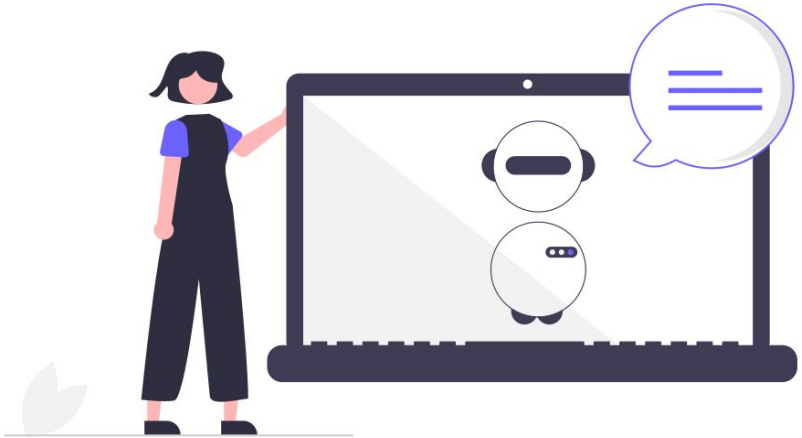
딥러닝 기초

딥러닝이란? 딥러닝 소개



오늘의 목차

Data Diving

- 머신 러닝 모델에 대한 간단한 복습
  - 기계가 학습을 한다는 것은?
  - 머신 러닝의 과정
- 딥러닝 소개
  - 딥러닝과 다른 머신러닝의 차이는?
  - 왜 deep 러닝인가?
- 딥러닝을 파이썬으로 하는 방식은? pytorch 사용해보기
  - pytorch를 쓰는 학습의 표준 방식
  - 간단한 데이터를 딥러닝 / 다른 머신 러닝 방식으로 처리해보자!

앞으로 한동안의 목차

Data Diving

- pytorch 써서 다양한 문제 풀어보기
  - NLP / 이미지
  - 강화학습
  - AutoEncoder 등등등…
- 딥러닝에 다른 머신 러닝 알고리즘들 같이 사용해보기
  - evolved transformer 구현해보기
- pytorch under the hood: 어떤 식으로 pytorch가 돌아가는지 살펴보기
  - computational graph 구현 / autograd 구현 등등

머신 러닝 모델이란?

가지고 있는 물감색

원하는 색

- 머신 러닝 모델 = 수식
  - 입력 / 출력이 있고, 입력/출력 외 변수들은 모수(Parameter)이라고 함
  - 모수를 데이터를 받아가면서 바꿔서, 주어진 입력을 넣었을 때 나와야 할 출력이 나오도록 조정하는 것

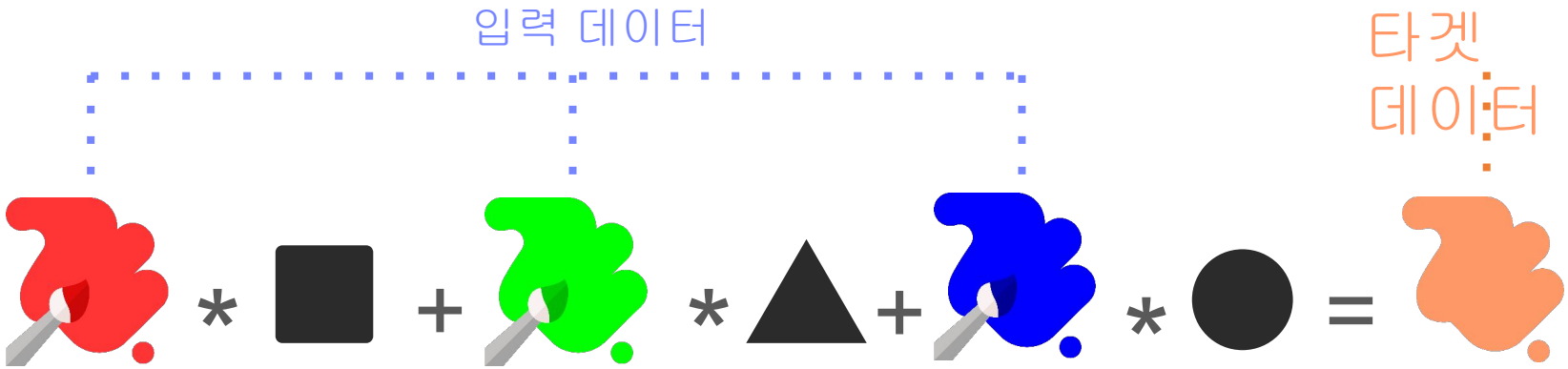
머신 러닝 모델이란?

가지고 있는 물감색

원하는 색

- 입력: 가지고 있는 물감들
- 모수: 각 물감들을 섞을 비율
- 출력: 섞여서 나올 색

머신 러닝 모델이란?



Parameter

■ , ▲ , ● 에 계속 값을 넣어보며 가장 적절한 값을 찾는다.

머신 러닝 모델이란?

- 적절함의 기준 = 가장 비슷한 출력이 나오는가
  - 수식으로는 (출력 - 원하는 값)의 차이 등으로 계산

학습은 어떤 방식으로 하는가?

Data Diving

Interviewer: What's your biggest strength?

Me: I'm an expert in machine learning.

Interviewer: What's 9 + 10?

Me: Its 3.

Interviewer: Not even close. It's 19.

Me: It's 16.

Interviewer: Wrong. Its still 19.

Me: It's 18.

Interviewer: No, it's 19.

Me: it's 19.

