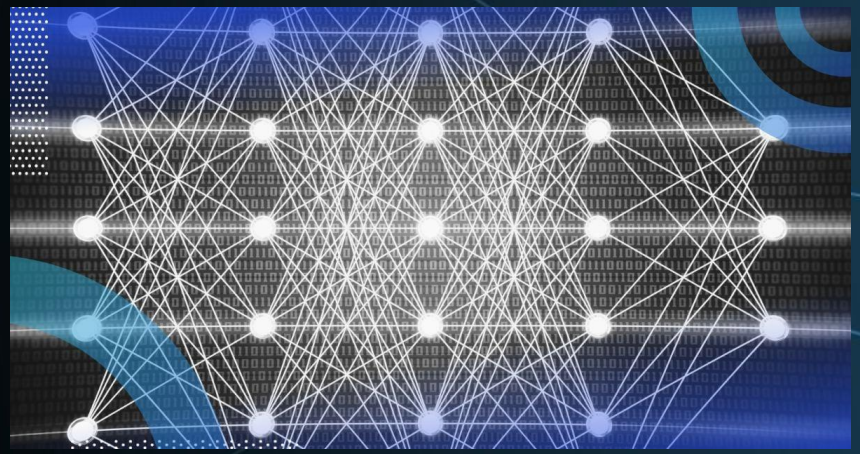
Neural Network



Neural Network

인공지능(AI)
컴퓨터에서 음성 및 작성된 언어를 보고 이해하고 번역하고
데이터를 분석하고 추천하는 기능을 포함하여
다양한 고급 기능을 수행할 수 있는 일련의 기술

기계학습
데이터를 제공하여 명시적으로 프로그래밍하지 않고
신경망과 딥 러닝을 사용하여 시스템이 자율적으로
학습하고 개선하는 과정

딥 러닝
여러 '비선형 변환기법'의 조합을 통해 높은 수준의 추상화를
시도하는 기계 학습 알고리즘의 집합
인경신경망 + 여러 층의 히든레이어

Neural Network

전통적인 ML 알고리즘 *vs* 딥러닝

- **딥러닝**
  > 신경망 (neural networks) based
  > 데이터에서 자동으로 복잡한 특징을 학습
  > **복잡한 데이터 구조**와 패턴 학습 가능. **비선형성**
  > 일반적으로 성능이 더 좋다고 알려져 있음
    - 데이터의 크기와 특성 등에 따라 다름
    - 보통 비정형 데이터에 대한 성능 우수

- **전통적인 ML 알고리즘**
  > 상대적으로 간단한 데이터 구조에서 잘 작동하며, 데이터의 **선형성**, 분포 등에 대한 가정을 기반
  > 주로 **구조화된 데이터**에 적용
  > 특징 추출(feature extraction) 과정에서 도메인 지식이 중요.
  > 모델의 성능은 선택된 특징의 질에 크게 의존

