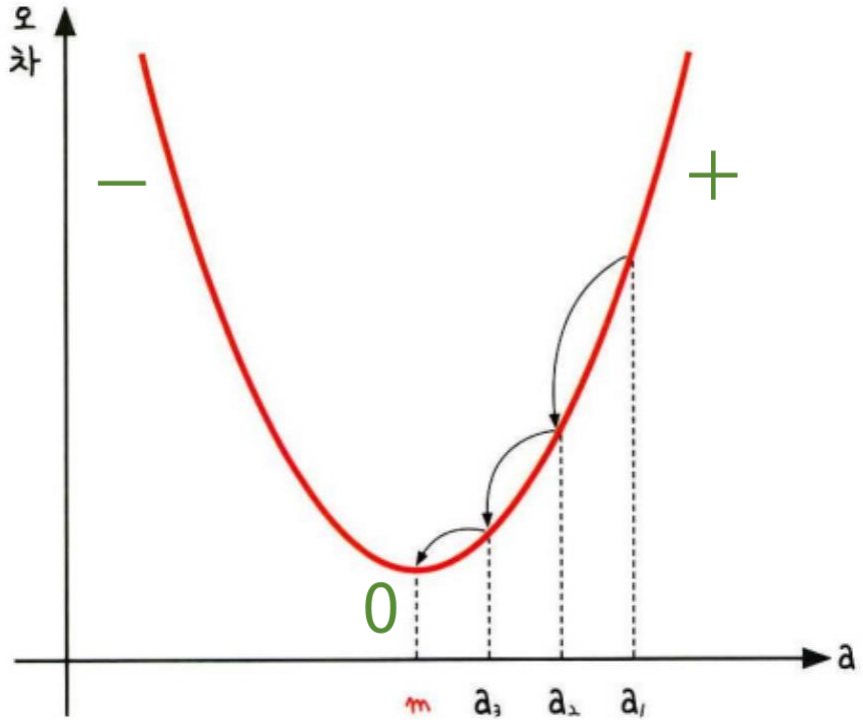
기울기-오차 그래프



평균 제곱 오차

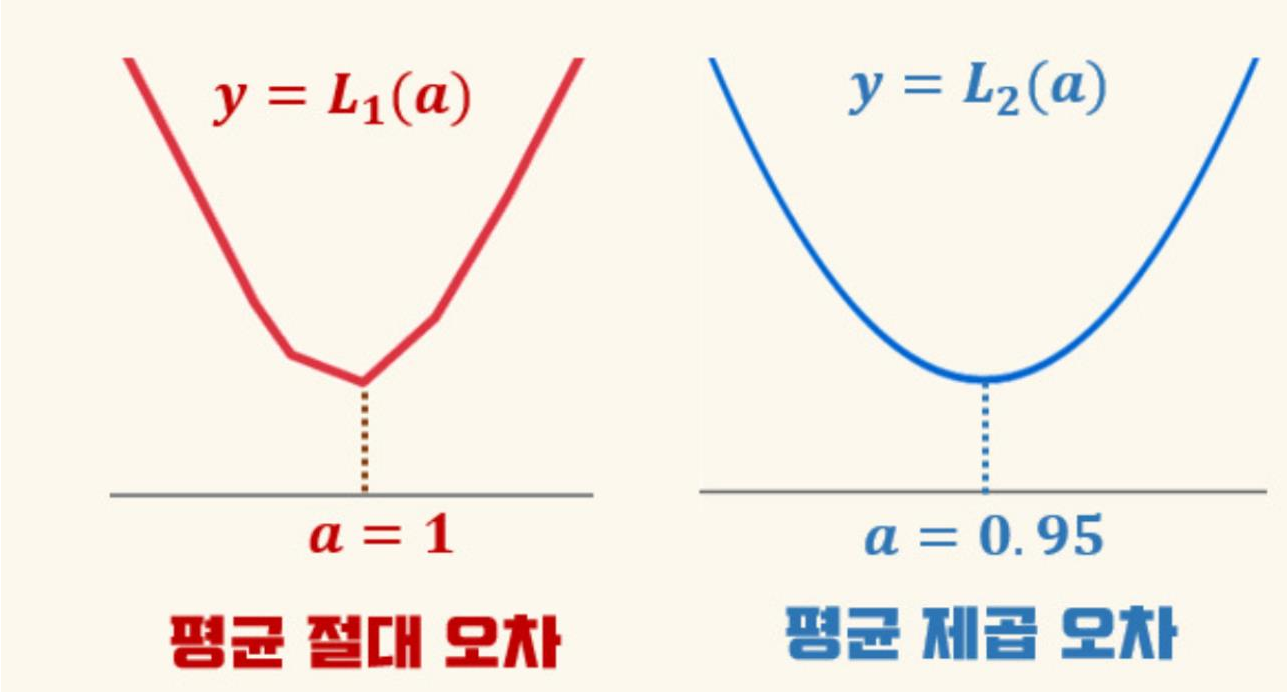
$$\frac{1}{N}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

기울기-오차 그래프



순간 기울기 0
→ 미분 했을 때의 값이 0

기울기-오차 그래프



경사 하강법

AI STUDY

경사 하강법이란?