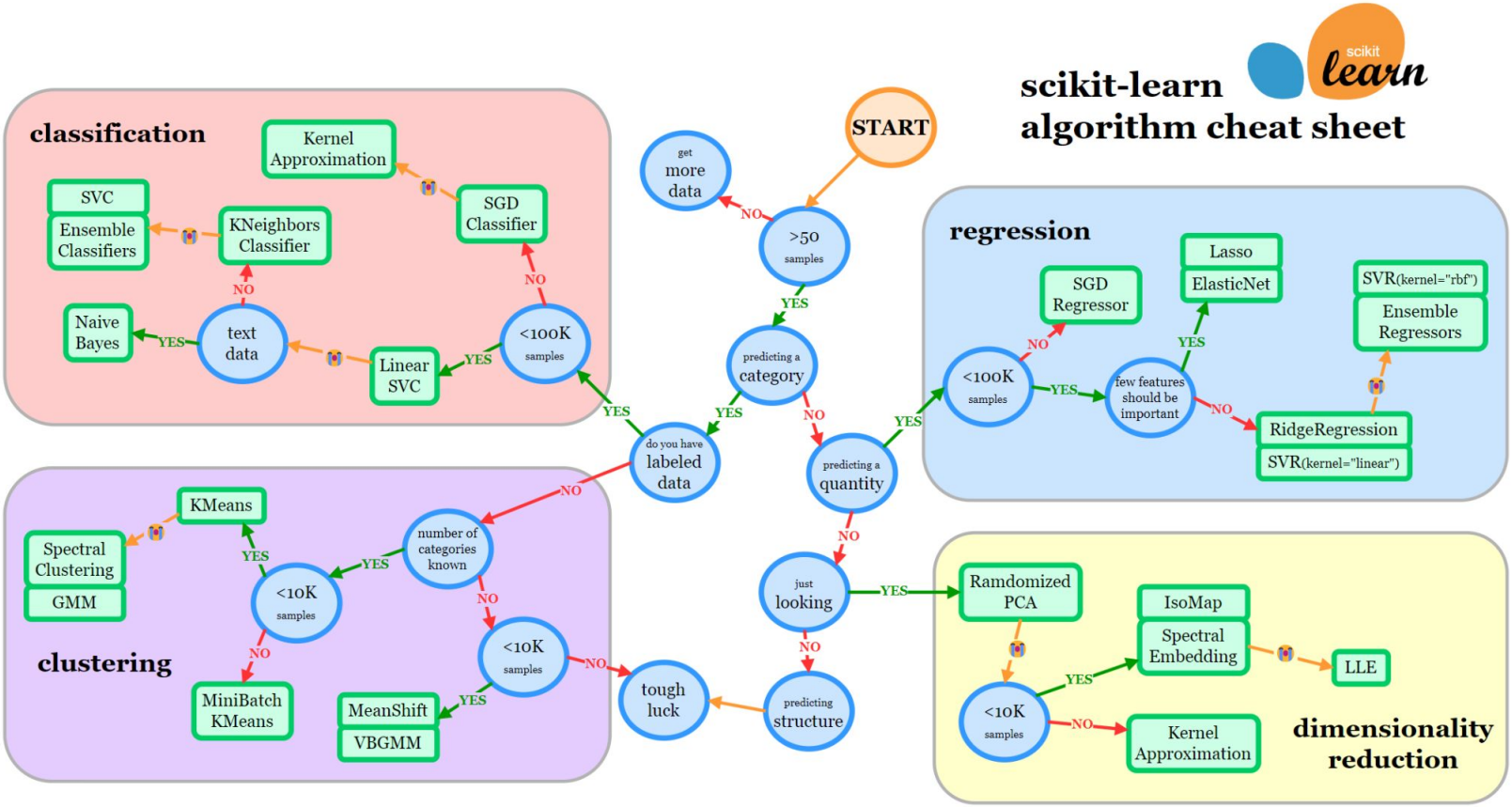
Interviewer: You're hired

- 놀랍게도 옆의 사진처럼 함
- 처음에 아무 값이나 찍음
  - 이후 그 값에 최대한 맞춰가는 방향으로 답을 조금씩 바꿔감
  - 이 답을 조금씩 바꿔가는 과정을 gradient descent라고 함
- 문제가 쉬운 경우에는 공식에 맞춰서 바로 답이 나오는 경우도 있음
  - 선형회귀 같은 경우 그럼

머신 러닝 과정

- 머신 러닝의 일반적인 과정
  - Modeling
    - 데이터가 어떻게 생겼을까? (데이터와 상견례 - EDA라고 함)
    - 데이터를 어떻게 넣어줄까? (데이터 -> 숫자)
      - 예) 영-한 번역기: 영어 문장을 어떻게 입력해줄까?
      - 예) 이미지 분류기: 이미지를 어떻게 입력해줄까?
    - 어떤 모델을 쓰는 게 좋을까?
      - Task별로 state-of-the-art 모델을 찾아보기
      - 잘 모르겠으면 진화알고리즘, random forest, 기본적인 딥러닝 등 끼얹어보기
      - 최근에는 LLM 일단 넣어보고 생각
  - 전처리
    - 없는 값 채우고 (결측치)
    - 데이터 넣어주는 코드 짜고
    - 분포 이상한지 보고 정상화하고 (숫자가 너무 크다/작다, 한쪽으로 쏠려있다 등등..)
    - 데이터 로딩하기
  - 모델 구현: 엄청 금방 함
  - 학습 / 튜닝
    - 하이퍼파라미터 튜닝: 노가다와 운의 산물

여러분들이 배운 scikit-learn에서의 방식



머신러닝 과정 구현 시 시간 비중

Data Diving

- 머신 러닝의 일반적인 과정
  - Modeling **(30%)**
    - 데이터가 어떻게 생겼을까? (데이터와 상견례 - EDA라고 함)
    - 데이터를 어떻게 넣어줄까? (데이터 -> 숫자)
      - 예) 영-한 번역기: 영어 문장을 어떻게 입력해줄까?
      - 예) 이미지 분류기: 이미지를 어떻게 입력해줄까?
    - 어떤 모델을 쓰는 게 좋을까?
      - Task별로 state-of-the-art 모델을 찾아보기
      - 잘 모르겠으면 진화알고리즘, random forest, 기본적인 딥러닝 등 끼얹어보기
      - 최근에는 LLM 일단 넣어보고 생각
  - 전처리 **(50%)**
    - 없는 값 채우고 (결측치)
    - 데이터 넣어주는 코드 짜고
    - 분포 이상한지 보고 정상화하고 (숫자가 너무 크다/작다, 한쪽으로 쏠려있다 등등..)
    - 데이터 로딩하기
  - 모델 구현: 엄청 금방 함 **(<1%)**
  - 학습 / 튜닝 **(20%)** (학습: 컴퓨터가 일하는 시간 / 튜닝: 운과 눈치로 결정됨)
    - 하이퍼파라미터 튜닝: 노가다와 운의 산물

딥러닝은 뭐가 다른가?

- 딥러닝의 최고 장점: 성능이 좋다
  - 언제나 좋은 건 아니지만, 대개의 경우 모르겠을 때 끼얹으면 괜찮은 경우가 많음
- 부차적인 장점들
  - 일반적인 상황에서 쓰기 좋음: 입력을 바꾸기가 애매할 때도, 말도 안 되게 naive하게 바꾸더라도 잘 돌아가는 경우가 많음
  - 인간의 개입이 별로 없음 / 도메인 지식을 필요로 하지 않음 = 쉬움
  - 많은 데이터를 처리하기에 용이한 구조

End2End Learning

- 데이터 → 모델 → 결과 외에 다른 요소가 없는 경우를 의미함
  - 대개의 경우, 고전적인 머신 러닝 알고리즘들은 다양한 요소를 들고 있음
  - 예) 영어 → 한국어 번역: 영어 형태소 분석기, 한국어 형태소 병합기, …
- 딥러닝의 경우, End2End인 것이 가장 좋음
  - 일단 편함… 다른 요소들을 생각 안 해도 됨
  - 테슬라 vs 다른 로봇회사들

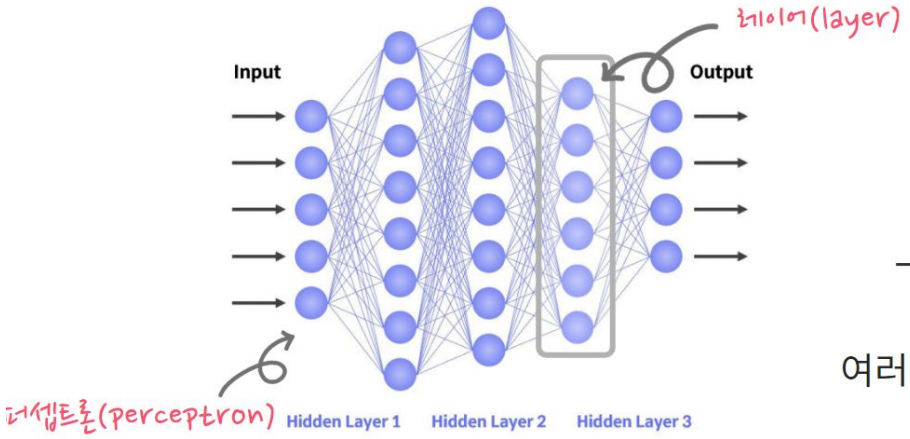
Twitter Sentiment Analysis using Support Vector Machine and Deep Learning Model in E-Learning Implementation during the Covid-19 Outbreak

Dinar Ajeng Kristiyanti,[1, a)] Dwi Andini Putri,[2, b)] Elly Indrayuni,[3, c)] Acmad Nurhadi,[4, d)] and Akhmad Hairul Umam[5, e)]

[1)]*Information System, Universitas Multimedia Nusantara, Tangerang, Indonesia*
[2)]*Information Technology, Universitas Bina Sarana Informatika, Jakarta, Indonesia*
[3)]*Accounting Information System, Universitas Bina Sarana Informatika, Jakarta, Indonesia*
[4)]*Computer Technology, Universitas Bina Sarana Informatika, Jakarta, Indonesia*
[5)]*Communication, Tanri Abeng University, Jakarta, Indonesia*

[a)]*Corresponding author: dinar.kristiyanti@umn.ac.id*
[b)]*Electronic mail: dwi.dwd@bsi.ac.id*
[c)]*Electronic mail: elly.eiy@bsi.ac.id*
[d)]*Electronic mail: achmad.ahh@bsi.ac.id*
[e)]*Electronic mail: ahmad.umam@tau.ac.id*

딥러닝은 그러면 어떻게 하는 것인가?