NN_2

Neural Network

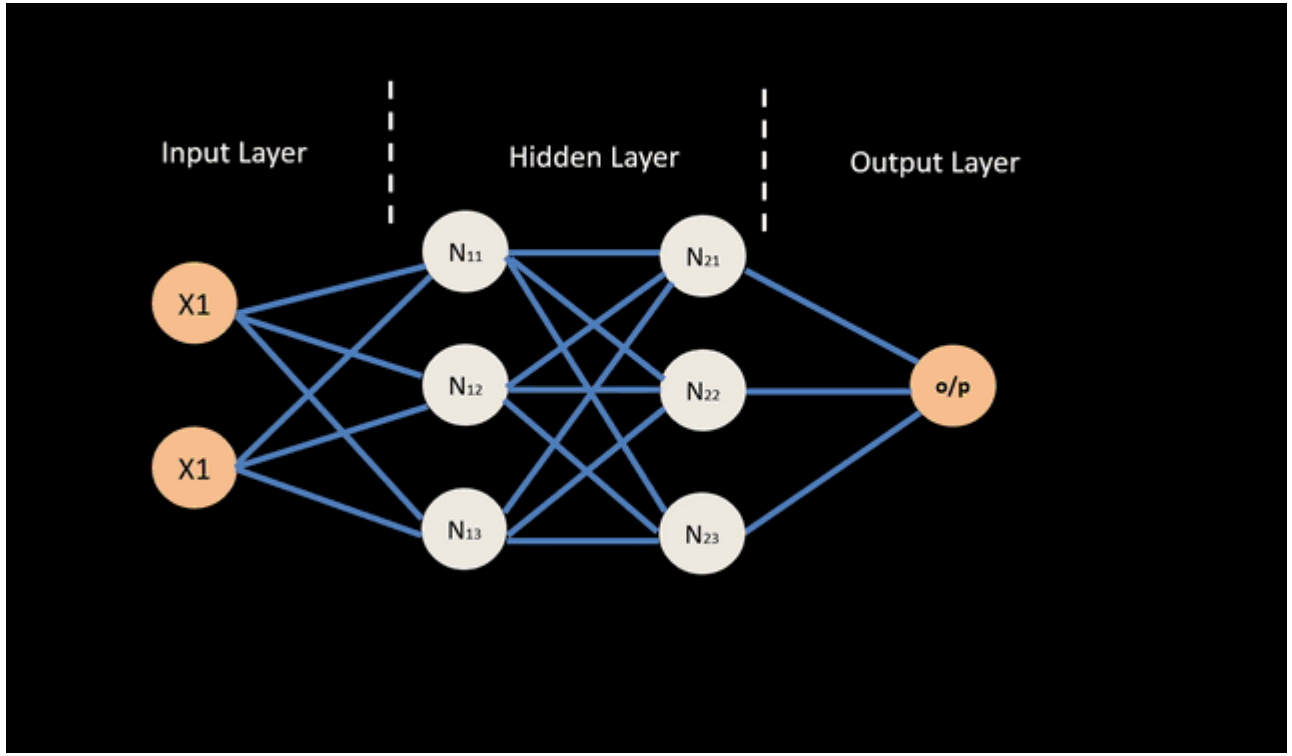
인공 신경망학습 프로세스

순전파(활성화함수) → 오차 계산(비용함수) → 오류 역전파 → 가중치 업데이트

(과적합 방지 -> 정규화(Regularization))

Neural Network

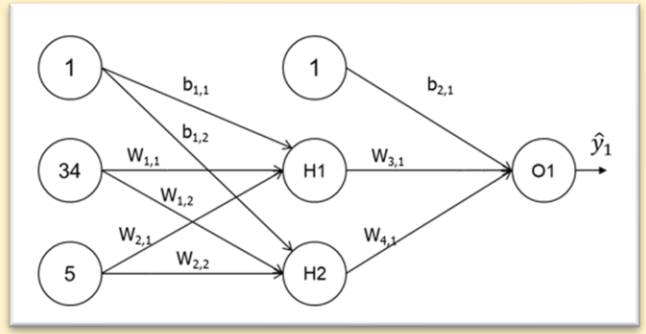
인공 신경망학습 프로세스



https://seahahn.tistory.com/98

Neural Network

인공 신경망학습 프로세스



1. 초기화(Initialization)

> 학습 과정을 시작하기 위한 준비 단계 . (ex,)무작위 값으로 초기화.

2. 순전파(Forward Propagation)

> 입력 **데이터**가 신경망의 입력층으로 주어짐, 각 층의 뉴런을 통과하면서 **활성화 함수**에 의해 처리

> 신경망의 출력층까지 과정 수행

*** 활성화 함수** : 신경망에 **비선형성**을 도입하여 복잡한 문제 해결 수행

3. 오차 계산(Error Calculation)

> 신경망의 출력과 실제 타깃 값 사이의 차이 계산(**오차 함수** - 평균 제곱 오차(MSE)나 교차 엔트로피(Cross-Entropy) 사용)

> 신경망의 예측이 얼마나 정확한지를 수치적으로 표현

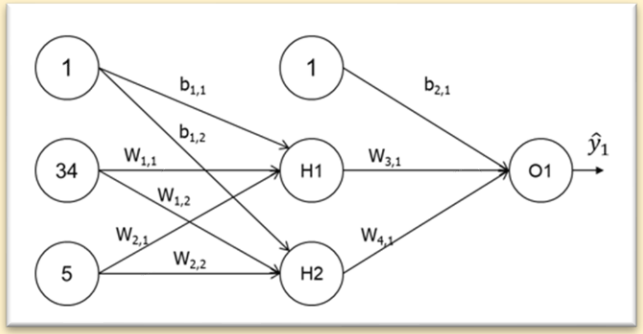
4. 오류 역전파(Error **Backpropagation**)