- 함수 값이 낮아지는 방향으로 독립 변수 값을 변형시켜가면서 최종적으로는 최소 함수값을 갖도록 하는 독립 변수 값을 찾는 방법이다.

- 경사 하강법은 반복적으로 기울기 a를 변화시켜서 m의 값을 찾아내는 방법을 말한다.

경사 하강법이란?

- 경사 하강법은 결국 '미분 값이 0인 지점'을 찾는 것이다.

1. a_1에서 미분을 구한다.

2. 구해진 기울기의 반대 방향(기울기가 +면 음의 방향, -면 양의 방향)으로 얼 마간 이동시킨 a_2에서 미분을 구한다.

3. 위에서 구한 미분 값이 0이 아니면 위 과정을 반복한다.

경사 하강법의 목적과 사용 이유

- 미분계수와 근을 계산하기 어려운 함수 존재

- 컴퓨터에서 미분계수를 구현하는 과정보다 경사 하강법을 구현하는 과정이 더 쉬움

경사 하강법의 한계

1. 지역 최소값(Local Minimum)에 빠지기 쉽다는 점
2. 안장점(Saddle point)을 벗어나지 못한다는 점

학습률

- 딥러닝에서 학습률의 값을 적절히 바꾸면서 최적의 학습률을 찾는 것은 중요한 최적화 과정 중 하나다.

- 기울기의 부호를 바꿔 이동시킬 때 얼마만큼 이동시킬지를 신중히 결정해야 하는데, 이때 이동 거리를 정해 주는 것이 바로 학습률이다.

경사 하강법의 수식 유도

$$x_{i+1} = x_i - ((\text{이동 거리}) \times (\text{기울기의 부호}))$$

$$x_{i+1} = x_i - \alpha \frac{df}{dx}(x_i)$$

$$x_{i+1} = x_i - \alpha \nabla(x_i)$$

경사 하강법의 수식 유도

$$\frac{1}{n}\sum(y_i - \hat{y_i})^2$$

$$\frac{1}{n}\sum(y_i - (ax_i + b))^2$$

$$\frac{2}{n}\sum(ax_i + b - y_i)x_i$$

$$\frac{2}{n}\sum(ax_i + b - y_i)$$

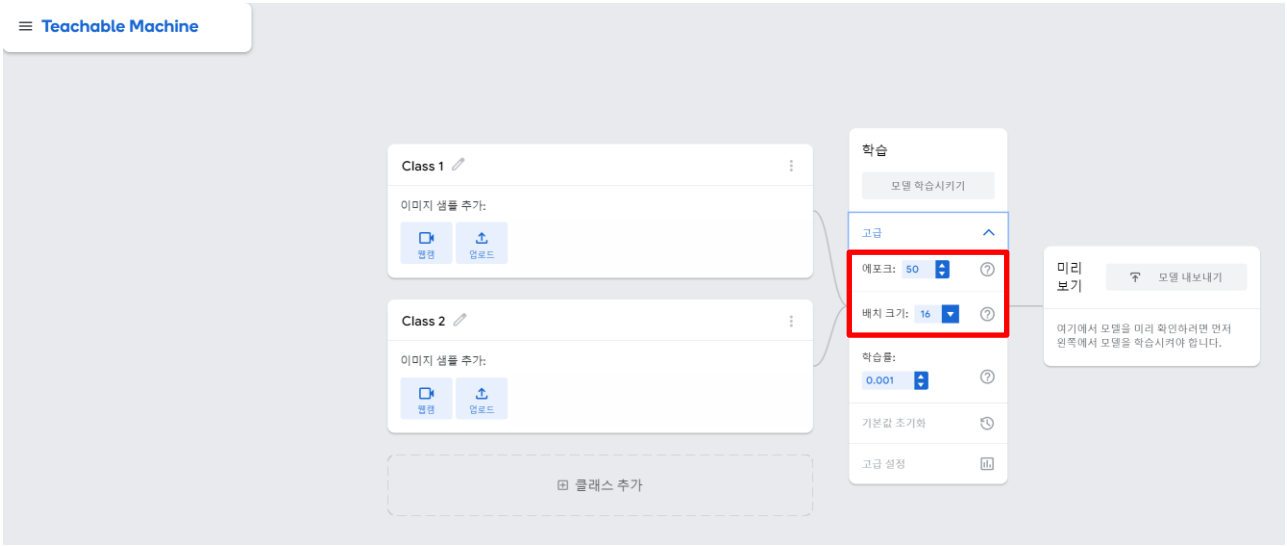
경사 하강법 코드

```
y_pred = a * x_data + b #y를 구하는 식 세우기

error = y_data - y_pred # 오차를 구하는 식

a_diff = -(2/len(x_data)) * sum(x_data * (error)) # 오차 함수를 a로 미분한 값

b_diff = - (2/len(x_data)) * sum(error) # 오차 함수를 b로 미분한 값


a = a - lr * a_diff # 학습률을 곱해 기존의 a값 업데이트

b = b - lr * b_diff # 학습률을 곱해 기존의 b값 업데이트
```