Artificial Neural Network

인공 신경망

인간의 신경을 흉내 낸 머신러닝 기법
여러 개의 퍼셉트론으로 구성된 하나의 네트워크

- 선형 회귀의 (좀 많은) 일반화
  - 선형 회귀: 가장 기본적인 딥러닝
  - 기본적으로 행렬을 정해진 식(=딥러닝 구조, 뉴럴 넷 아키텍쳐 등으로 부름)에 따라 계산해서
  - 행렬의 원소들을 데이터에 맞춰서 update하는 것
- 딥러닝인 이유: 행렬을 여러 개를 쌓아서 deep한 구조를 만들었다고 함

Data Diving

Artificial Neural Network

인공 신경망

01 퍼셉트론 (Perceptron)

02 가중치 (weight)
편차 (bias)

03 활성 함수 (Activation Function)

Data Diving

Artificial Neural Network

인공 신경망

04 타켓/예측 (Target/Predict)

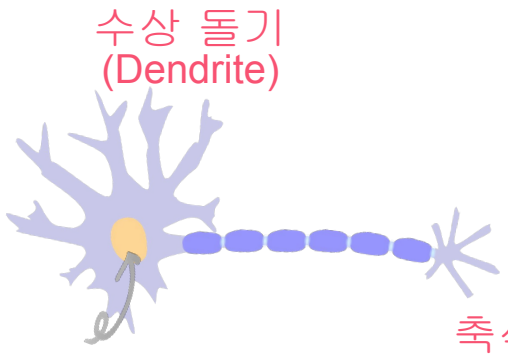
05 손실 (Loss) 손실함수(Loss Function)

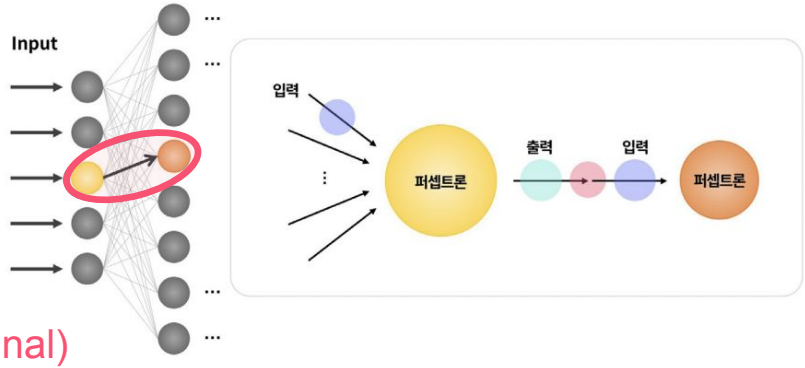
06 최적화 함수 (Optimization)

07 학습/테스트 반복 (Test)

1 퍼셉트론 (Perceptron)

Data Diving





인공신경망의 기본 단위. 인간의 뇌를 구성하는 신경 세포 뉴런을 모방

뉴런은 수상돌기 다발로부터 여러 신호를 받아들이고 종합한 후,
종합한 신호가 일정치 이상의 크기를 가지면 축삭돌기를 통해서 신호 전달

Data Diving

Weight

입력 데이터에 적용되는
가중치
입력 데이터와 출력 데이터의 관계를 결정하는
조건

Bias

가중합에 추가적으로 더해주는 상수
모든 네트워크에 포함되는 것은 아님
weight의 보조수단

학습

Weight를 계속 수정해보며 출력 데이터를 타겟 데이터와 비슷하게 만들어 가는 것

가지고 있는 물감색

원하는 색

Data Diving

Weight


,
,
에 계속 값을 넣어보며 가장 적절한 값을 찾는다.

2 가중치 (weight), 편차 (bias)

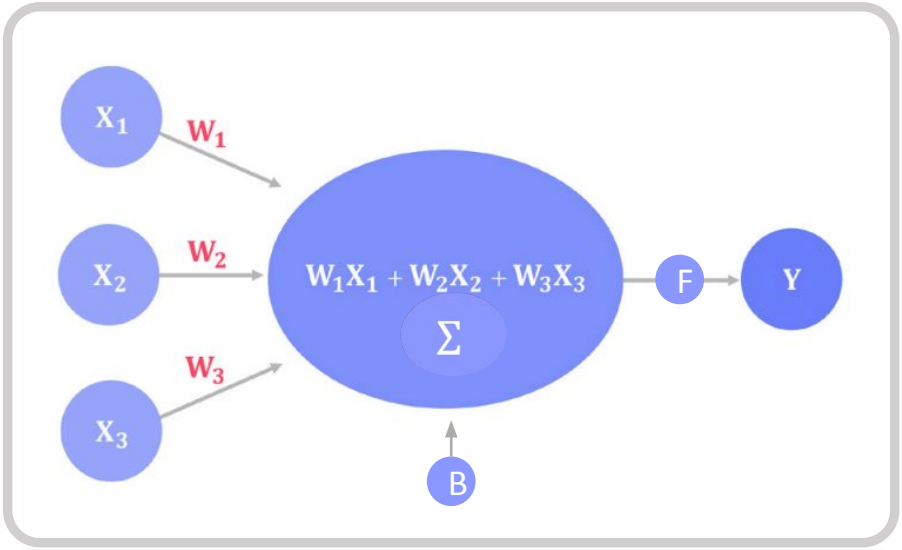
Data Diving

2 가중치 (weight), 편차 (bias)

Data Diving

2 가중치 (weight), 편차 (bias)

Data Diving



X1, X2, X3 : 입력 데이터

W1, W2, W3 : 가중치

B : 편차

F : 활성함수

Y : 타겟 데이터

W1X1+W2X2+W3X3 > B

W1X1+W2X2+W3X3 - B > 0

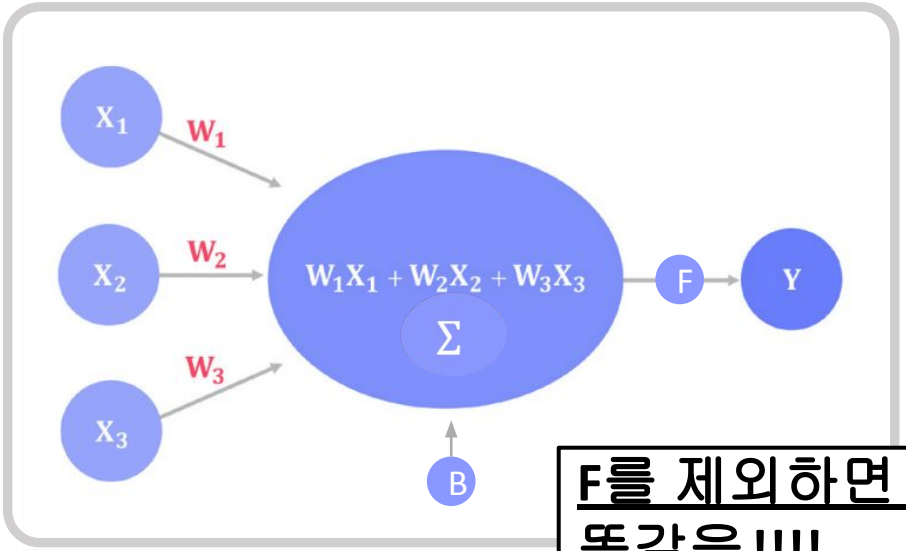
→ Y=1

W1X1+W2X2+W3X3 < B

W1X1+W2X2+W3X3 - B < 0

→ Y=0

Data Diving



X1, X2, X3 : 입력 데이터

W1, W2, W3 : 가중치

B : 편차

F : 활성함수

Y : 타겟 데이터

F를 제외하면 그냥 선형회귀랑 똑같음!!!!

W1X1+W2X2+W3X3 > B

W1X1+W2X2+W3X3 - B > 0

→ Y=1

W1X1+W2X2+W3X3 < B

W1X1+W2X2+W3X3 - B < 0

→ Y=0

평수 (평)	방 개수 (개)	지하철 역과의 거리 (km)	집 값 (억 원)
32	3	0.3	6
33	3	0.5	7
36	4	0.5	8.5

32 × 평수의 weight + 3 × 방 개수의 weight + 0.3 × 지하철 역과의 거리 weight ▶ 집 값(6억)이 되도록

33 × 평수의 weight + 3 × 방 개수의 weight + 0.5 × 지하철 역과의 거리 weight ▶ 집 값(7억)이 되도록

36 × 평수의 weight + 4 × 방 개수의 weight + 0.5 × 지하철 역과의 거리 weight ▶ 집 값(8.5억)이 되도록

Data Diving

생각해보기

앞에서 배웠던 식을 이용하여 어떤 하나의 weight를 가지고 모든 데이터에 계산을 했을 때
어떤 값이 나오는지 확인해볼까요?

입력 데이터 × weight = 예측값

weight = [평수 w, 방 개수 w, 지하철 역과의 거리 w] = [0.1 , 0.3 , 4]

	평수 (평)	방 개수 (개)	지하철 역과의 거리 (km)	집 값 (억 원)
1)	32	3	0.3	6
	→ 32 × 0.1 + 3 × 0.3 + 0.3 × 4 = 5.3(억)			
2)	33	3	0.5	7
	→ [ ] × 0.1 + [ ] × 0.3 + [ ] × 4 = [ ](억)			
3)	36	4	0.5	8.5
	→ [ ]			

Data Diving

🧐 생각해보기

앞에서 배웠던 식을 이용하여 어떤 하나의 weight를 가지고 모든 데이터에 계산을 했을 때
어떤 값이 나오는지 확인해볼까요?