> 계산된 오차는 출력층에서부터 입력층 방향으로 역전파 되어 각 가중치가 오차에 얼마나 기여하는지 계산

각 가중치가 오차에 미치는 영향 계산 -> 가중치 조정

Neural Network

인공 신경망학습 프로세스



5. 가중치 업데이트(Weight Update)

> **경사 하강법**을 사용하여 각 가중치를 업데이트.

> 오차를 최소화하는 방향으로 가중치를 조정하여 신경망의 성능 개선

6. 과적합(Overfitting) 및 정규화(Regularization)

> 학습 과정 중 **과적합** 발생 가능(학습 데이터에 대해 너무 잘 맞추어져 새로운 데이터에 대한 일반화 성능이 떨어지는 현상)

> **정규화** 기법(L1, L2 규제, 드롭아웃 등)을 통해 모델의 복잡도 제어, 과적합 줄임

7. 반복 학습(Iterative Learning)

> 위의 과정**(순전파 → 오차 계산 → 오류 역전파 → 가중치 업데이트)** 계속 반복, 최적화 진행

위 학습 단계를 반복 -> 입력 데이터에 내재된 복잡한 패턴과 구조를 학습,

-> 다양한 문제에 대한 예측(계산)을 수행

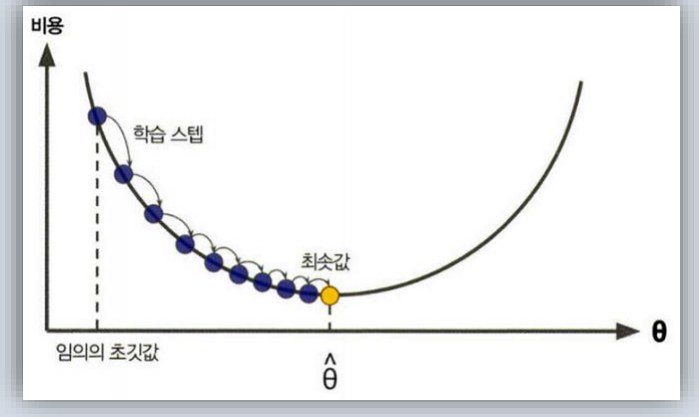
Neural Network

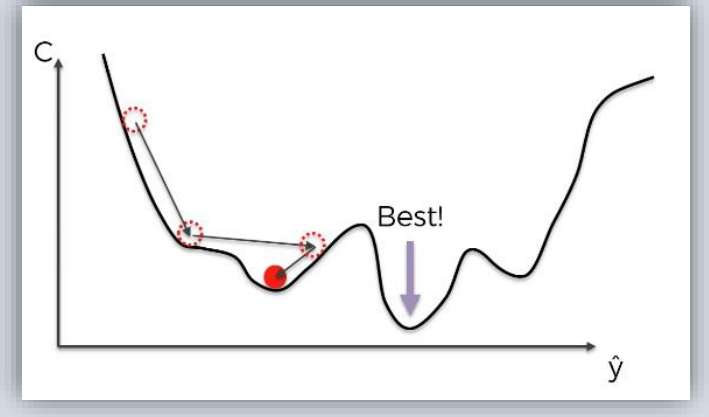
인공 신경망학습 프로세스

1. 초기화(Initialization) - Weight

> 학습 과정을 시작하기 위한 준비 단계 . (ex,)무작위 값 초기화

적절한 초기화를 통해 신경망이 빠르게 수렴, **경사 하강법**이 효율적으로 작동하도록 마련
If, 부적절한 초기화일 경우
- 학습 속도가 늦어 짐
- 신경망 학습 방해 (gradient vanishing/exploding),
- **<u>지역 최솟값</u>**(local minima) 문제 발생






```
print("Weight of first hidden layer:", model.fc1.weight.
    [-0.0058, -0.2836, -0.0217,  0.2652],
    [ 0.3448,  0.0891,  0.1539, -0.1527],
    [ 0.2777,  0.4313, -0.0363,  0.3461],
    [-0.2031, -0.2037,  0.1142, -0.3359],
    [ 0.4055, -0.4603,  0.3420, -0.4732],
    [-0.0758, -0.3533,  0.1142,  0.0566],
    [-0.3811,  0.0252, -0.0242, -0.4098],
    [ 0.2326,  0.1702, -0.3825,  0.3353],
    [-0.3802, -0.4732,  0.0997, -0.2166],
    [-0.2036,  0.2465, -0.1772, -0.0342],
    [ 0.0769, -0.3079, -0.3603,  0.3320],
    [-0.1640,  0.0087,  0.1098,  0.0172],
    [-0.0333,  0.1915,  0.4446,  0.2531],
    [ 0.3915,  0.0048, -0.3163, -0.1294],
    [ 0.0320,  0.4789,  0.4915,  0.0685],
    [-0.1768, -0.1448,  0.3675,  0.0896],
    [ 0.4665,  0.4394, -0.4358, -0.4374],
    [ 0.0072, -0.2807, -0.4197, -0.1590],
    [ 0.4765,  0.4422, -0.4353,  0.1949]