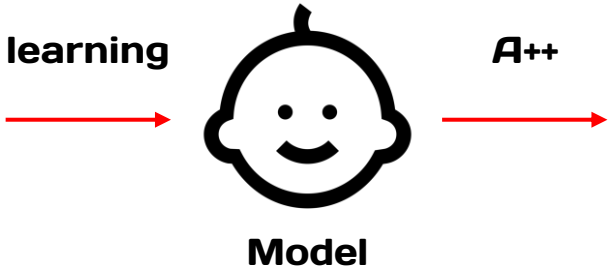
감사합니다

AI Study
: Epoch

Table of contents


Teachable Machine
Class 1
이미지 샘플 추가:
웹캠
업로드
Class 2
이미지 샘플 추가:
웹캠
업로드
클래스 추가
학습
모델 학습시키기
고급
에포크: 50
배치 크기: 16
학습률:
0.001
기본값 초기화
고급 설정
미리 보기
모델 내보내기
여기에서 모델을 미리 확인하려면 먼저 왼쪽에서 모델을 학습시켜야 합니다.

1. **기울기 계산이** 엄청 많이 일어남
2. 우리가 기계를 학습시킬 때 사용하는 데이터는 일반적으로 **최소 1000만 건 이상**
3. 이렇게 많은 데이터를 한 번에 돌리면, 아무리 좋은 컴퓨터라도 버티지 못할 것
4. 데이터의 크기가 너무 큰 경우, **메모리가 너무 많이 필요해지고**, 학습 한 번에 계산되어야 할 수가 너무 많아서 **계산 시간이 오래 걸림**



공부 잘하는 법

1. 반복 학습하기

Dataset 1회 학습: 1 epoch
ex) 4회 학습: 4 epoch

과적합: 데이터에 너무 집중한 나머지
다른 상황에는 적용을 못함

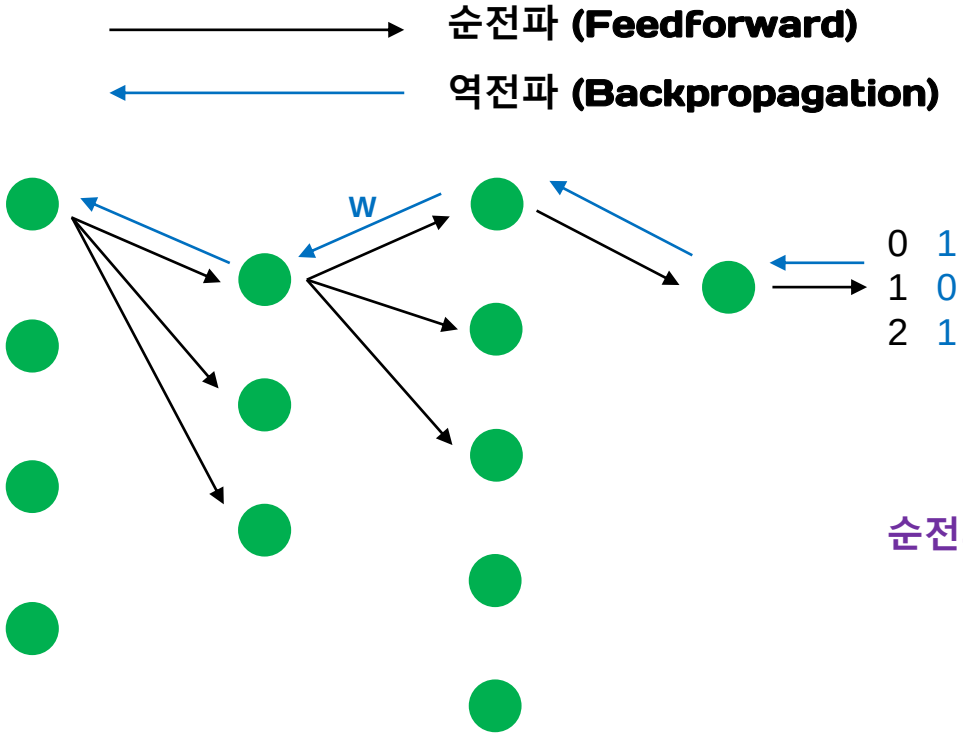
2. 나눠서 공부하기

Dataset을 n개씩 나누기: mini batch(크기 n)
ex) 12개 데이터를 3개씩 나눔: 미니배치(크기 3)