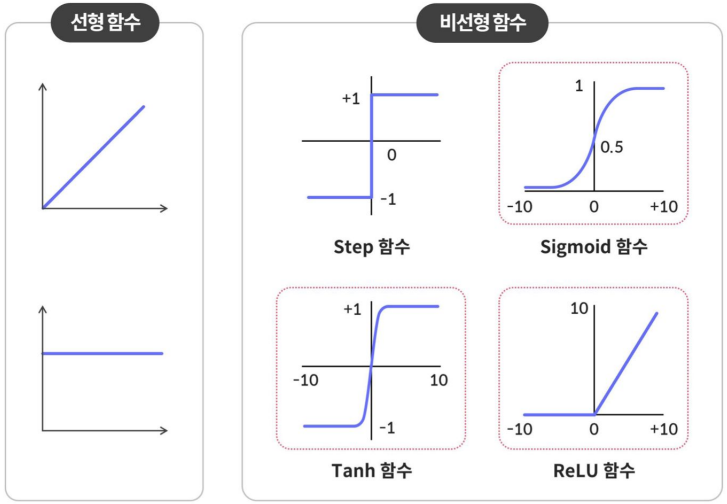
입력 데이터 × weight = 예측값

weight = [평수 w, 방 개수 w, 지하철 역과의 거리 w] = [0.1 , 0.3 , 4]

	평수 (평)	방 개수 (개)	지하철 역과의 거리 (km)	집 값 (억 원)
1)	32	3	0.3	6
	→ 32 × 0.1 + 3 × 0.3 + 0.3 × 4 = 5.3(억)			
2)	33	3	0.5	7
	→ [33] × 0.1 + [3] × 0.3 + [0.5] × 4 = [6.2](억)			
3)	36	4	0.5	8.5
	→ [36 x 0.1 +4 x 0.3 + 0.5 x 4=6.8(억)]			

3 활성 함수 (Activation Function)

Data Diving

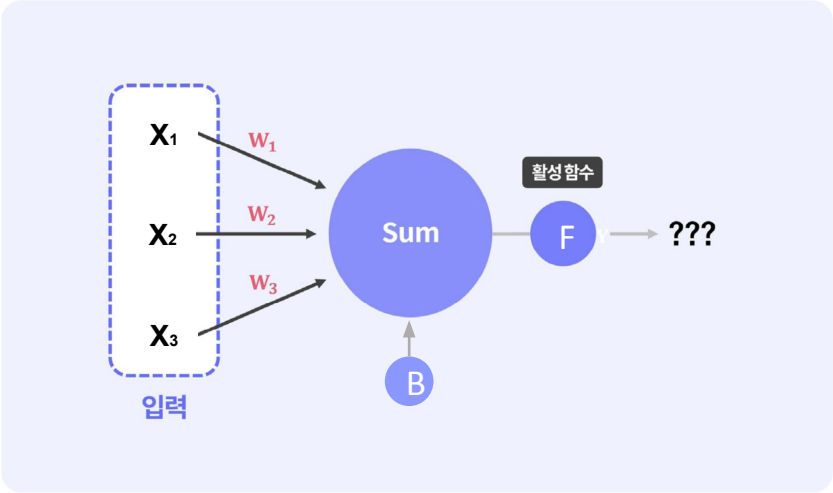


인공 신경망에서 사용하는 활성 함수는 모두 비선형 함수

즉, 활성 함수를 거친다는 것은 비선형성을 준다는 의미

3 활성 함수 (Activation Function)

Data Diving



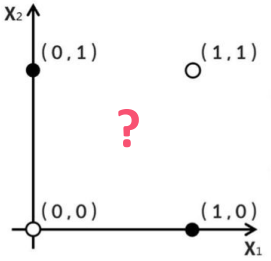
입력 값에 가중치를 곱한 것들을 다 더하고(가중합),

가중합에 편향을 더하여 활성 함수를 거쳐 결과 값을 도출

필요하다면

"분류해보시오!"

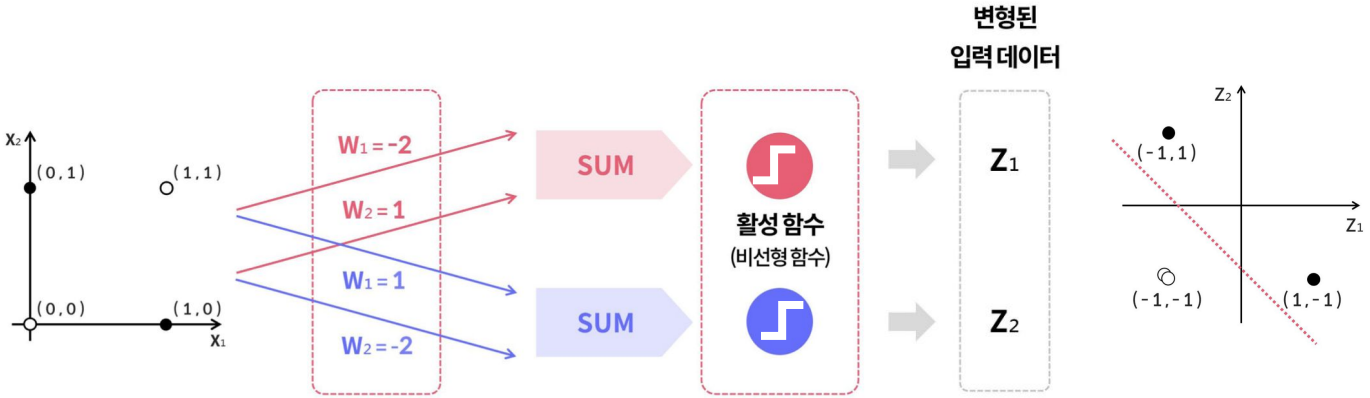
X_1	X_2	타겟 데이터
0	0	흰색
1	0	검은색
0	1	검은색
1	1	흰색



하나의 직선으로 검은 색 점과 흰 색 점을 분류할 수 없음

3 활성 함수 (Activation Function)

Data Diving



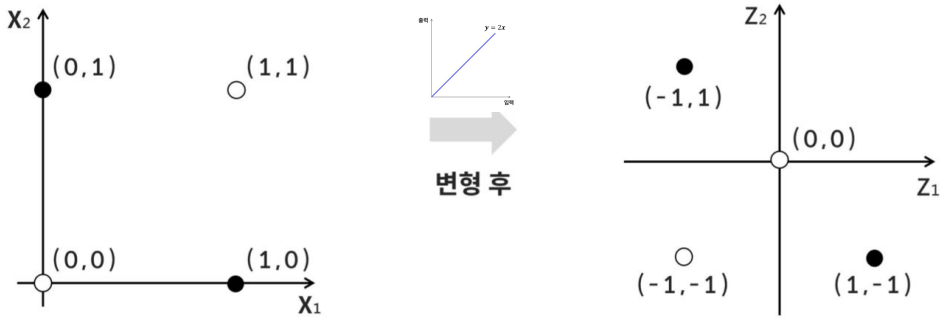
- weight는 학습을 통해 미리 구했다고 가정
- 활성 함수는 Step 함수를 사용했다고 가정

비선형성을 적용하여 점들을 새롭게 변형하는 것이 활성 함수의 목적

전통적인 머신러닝에서 사람이 직접하던 특징 추출 과정을, 딥러닝에서 활성 함수가 달린 퍼셉트론이 대신 해주는 것!

생각해보기

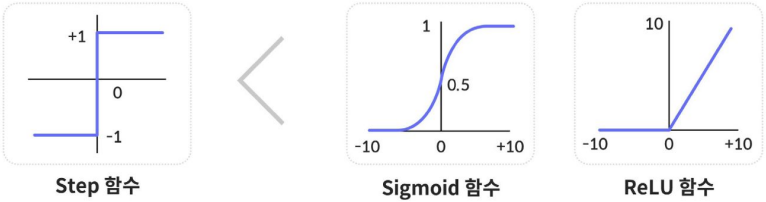
만약 비선형 함수를 쓰지 않았다면?