```

Neural Network

인공 신경망학습 프로세스

1. 초기화(Initialization) - Weight

> 학습 과정을 시작하기 위한 준비 단계 . (ex,)무작위 값 초기화

- **<u>효율적 가중치 초기화의 목적</u>**

1. Breaking Symmetry(대칭) :
if, 모든 가중치가 동일 값으로 초기화 -> 모든 뉴런이 동일하게 학습되어 대칭적인 가중치로 세팅

2. 기울기 소실/폭발 문제(Vanishing/Exploding Gradient):
가중치를 너무 크게 또는 너무 작게 초기화하면, 기울기가 소실되거나 폭발하는 문제가 발생

Neural Network

인공 신경망학습 프로세스

1. 초기화(Initialization) - Weight

> 학습 과정을 시작하기 위한 준비 단계 . (ex,)무작위 값 초기화

- **초기화 방법**

1. 무작위 초기화(Random Initialization):
  - 가중치를 **작은 난수**로 설정하여 각 뉴런이 다른 값을 가지도록
  - ex, 작은 값에서 균등 분포(uniform distribution)나 정규 분포(normal distribution)로 무작위 초기화

2. Xavier 초기화(Glorot Initialization):
  - 선형 및 비선형 활성화 함수(ReLU, Sigmoid 등)에 사용
  - 입력값과 출력값의 분산을 비슷하게 유지하여 기울기 소실 문제를 완화

3. He 초기화(He Initialization):
  - **ReLU** 계열의 활성화 함수에 적합한 초기화 방법
  - Xavier 초기화와 유사, 활성화 함수가 비선형인 경우(예: ReLU) 더 큰 분산을 사용.

4. 제로 초기화(Zero Initialization):
  - 가중치를 모두 0으로 초기화

Neural Network

인공 신경망학습 프로세스

1. 초기화(Initialization) - Weight

> 학습 과정을 시작하기 위한 준비 단계 . (ex,)무작위 값 초기화

```
# 모델 정의
model = tf.keras.Sequential([
    layers.Dense(64, activation='relu',
kernel_initializer=initializers.GlorotUniform(), bias_initializer='zeros'),
    layers.Dense(10, activation='softmax',
kernel_initializer=initializers.HeNormal())
])
```

Neural Network

인공 신경망학습 프로세스

```
 Layer 1: dense_6
Weight(shape: (4, 8)):
[[-0.5094315   0.67358464 -0.6508793  -0.49807882  0.6609774  -0.37423384
  -0.37270844 -0.70390886]
 [-0.6634566  -0.534152    0.02072477  0.33963436 -0.4090977  -0.09815413
  -0.3426215   0.481216  ]
 [ 0.6851732   0.0888629   0.20804745  0.3694237  -0.6215485  -0.00365698
   0.21699995 -0.12497181]
 [-0.2685174  -0.03484076 -0.53376067 -0.48523277 -0.07726377  0.40226692
   0.07783139  0.05404943]]
```