여러가지 데이터를 공부함

01

Epoch



순전파 + 역전파 : **epoch**

Dataset: 1000

100
100
100
100
100
100
100
100
100
100

Epoch 1

전체 Dataset을 학습한 횟수

100	100	100
100	100	100
100	100	100
100	100	100
100	100	100
100	100	100
100	100	100
100	100	100
100	100	100
100	100	100

Epoch 3

02

Batch Size

Dataset: 1000

100
100
100
100
100
100
100
100
100
100
100

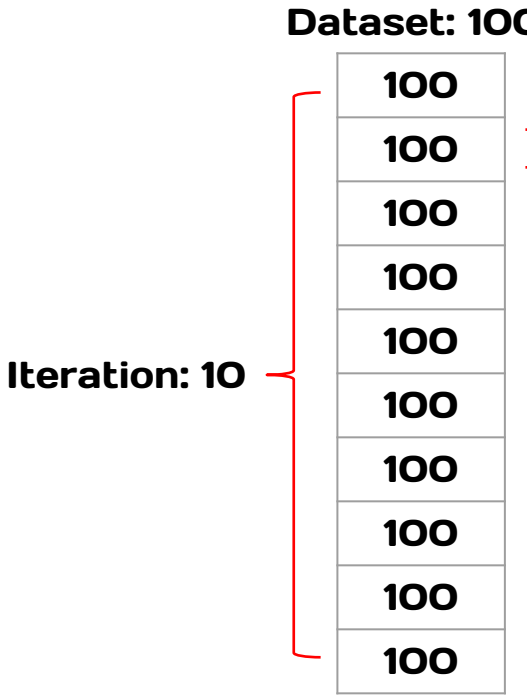
} **Batch size: 100**

모델 학습 중 parameter를 업데이트할 때 사용할 데이터 개수

전체 데이터가 1000개이고 **Batch size**가 100이라면, 데이터를 **100개씩 활용하여** 모델을 점차 학습시켜 나가는 것

03

Iteration



전체 데이터에 대한 총 Batch의 수

Batch size가 100이고 **전체 데이터 수가** 1000이라면, 전체 데이터셋을 학습시키기 위해서 **10개의 Batch가 필요함**

10번에 걸쳐 파라미터 업데이터 = **10 Iteration**

Batch size ≠ Batch 개수

Advantages & Disadvantages

Epoch 증가

장점: 정확도 증가, 데이터 수 적을 때 큰 효과

단점: 과적합 위험 증가, 계산 시간 증가

Batch Size

장점: 계산 속도 증가, 과적합 가능성 낮춤

단점: 데이터 수 적을 때 정확도 낮아짐

<Question>

100개의 데이터를 크기가 10인 미니배치를 만든 후, 30 에포크를 실시했다면, 총 몇 번 반복한 것인가?

10(미니배치) * 30(에포크) = 300번 반복
100개의 데이터를 가지고 300번 반복한 효과

```
Epoch 1/2000
1/1 [==============================] - 1s 527ms/step - loss: 4724.3628
Epoch 2/2000
1/1 [==============================] - 0s 7ms/step - loss: 1959.9148
Epoch 3/2000
1/1 [==============================] - 0s 7ms/step - loss: 859.0663
Epoch 4/2000
1/1 [==============================] - 0s 5ms/step - loss: 413.6408
Epoch 5/2000
1/1 [==============================] - 0s 20ms/step - loss: 228.4168
Epoch 6/2000
1/1 [==============================] - 0s 9ms/step - loss: 147.9081
Epoch 7/2000
1/1 [==============================] - 0s 6ms/step - loss: 110.5472
Epoch 8/2000
1/1 [==============================] - 0s 6ms/step - loss: 91.6707
Epoch 9/2000
1/1 [==============================] - 0s 8ms/step - loss: 81.1955
Epoch 10/2000
1/1 [==============================] - 0s 12ms/step - loss: 74.8566
```

```
Epoch 1990/2000
1/1 [==============================] - 0s 11ms/step - loss: 23.4834
Epoch 1991/2000
1/1 [==============================] - 0s 8ms/step - loss: 23.4722
Epoch 1992/2000
1/1 [==============================] - 0s 11ms/step - loss: 23.4610
Epoch 1993/2000
1/1 [==============================] - 0s 11ms/step - loss: 23.4498
Epoch 1994/2000
1/1 [==============================] - 0s 9ms/step - loss: 23.4387
Epoch 1995/2000
1/1 [==============================] - 0s 11ms/step - loss: 23.4275
Epoch 1996/2000
1/1 [==============================] - 0s 11ms/step - loss: 23.4164
Epoch 1997/2000
1/1 [==============================] - 0s 10ms/step - loss: 23.4052
Epoch 1998/2000
1/1 [==============================] - 0s 11ms/step - loss: 23.3941
Epoch 1999/2000
1/1 [==============================] - 0s 10ms/step - loss: 23.3829
Epoch 2000/2000
1/1 [==============================] - 0s 8ms/step - loss: 23.3718
```

Thanks!