동일한 weight로 계산을 한 후 계단 함수 대신 일차 함수를 적용하면 4개의 점이 아래와 같이 변형이 됩니다. 점들이 이동하긴 했지만 계단 함수(비선형 함수)를 적용했을 때와 다르게 여전히 선 하나로 구분할 수 없는 상황이죠. 그리고 **선형 함수**를 적용했기 때문에 **전체적인 배치가 유사**하다는 것을 확인할 수 있습니다. 선형 함수를 적용하면 아무리 다른 일차 함수(y = 3x, y = x+2 등등...)을 적용한다고 하더라도 절대 선 하나로 4개의 점이 구분되지 않습니다. 따라서 **비선형성을 적용하여 점들을 새롭게 변형시키는 것이 활성 함수의 목적**입니다.

더 알아보기

자주 쓰이는 활성 함수?



Step 함수 Sigmoid 함수 ReLU 함수

계단 함수를 포함하여 앞에서 보았던 활성 함수 3가지에 대한 그래프입니다. 이 그래프에서 가로축은 활성 함수에 들어가는 입력 값 세로축은 그 때의 출력 값을 의미합니다. 그렇다면 계단 함수와 오른쪽에 있는 두 활성 함수의 가장 큰 차이점은 무엇일까요?

그것은 바로 **출력값의 형태**입니다. 계단 함수는 출력 값이 0 또는 1입니다. 그 중간에 대한 값은 전혀 없죠. 그 반면 sigmoid 함수와 ReLU 함수는 출력 값으로 0과 1뿐만 아니라 더 다양하고 연속적인 값을 가질 수 있습니다.

따라서 입력 데이터를 퍼셉트론에 통과시킨다고 할 때 어떤 활성 함수를 사용하느냐에 따라 출력의 형태가 다르게 됩니다. 비선형 함수로써의 기능은 동일하더라도 **계단 함수**를 선택하면 **불연속적인 형태로 변형**이 되고 **sigmoid나 ReLU 함수**를 선택하면 **부드럽고 연속적인 형태로 변형**이 됩니다. 그래서 보통 활성함수로 simgoid나 ReLU같은 함수를 많이 사용합니다.

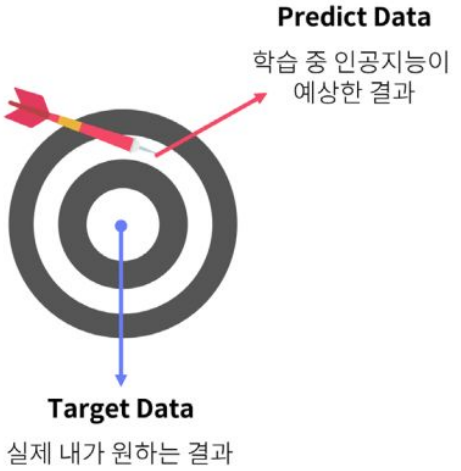
4 학습 : Target / predict

사용자가 설정한 실제 타겟 데이터 : (6, 7, 8.5) [단위 : 억] → **Target**

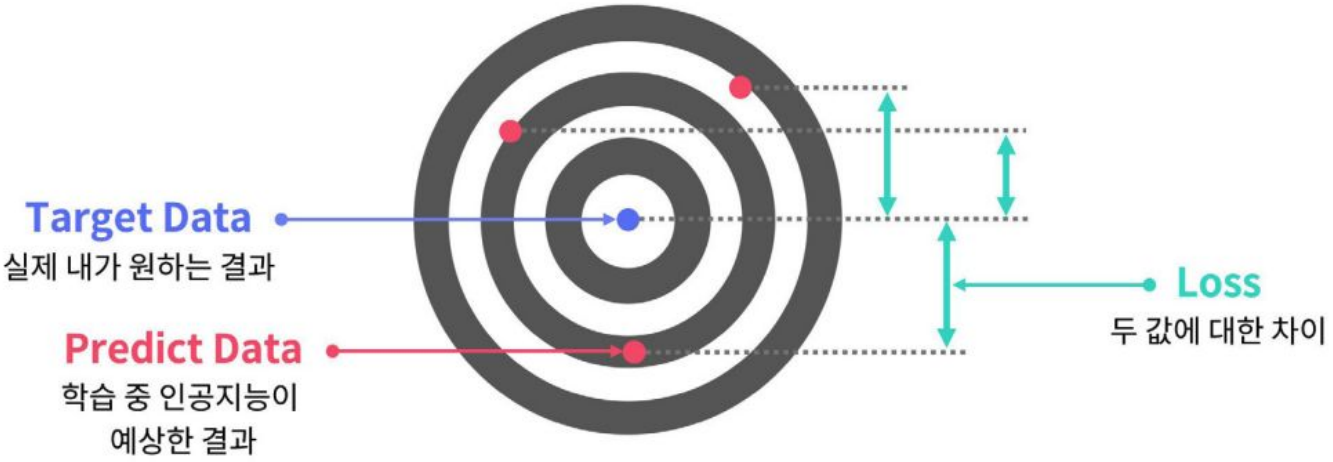
Weight를 이용해 머신러닝이 계산한 예측값 : (5.3, 6.2, 6.8) [단위 : 억] → **Predict**

- 고정된 Target, 변하는 Predict

학습 중인 인공지능



5 - 1 학습 : Loss



Loss = Target과 Predict의 차이

Loss가 줄어든다 = Target과 Predict가 가까워진다

Loss = Target과 Predict의 차이

입력 데이터 × weight = 예측값

weight = [평수 w, 방 개수 w, 지하철 역과의 거리 w] = [0.1 , 0.3 , 4]

	평수 (평)	방 개수 (개)	지하철 역과의 거리 (km)	집 값 억 원)
1)	32	3	0.3	6

→ 32 × 0.1 + 3 × 0.3 + 0.3 × 4 = 5.3(억)

2)	33	3	0.5	7

→ [33] × 0.1 + [3] × 0.3 + [0.5] × 4 = [6.2](억)

3)	36	4	0.5	8.5

→ [36 x 0.1 + 4 x 0.3 + 0.5 x 4 = 6.8 (억)]

인공지능(머신러닝)의 목표 : 최소의 Loss 찾기

Why?

머신러닝으로 출력한 predict(예측값)가 target(실제값)에 가장 가까워지면,
실제와 가장 흡사한 값을 예측하는 AI를 만들었다는 뜻이기 때문

생각해보기

생각해보기

weight와 bias가 각각 다음과 같을 때 그 때의 loss를 계산해봅시다.

1. weight = [0.1, 0.3, 3], bias = 0.5
2. weight = [0.1, 0.3, 4], bias = 0.5
3. weight = [0.1, 0.2, 2], bias = 0.5

평수 (평)	방 개수 (개)	지하철 역과의 거리 (km)	집 값 (억 원)
32	3	0.3	6
33	3	0.5	7
36	4	0.5	8.5

평수 (평)	방 개수 (개)	지하철 역과의 거리 (km)	집 값 (억 원)
32	3	0.3	6
33	3	0.5	7
36	4	0.5	8.5

1. weight = [0.1, 0.3, 3], bias = 0.5

	예측 집 값 (predict)	실제 집 값 (target)	|target-predict|
1)	$32 \times 0.1 + 3 \times 0.3 + 0.3 \times 3 + 0.5 = 5.5$	6	0.5
2)	$33 \times 0.1 + 3 \times 0.3 + 0.5 \times 3 + 0.5 = 6.2$	7	0.8
3)	$36 \times 0.1 + 4 \times 0.3 + 0.5 \times 3 + 0.5 = 6.8$	8.5	1.7

1), 2), 3)의 |target-predict| 총합 = 3

총합의 평균 = Loss = 3 ÷ 3 = 1

평수 (평)	방 개수 (개)	지하철 역과의 거리 (km)	집 값 (억 원)
32	3	0.3	6
33	3	0.5	7
36	4	0.5	8.5

2. weight = [0.1, 0.3, 4], bias = 0.5

	예측 집 값 (predict)	실제 집 값 (target)	|target-predict|
1)	32 × 0.1 + 3 × 0.3 + 0.3 × 4 + [ ] = [ ]	6	[ ]
2)	33 × 0.1 + 3 × 0.3 + 0.5 × 4 + [ ] = [ ]	7	[ ]
3)	36 × 0.1 + 4 × 0.3 + 0.5 × 4 + [ ] = [ ]	8.5	[ ]

1), 2), 3)의 |target-predict| 총합 = [ ] **총합의 평균** = Loss = [ ]