1. 초기화(Initialization) - Weight

> 학습 과정을 시작하기 위한 준비 단계 . (ex,)무작위 값 초기화

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

# 모델 정의
model = Sequential()
model.add(Dense(8, input_dim=4, activation='relu'))
model.add(Dense(3, activation='softmax'))

# 층별 가중치 확인
for i, layer in enumerate(model.layers):
    weights, biases = layer.get_weights()
    print(f"\n Layer {i + 1}: {layer.name}")
    print(f"Weight(shape: {weights.shape}):\n{weights}")
```

Neural Network

- **실습 – NN**

  4.03.NN_iris.ipynb

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	320
dense_1 (Dense)	(None, 3)	195

```
Total params: 515 (2.01 KB)
Trainable params: 515 (2.01 KB)
Non-trainable params: 0 (0.00 B)
```

Neural Network

- TASK

  4.03.NN.wine.p.ipynb

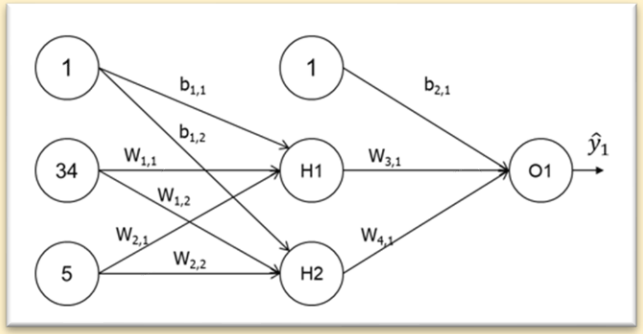
  - NN 모델링, 모델 구조 변경
  - 다른 ML 알고리즘으로 모델링, 성능 비교

품질예측, 회귀 모델

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6.0
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6.0
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6.0
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6.0
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6.0
5	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6.0
6	6.2	0.32	0.16	7.0	0.045	30.0	136.0	0.9949	3.18	0.47	9.6	6.0