Do you have any questions?

SMARCLE 2023 AI STUDY

4장 : 경사 하강법

2팀 – 강민서 김지율 고진영 한지원 이윤서

목차

다중 선형 회귀란

- 다중 선형 회귀에 대한 간단한 개념 설명
- 코딩으로 확인하는 다중 선형 회귀
- 텐서플로가 무엇인지 : 딥러닝은 어떤 방식으로 실행 될까
- 텐서플로에서 실행하는 선형 회귀

다중 선형 회귀란,

다중 선형 회귀는..

여러 개의 독립변수(x)를 가지고 종속변수(y)를 예측하기 위한 회귀 모형이다

- 더 정확한 예측을 위해 추가한 정보와 함께 예측값을 구하려면 변수의 개수를 늘려 다중 선형 회귀를 만들어야한다
- 독립변수들이 얼마나 영향을 주는지 알 수 있게 해주며 이를 통해 예측을 할 수 있게 한다
- 단순회귀와 마찬가지로 계수는 최소제곱법을 통해 편미분하여 계수를 추정한다

다중 선형 회귀 분석의 식

$$y = a_1 x_1 + a_2 x_2 + b$$

다중 선형 회귀 실습 코드

- 다중 선형 회귀도 단순 선형 회귀와 마찬가지로 최소제곱법을 통해 구하므로 실습코드가 거의 비슷하다. 여기에서 변수가 늘어났으므로 그에 따라 설정해주는 값의 개수만 달라질 뿐이다.
- 단순 선형 회귀와 마찬가지로 필요한 라이브러리들을 임포트 시켜주고 데이터 리스트를 선언하고 X1, X2, Y 리스트를 만든다.
- 그래프를 그리기 위한 코드로 3차원 그림으로 설정하고 각 라벨을 정해주고 그래프로 표현한다

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d

#공부시간 X와 성적 Y의 리스트를 만듭니다.
data = [[2, 0, 81], [4, 4, 93], [6, 2, 91], [8, 3, 97]]
x1 = [i[0] for i in data]
x2 = [i[1] for i in data]
y = [i[2] for i in data]

#그래프로 확인해 봅니다.
ax = plt.axes(projection='3d')
ax.set_xlabel('study_hours')
ax.set_ylabel('private_class')
ax.set_zlabel('Score')
ax.dist = 11
ax.scatter(x1, x2, y)
plt.show()
```

```python
#리스트로 되어 있는 x와 y값을 넘파이 배열로 바꾸어 줍니다.(인덱스를 주어 하나씩 불러와 계산이 가능해 지도록 하기 위함입니다.)
x1_data = np.array(x1)
x2_data = np.array(x2)
y_data = np.array(y)

# 기울기 a와 절편 b의 값을 초기화 합니다.
a1 = 0
a2 = 0
b = 0

#학습률을 정합니다.
lr = 0.02

#몇 번 반복될지를 설정합니다.(0부터 세므로 원하는 반복 횟수에 +1을 해 주어야 합니다.)
epochs = 2001

#경사 하강법을 시작합니다.
for i in range(epochs): # epoch 수 만큼 반복
    y_pred = a1 * x1_data + a2 * x2_data + b  #y를 구하는 식을 세웁니다
    error = y_data - y_pred  #오차를 구하는 식입니다.
    a1_diff = -(2/len(x1_data)) * sum(x1_data * (error)) # 오차함수를 a1로 미분한 값입니다.
    a2_diff = -(2/len(x2_data)) * sum(x2_data * (error)) # 오차함수를 a2로 미분한 값입니다.
    b_new = -(2/len(x1_data)) * sum(y_data - y_pred)  # 오차함수를 b로 미분한 값입니다.
    a1 = a1 - lr * a1_diff  # 학습률을 곱해 기존의 a1값을 업데이트합니다.
    a2 = a2 - lr * a2_diff  # 학습률을 곱해 기존의 a2값을 업데이트합니다.
    b = b - lr * b_new  # 학습률을 곱해 기존의 b값을 업데이트합니다.
    if i % 100 == 0:    # 100번 반복될 때마다 현재의 a1, a2, b값을 출력합니다.
        print("epoch=%.f, 기울기1=%.04f, 기울기2=%.04f, 절편=%.04f" % (i, a1, a2, b))
```

- 리스트 X와 Y값을 넘파이 배열로 바꾼다
- 기울기 값과 절편 값 초기화
- 에포크 값을 설정해준다
- 에포크 수만큼 반복문을 돌린다

실행 결과

에포크값이 높아짐에 따라 수렴하는 값을 통해 다중 선형 회귀 문제에서의 기울기 값과 절편 값을 찾아 확인할 수 있다