평수 (평)	방 개수 (개)	지하철 역과의 거리 (km)	집 값 (억 원)
32	3	0.3	6
33	3	0.5	7
36	4	0.5	8.5

2. weight = [0.1, 0.3, 4], bias = 0.5

	예측 집 값 (predict)	실제 집 값 (target)	|target-predict|
1)	32 × 0.1 + 3 × 0.3 + 0.3 × 4 + [0.5] = [5.8]	6	[0.2]
2)	33 × 0.1 + 3 × 0.3 + 0.5 × 4 + [0.5] = [6.7]	7	[0.3]
3)	36 × 0.1 + 4 × 0.3 + 0.5 × 4 + [0.5] = [7.3]	8.5	[1.2]

1), 2), 3)의 |target-predict| 총합 = [1.7] **총합의 평균** = Loss = [1.7÷3=0.567]

평수 (평)	방 개수 (개)	지하철 역과의 거리 (km)	집 값 (억 원)
32	3	0.3	6
33	3	0.5	7
36	4	0.5	8.5

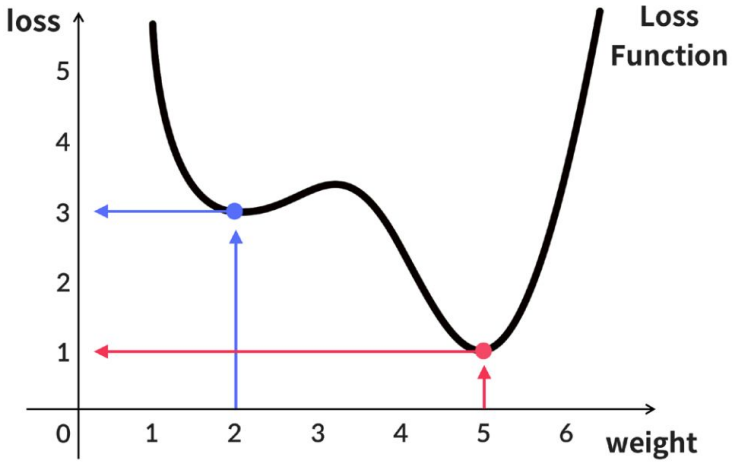
3. weight = [0.1, 0.2, 2], bias = 0.5

	예측 집 값 (predict)	실제 집 값 (target)	|target-predict|
1)	[32x0.1+3x0.2+0.3x2+0.5=4.9]	6	[1.1]
2)	[33x0.1+3x0.2+0.5x2+0.5=5.4]	7	[1.6]
3)	[36x0.1+4x0.2+0.5x2+0.5=5.9]	8.5	[2.6]

1),2),3)의 |target-predict| 총합 = [5.3] **총합의 평균** = Loss = [5.3÷3=1.767]