```
epoch=0, 기울기1=18.5600, 기울기2=8.4500, 절편=3.6200
epoch=100, 기울기1=7.2994, 기울기2=4.2867, 절편=38.0427
epoch=200, 기울기1=4.5683, 기울기2=3.3451, 절편=56.7901
epoch=300, 기울기1=3.1235, 기울기2=2.8463, 절편=66.7100
epoch=400, 기울기1=2.3591, 기울기2=2.5823, 절편=71.9589
epoch=500, 기울기1=1.9546, 기울기2=2.4427, 절편=74.7362
epoch=600, 기울기1=1.7405, 기울기2=2.3688, 절편=76.2058
epoch=700, 기울기1=1.6273, 기울기2=2.3297, 절편=76.9833
epoch=800, 기울기1=1.5673, 기울기2=2.3090, 절편=77.3948
epoch=900, 기울기1=1.5356, 기울기2=2.2980, 절편=77.6125
epoch=1000, 기울기1=1.5189, 기울기2=2.2922, 절편=77.7277
epoch=1100, 기울기1=1.5100, 기울기2=2.2892, 절편=77.7886
epoch=1200, 기울기1=1.5053, 기울기2=2.2875, 절편=77.8209
epoch=1300, 기울기1=1.5028, 기울기2=2.2867, 절편=77.8380
epoch=1400, 기울기1=1.5015, 기울기2=2.2862, 절편=77.8470
epoch=1500, 기울기1=1.5008, 기울기2=2.2860, 절편=77.8518
epoch=1600, 기울기1=1.5004, 기울기2=2.2859, 절편=77.8543
epoch=1700, 기울기1=1.5002, 기울기2=2.2858, 절편=77.8556
epoch=1800, 기울기1=1.5001, 기울기2=2.2858, 절편=77.8563
epoch=1900, 기울기1=1.5001, 기울기2=2.2857, 절편=77.8567
epoch=2000, 기울기1=1.5000, 기울기2=2.2857, 절편=77.8569
```

```
epoch (1) loss : 21.2874 | weight1 : 1.6577 | weight2 : 1.7782 | bias : 0.8736
epoch (2) loss : 7.3093 | weight1 : 2.0118 | weight2 : 2.2163 | bias : 1.3548
epoch (3) loss : 3.0495 | weight1 : 2.1988 | weight2 : 2.4664 | bias : 1.6194
epoch (4) loss : 1.7479 | weight1 : 2.2939 | weight2 : 2.6126 | bias : 1.7646
epoch (5) loss : 1.3470 | weight1 : 2.3385 | weight2 : 2.7011 | bias : 1.8440
epoch (6) loss : 1.2205 | weight1 : 2.3557 | weight2 : 2.7575 | bias : 1.8869
epoch (7) loss : 1.1779 | weight1 : 2.3580 | weight2 : 2.7960 | bias : 1.9099
epoch (8) loss : 1.1611 | weight1 : 2.3524 | weight2 : 2.8242 | bias : 1.9218
epoch (9) loss : 1.1525 | weight1 : 2.3427 | weight2 : 2.8467 | bias : 1.9276
epoch (10) loss : 1.1466 | weight1 : 2.3310 | weight2 : 2.8656 | bias : 1.9300
epoch (11) loss : 1.1417 | weight1 : 2.3184 | weight2 : 2.8824 | bias : 1.9306
epoch (12) loss : 1.1374 | weight1 : 2.3056 | weight2 : 2.8978 | bias : 1.9302
epoch (13) loss : 1.1335 | weight1 : 2.2929 | weight2 : 2.9122 | bias : 1.9293
epoch (14) loss : 1.1299 | weight1 : 2.2804 | weight2 : 2.9258 | bias : 1.9280
epoch (15) loss : 1.1265 | weight1 : 2.2684 | weight2 : 2.9388 | bias : 1.9266
epoch (16) loss : 1.1234 | weight1 : 2.2567 | weight2 : 2.9513 | bias : 1.9252
epoch (17) loss : 1.1205 | weight1 : 2.2455 | weight2 : 2.9633 | bias : 1.9237
epoch (18) loss : 1.1178 | weight1 : 2.2346 | weight2 : 2.9749 | bias : 1.9221
epoch (19) loss : 1.1153 | weight1 : 2.2242 | weight2 : 2.9860 | bias : 1.9206
epoch (20) loss : 1.1130 | weight1 : 2.2142 | weight2 : 2.9968 | bias : 1.9191

epoch (21) loss : 1.1109 | weight1 : 2.2046 | weight2 : 3.0072 | bias : 1.9176
epoch (22) loss : 1.1089 | weight1 : 2.1954 | weight2 : 3.0172 | bias : 1.9161
epoch (23) loss : 1.1071 | weight1 : 2.1865 | weight2 : 3.0268 | bias : 1.9147
epoch (24) loss : 1.1054 | weight1 : 2.1780 | weight2 : 3.0362 | bias : 1.9132
epoch (25) loss : 1.1038 | weight1 : 2.1698 | weight2 : 3.0452 | bias : 1.9118
epoch (26) loss : 1.1023 | weight1 : 2.1619 | weight2 : 3.0539 | bias : 1.9103
epoch (27) loss : 1.1009 | weight1 : 2.1544 | weight2 : 3.0623 | bias : 1.9089
epoch (28) loss : 1.0997 | weight1 : 2.1471 | weight2 : 3.0704 | bias : 1.9076
epoch (29) loss : 1.0985 | weight1 : 2.1402 | weight2 : 3.0782 | bias : 1.9062
epoch (30) loss : 1.0974 | weight1 : 2.1335 | weight2 : 3.0857 | bias : 1.9048
epoch (31) loss : 1.0964 | weight1 : 2.1271 | weight2 : 3.0930 | bias : 1.9035
epoch (32) loss : 1.0954 | weight1 : 2.1209 | weight2 : 3.1001 | bias : 1.9022
epoch (33) loss : 1.0945 | weight1 : 2.1150 | weight2 : 3.1069 | bias : 1.9009
epoch (34) loss : 1.0937 | weight1 : 2.1094 | weight2 : 3.1135 | bias : 1.8996
epoch (35) loss : 1.0930 | weight1 : 2.1039 | weight2 : 3.1198 | bias : 1.8983
epoch (36) loss : 1.0923 | weight1 : 2.0987 | weight2 : 3.1259 | bias : 1.8971
epoch (37) loss : 1.0916 | weight1 : 2.0937 | weight2 : 3.1319 | bias : 1.8958
epoch (38) loss : 1.0910 | weight1 : 2.0889 | weight2 : 3.1376 | bias : 1.8946
epoch (39) loss : 1.0905 | weight1 : 2.0843 | weight2 : 3.1431 | bias : 1.8934
epoch (40) loss : 1.0899 | weight1 : 2.0799 | weight2 : 3.1485 | bias : 1.8922
```

추가 _ 손실값

손실값 : (실제값 - 예측값)의 제곱

- 에포크 값마다 손실값을 구해주면 출력해줬을 때 사진과 같이 나올 수 있다
- 손실값이 3 에포크까지는 빠르게 작아지다가 이후 천천히 작아지는 것을 확인할 수 있다
- 에포크 값이 높아지면서 점점 최적의 값으로 수렴하는 경사하강법을 채택했기에 당연한 결과
- 가중치들과 바이어스도 어떤 값에 수렴하는 것을 위의 사진을 통해 확인할 수 있다

텐서플로가 무엇인지 : 딥러닝은 어떤 방식으로 실행 될까

탠서플로우(TENSORFLOW) : 구글(GOOGLE)에서 만든, 딥러닝 프로그램을 쉽게 구현할 수 있도록 다양한 기능을 제공해주는 라이브러리

- 텐서플로우 배우면 딥러닝할 수 있는게 아니라 그냥 딥러닝 수학계산을 조금 더 쉽게 도와주는 라이브러리일 뿐이다
- 이에 반해 직접 파이썬 쌩코딩으로 짜면 매우 코드가 길고 복잡해진다

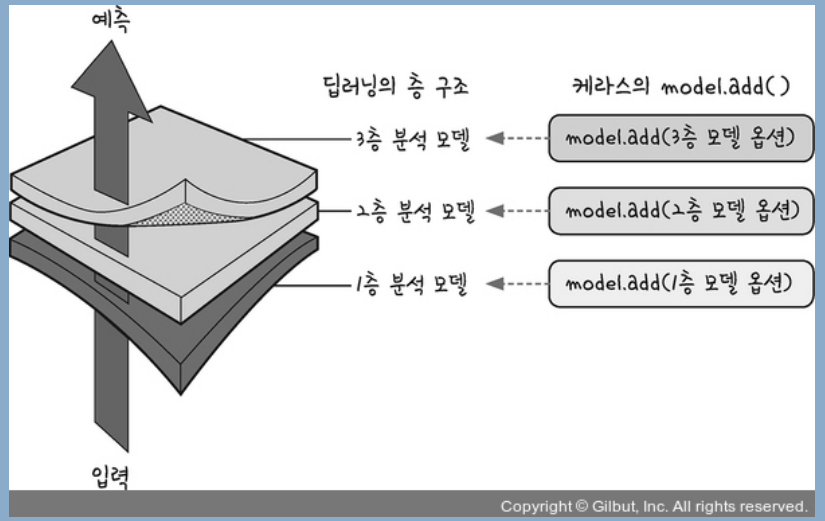
텐서플로가 무엇인지 : 딥러닝은 어떤 방식으로 실행 될까

딥러닝은 여러 층이 쌓여 있는 구조

- 준비된 데이터가 입력되는 입력층에 이어 작업을 진행하는 여러 층에 이어 출력 결과 간 나오는 출력층까지 여러 개의 층이 각자 자신이 맡은 일을 하면서 앞뒤로 정보를 주고 받는다

이번 장의 목표

- 딥러닝을 실행하기 위해 텐서플로라는 라이브러리의 케라스 API를 불러와 사용할 것이다
- 지금까지 배운 선형회귀의 개념과 딥러닝 라이브러리들이 어떻게 연결되는지 살펴보기 위함이다
- 텐서플로 및 케라스의 사용법을 익히고 딥러닝 자체에 대한 학습에 한 걸음 더 나아가기 위함이다



머신 러닝에서 쓰이는 용어

가설함수 H(X)

문제를 해결하기 위해 가정하는 식
ex) y = ax + b

가중치 W

변수 x에 어느 정도의 가중치를 곱하는 지 결정하는 값
ex) 기울기 값 a

편향(BIAS) B

데이터의 특성에 따라 따로 부여되는 값
ex) 절편 값 b

손실함수

실제값과 예측값 사이의 오차에 대한 식
ex) 평균 제곱 오차

옵티마이저

손실함수의 최소값을 찾는 것을 수행하는 알고리즘
ex) 경사하강법이 바로 여러 옵티마이저 중 하나, 경사하강법 이외의 옵티마이저에 대해서는 9장에서 배운다

$$y = ax + b \rightarrow H(x) = wx + b$$

텐서플로에서 실행하는 선형 회귀