5 - 2 학습 : Loss Function

모든 weight에 대한 loss를 모두 모아놓은 함수 = Loss Function



신경망이 얼마나 못하는지를 측정하는 지표

- 손실함수의 값이 작을수록 학습이 잘 되었음을 의미
- 손실함수의 값을 관찰함으로써 신경망이 학습이 되고 있는지 관찰 가능

$$\mathrm{MSE} = \frac{SSE}{n-2} = \frac{1}{n-2}\sum(\hat{y} - y)^2$$

회귀의 경우,
평균제곱오차 (mean square error; MSE) 사용

$$E = -\sum_i y \log \tilde{y} + (1-y)\log(1-\tilde{y})$$

분류의 경우,
교차엔트로피오차 (cross entropy error) 사용

신경망의 기능에 따라 손실함수가 정해짐

```
[ ]  import torch.optim as optim

     criterion = nn.CrossEntropyLoss()

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

함수가 내장되어 있어 손실함수의 식은 몰라도 돼요!

다만, 다음과 같이 쓸거니까 이름은
기억해주세요!

생각해보기

머신러닝의 학습 과정에서 나온 용어들을
정리해보겠습니다.
< 보기 >에서 알맞은 단어를 찾아 빈칸을 채워주세요

< 보기 >

Target / Weight / Loss / Predict / 고정된 / 변하는 / 줄이는

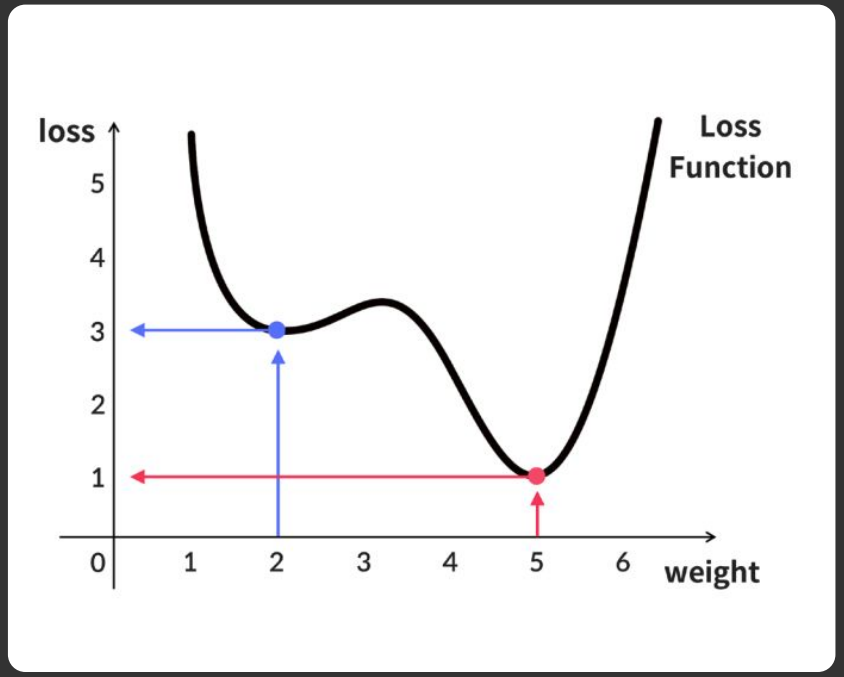
1. 머신러닝에서 최적의 ( )을/를 찾는 것을 학습이라고 합니다.

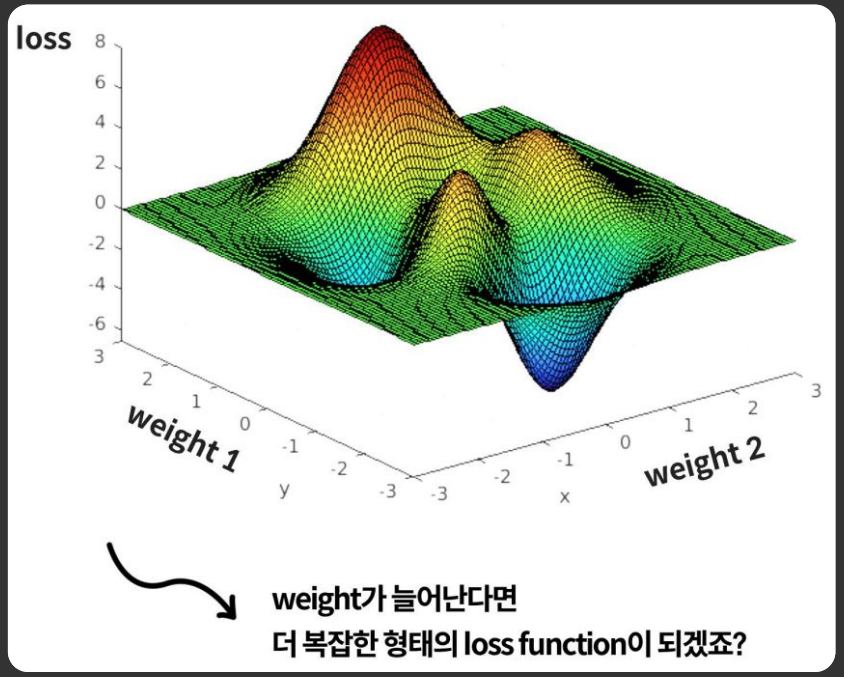
2. Weight를 통해 계산한 값은 ( )(이)라고 하고 이는 ( ) 값입니다.

3. 기계가 정답으로 인식하고 있는 목푯값에 해당하는 것은 ( )(이)라고 하며 이는 ( )값입니다.

4. Predict와 Target의 차이를 ( )(이)라고 하고 이를 ( ) 것이 머신러닝의 목표가 됩니다.

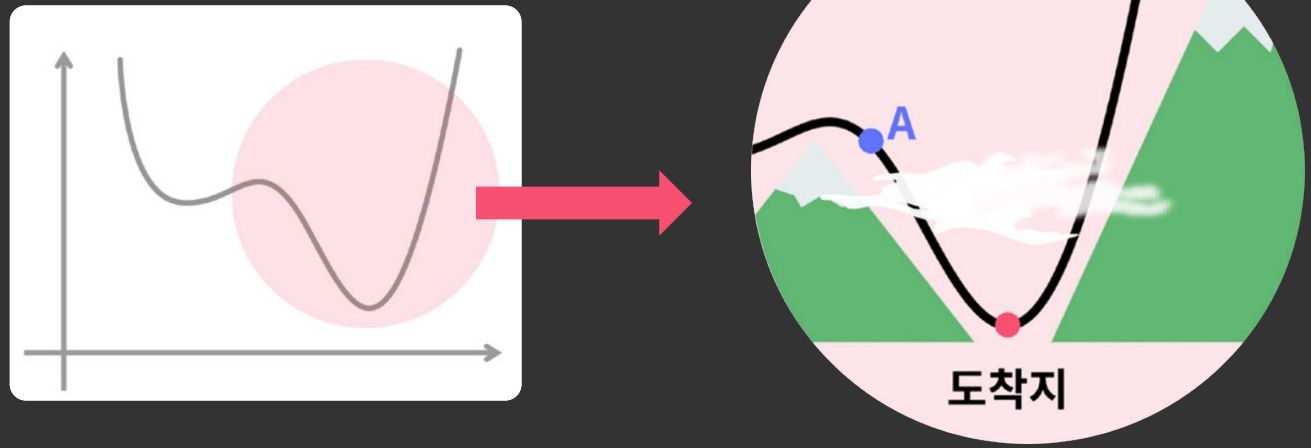
6 학습 : Optimization





6 학습 : Optimization

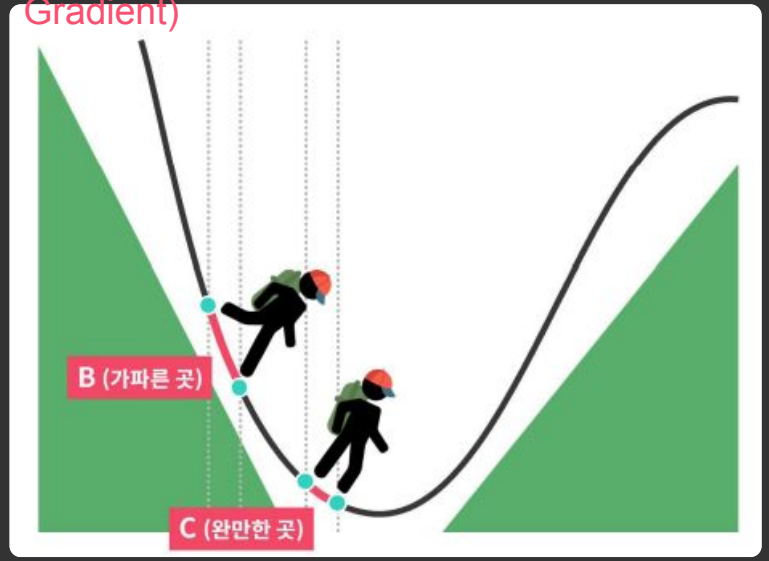
- 경사하강법 (Gradient Descent)



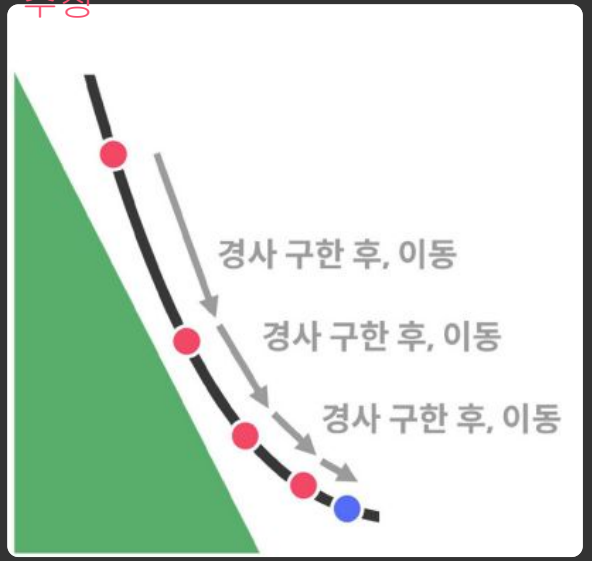
loss function 그래프의 모양을 몰라도,
더듬더듬 낮은 쪽을 찾아 한걸음씩 그 방향으로 내려가는 것이 경사하강법의 기본 원리

6 학습 : Optimization

- 경사도 (기울기 · Gradient)



- 한 단계씩 weight 수정

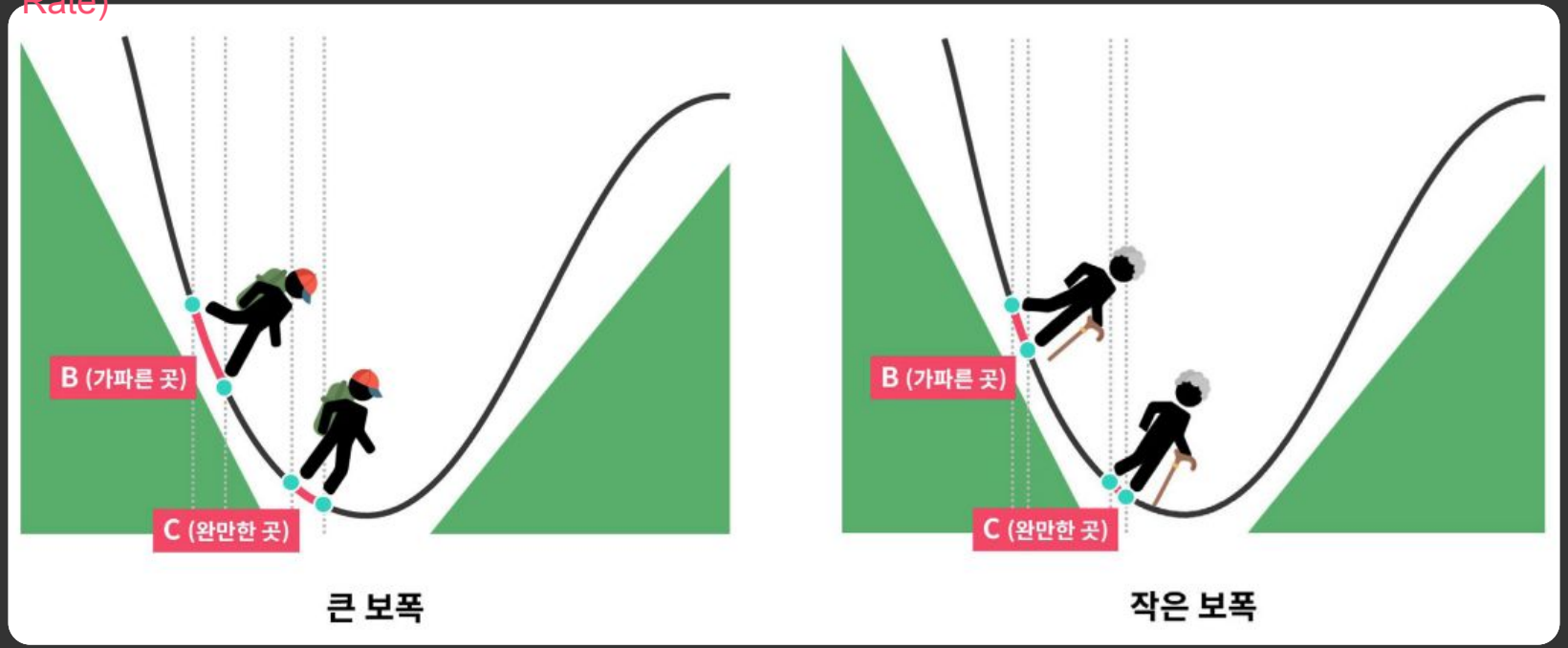