```
import numpy as np
import matplotlib.pyplot as plt

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

x = np.array([2, 4, 6, 8])
y = np.array([81, 93, 91, 97])
```

- 필요한 라이브러리들을 임포트해준다
- 텐서플로에 포함된 케라스 API 중 필요한 함수들을 다음과 같이 불러온다

텐서플로에서 실행하는 선형 회귀

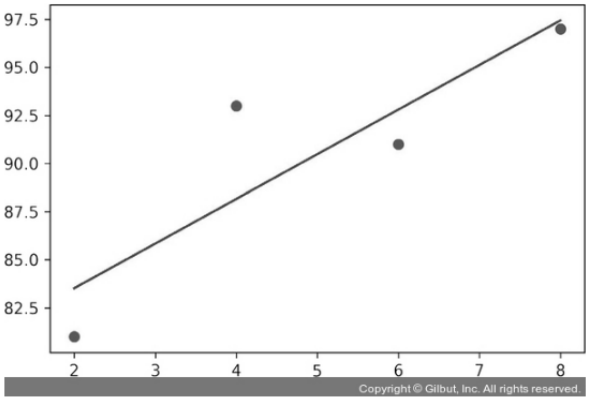
```
model = Sequential()

model.add(Dense(1, input_dim=1, activation='linear'))

model.compile(optimizer='sgd', loss='mse')

model.fit(x, y, epochs=2000)
```

- **SEQUENTIAL() 함수를 MODEL로 선언**
  - **Sequential()은 딥러닝의 한 층 한 층을 MODEL.ADD() 함수를 사용해 층을 추가시켜준다**
- **DENSE() 각 층의 입력과 출력을 촘촘하게 모두 연결하라는 것**
  - **DENSE() 함수의 인자들을 하나 씩 살펴보자면, 첫 번째 인자는 본 실습문제의 가설함수는 H(X) = WX + B로 출력되는 값이 성적 하나이므로 1이라고 설정.**
  - **입력될 변수도 학습시간으로 하나뿐이므로 INPUT_DIM=1이라고 설정**
  - **ACTIVATION은 활성화 함수를 정하는 옵션**
    - **활성화 함수 : 입력된 값을 다음 층으로 넘길때 각 값을 어떻게 처리할지 결정하는 함수**
    - **LINEAR : 선형회귀를 다루고 있다**
- **MODEL.COMPILE() 함수는 앞서 만든 MODEL의 설정을 그대로 실행하라는 의미**
  - **OPTIMIZER='SGD', LOSS='MSE' : 경사하강법을 실행시킨다는 SGD, 손실함수로 평균제곱오차를 사용한다는 MSE로 설정**
- **MODEL.FIT() 몇 번을 오갈 것인지, 그리고 한 번 오갈 때 몇 개의 데이터를 사용할 것인지 정하는 함수, 에포크 값을 적어준다**

텐서플로에서 실행하는 선형 회귀

```
plt.scatter(x, y)
plt.plot(x, model.predict(x), 'r')    # 예측 결과를 그래프로 나타냅니다.
plt.show()

# 임의의 시간을 집어넣어 점수를 예측하는 모델을 테스트해 보겠습니다.
hour = 7
prediction = model.predict([hour])
print("%.f시간을 공부할 경우의 예상 점수는 %.02f점입니다." % (hour, prediction))
```

```
Epoch 1/2000
1/1 [==============================] - 1s 114ms/step - loss: 9241.3984
... (중략) ...
Epoch 2000/2000
1/1 [==============================] - 0s 2ms/step - loss: 8.3022
```



7시간을 공부할 경우의 예상 점수는 95.12점입니다.

실행 결과