loss function에서 경사도는 기울기에 해당

6 학습 : Optimization

- 보폭 (학습률 · Learning Rate)

https://developers.google.com/machine-learning/crash-course/fitter/graph?hl=ko

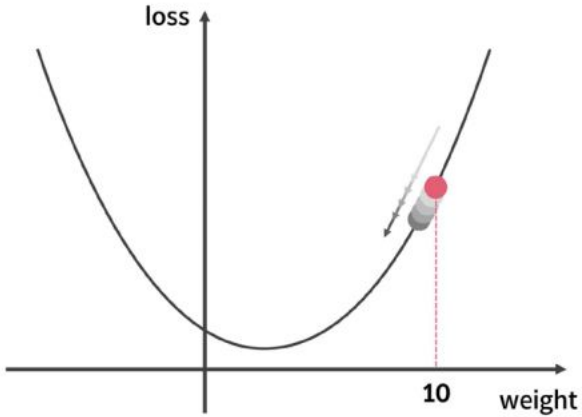


6 학습 : Optimization

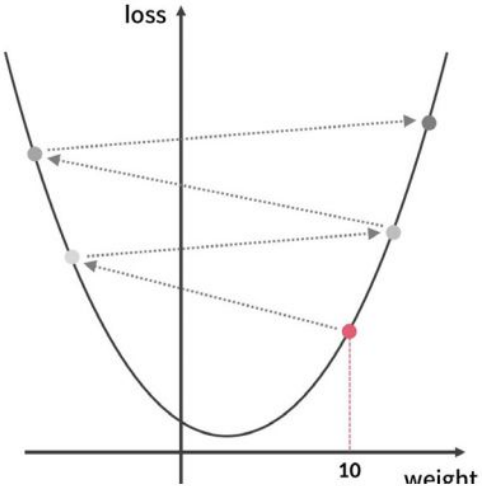
- 보폭 (학습률 · Learning Rate)

https://developers.google.com/machine-learning/crash-course/fitter/graph?hl=ko

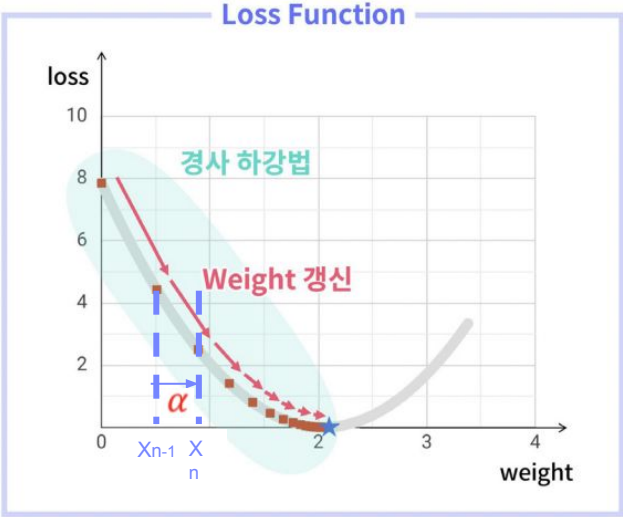
- 학습률이 너무 작을 때



- 학습률이 너무 클 때



6 학습 : Optimization



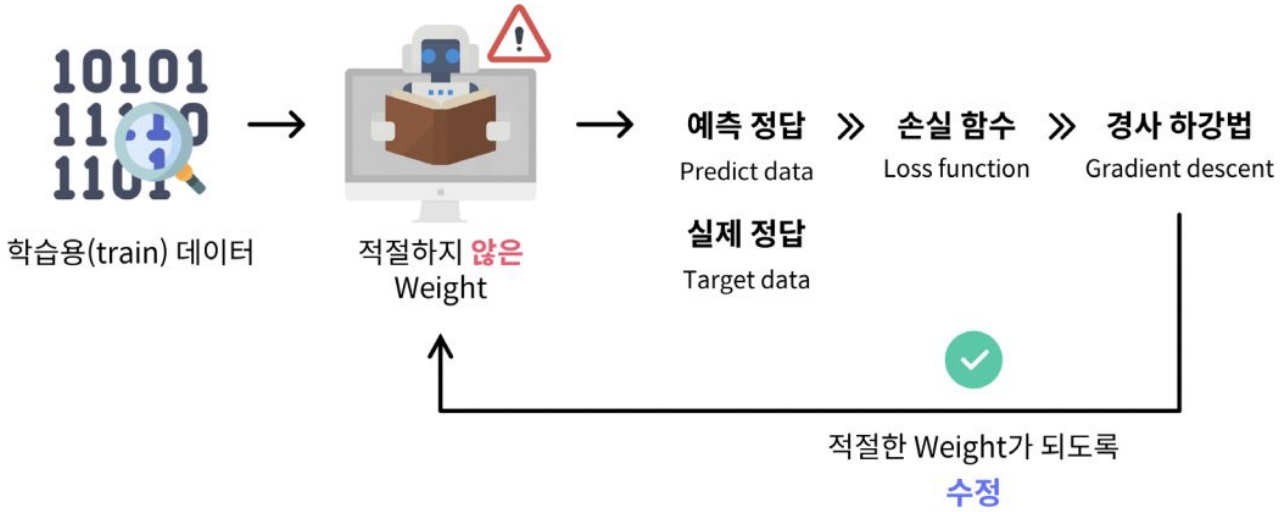
그 다음 weight = 현재 weight - 학습률 × 기울기

$$x_n = x_{n-1} - \alpha \frac{df(x_{n-1})}{dx}$$

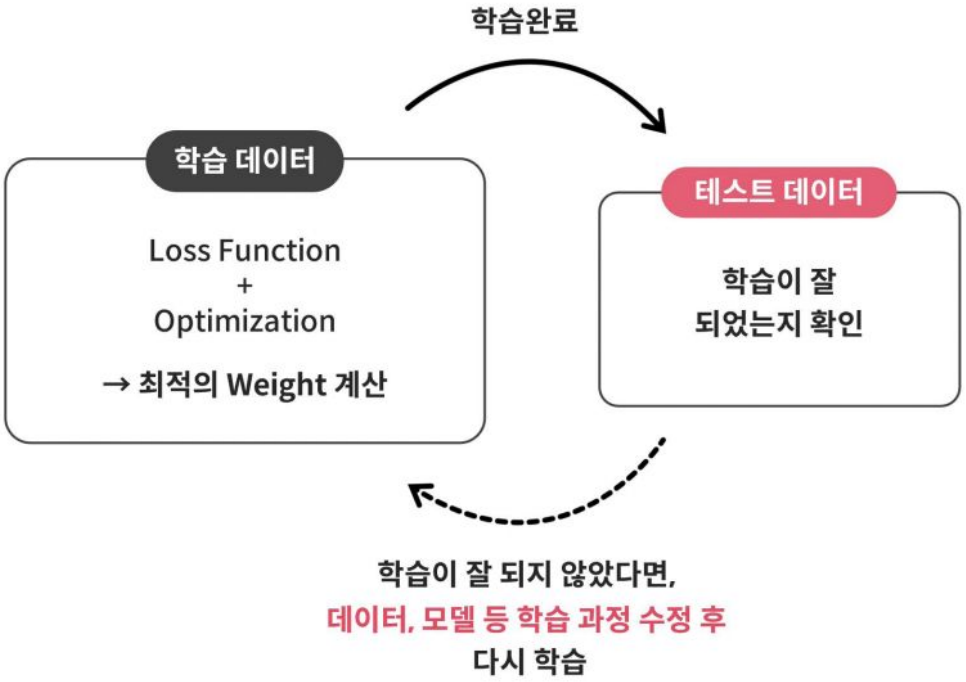
- α: 학습률(learning rate)
- 주로 $\alpha = 0.001$ 또는 $\alpha = 0.0001$ 사용

- 손실함수의 값이 감소하도록 신경망의 **가중치 (weight)**와 **편차 (bias)**를 업데이트 시키는 알고리즘
- 경사하강법 (gradient descent) 기반 기법이 주로 사용

6 테스트 : Test


학습용(train) 데이터
적절하지 않은
Weight
예측 정답
Predict data
손실 함수
Loss function
경사 하강법
Gradient descent
실제 정답
Target data
적절한 Weight가 되도록
수정

6 테스트 : Test


학습완료
학습 데이터
Loss Function
+
Optimization
→ 최적의 Weight 계산
테스트 데이터
학습이 잘
되었는지 확인
학습이 잘 되지 않았다면,
데이터, 모델 등 학습 과정 수정 후
다시 학습

python으로 하는 딥러닝 - 파이토치

Data Diving

- 파이썬으로 딥러닝을 하는 방식 - tensorflow vs pytorch
- pytorch가 해 주는 것
  - 일반적인 뉴럴 넷에서 쓰이는 것들을 미리 만들어둠 (맨 처음 행렬부터 다 만들 필요 x)
  - 정해진 방식으로 학습하기 (gradient descent)
- 우리가 해야 하는 것
  - 행렬을 어떻게 쌓을지 잘 정하기 (딥러닝 모델 설계)
  - 어떤 식으로 학습할지 정하기 (여러 optimizer 중에서)
  - 기타 옵션들 정하기 (dropout을 얼마나 할지? 파라미터 사이즈는 어떻게 할지?)