7	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6.0
8	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6.0
9	8.1	0.22	0.43	1.5	0.044	28.0	129.0	0.9938	3.22	0.45	11.0	6.0

Neural Network

인공 신경망학습 프로세스



1. 초기화(Initialization)

> 학습 과정을 시작하기 위한 준비 단계 . (보통)무작위 값으로 초기화.

2. 순전파(Forward Propagation)

> 입력 **데이터**가 신경망의 입력층으로 주어짐, 각 층의 뉴런을 통과하면서 **활성화 함수**에 의해 처리

> 신경망의 출력층까지 과정 수행

*** 활성화 함수** : 신경망에 **비선형성**을 도입하여 복잡한 문제 해결 수행

3. 오차 계산(Error Calculation)

> 신경망의 출력과 실제 타깃 값 사이의 차이 계산(**오차 함수** - 평균 제곱 오차(MSE)나 교차 엔트로피(Cross-Entropy) 사용)

> 신경망의 예측이 얼마나 정확한지를 수치적으로 표현

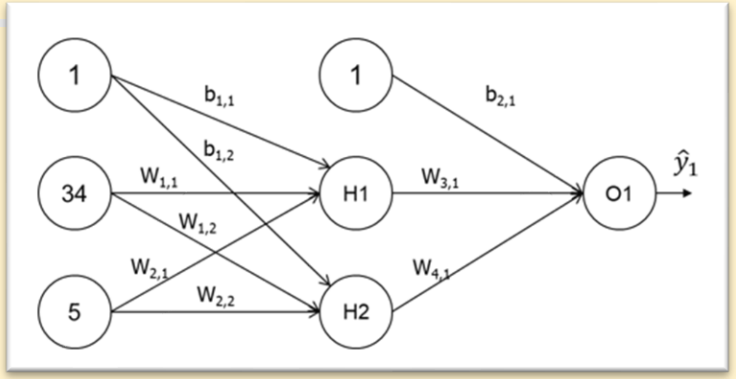
4. 오류 역전파(Error **Backpropagation**)

> 계산된 오차는 출력층에서부터 입력층 방향으로 역전파 되어 각 가중치가 오차에 얼마나 기여하는지 계산

각 가중치가 오차에 미치는 영향 계산 -> 가중치 조정

Neural Network

인공 신경망학습 프로세스



5. 가중치 업데이트(Weight Update)

> **경사 하강법**을 사용하여 각 가중치를 업데이트.

> 오차를 최소화하는 방향으로 가중치를 조정하여 신경망의 성능 개선

6. 과적합(Overfitting) 및 정규화(Regularization)

> 학습 과정 중 **과적합** 발생 가능(학습 데이터에 대해 너무 잘 맞추어져 새로운 데이터에 대한 일반화 성능이 떨어지는 현상)

> **정규화** 기법(L1, L2 규제, 드롭아웃 등)을 통해 모델의 복잡도 제어, 과적합 줄임

7. 반복 학습(Iterative Learning)

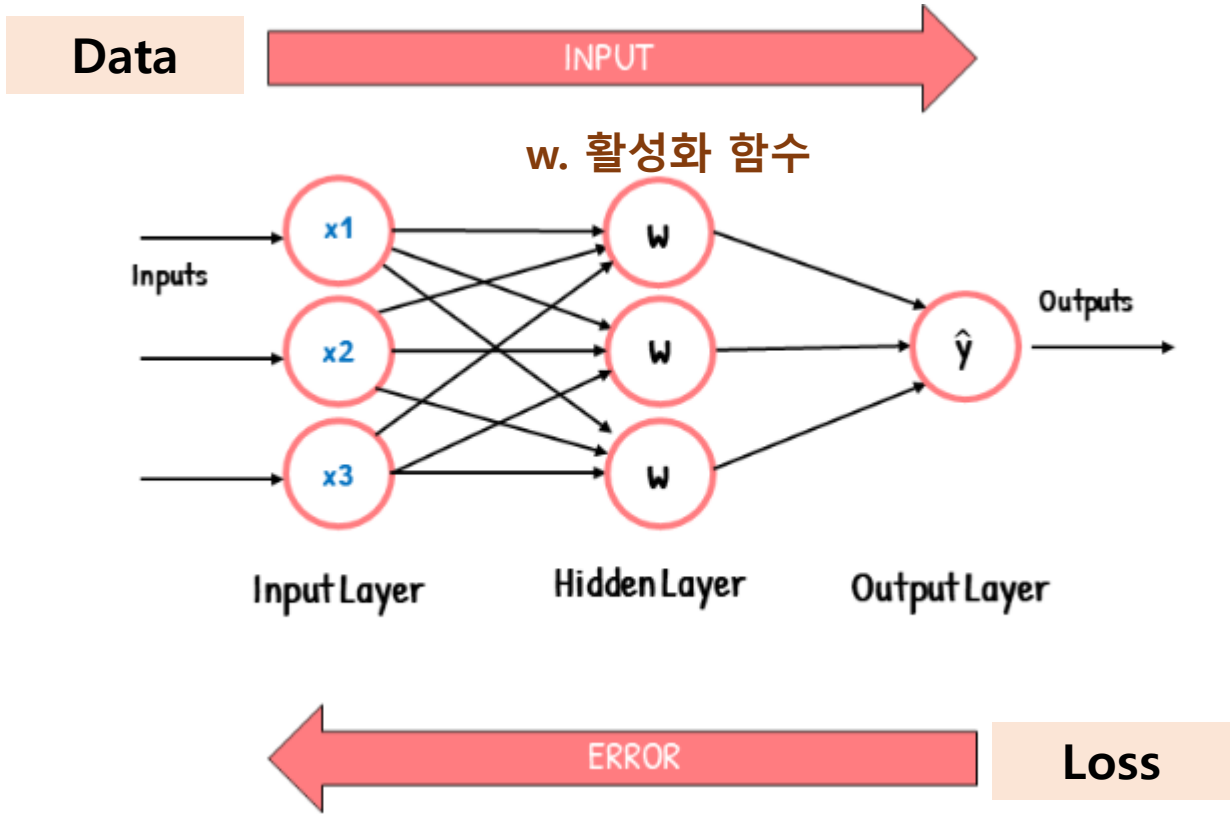
> 위의 과정**(순전파 → 오차 계산 → 오류 역전파 → 가중치 업데이트)** 계속 반복, 최적화 진행

위 학습 단계를 반복 -> 입력 데이터에 내재된 복잡한 패턴과 구조를 학습,
다양한 문제에 대한 예측(계산)을 수행

Neural Network

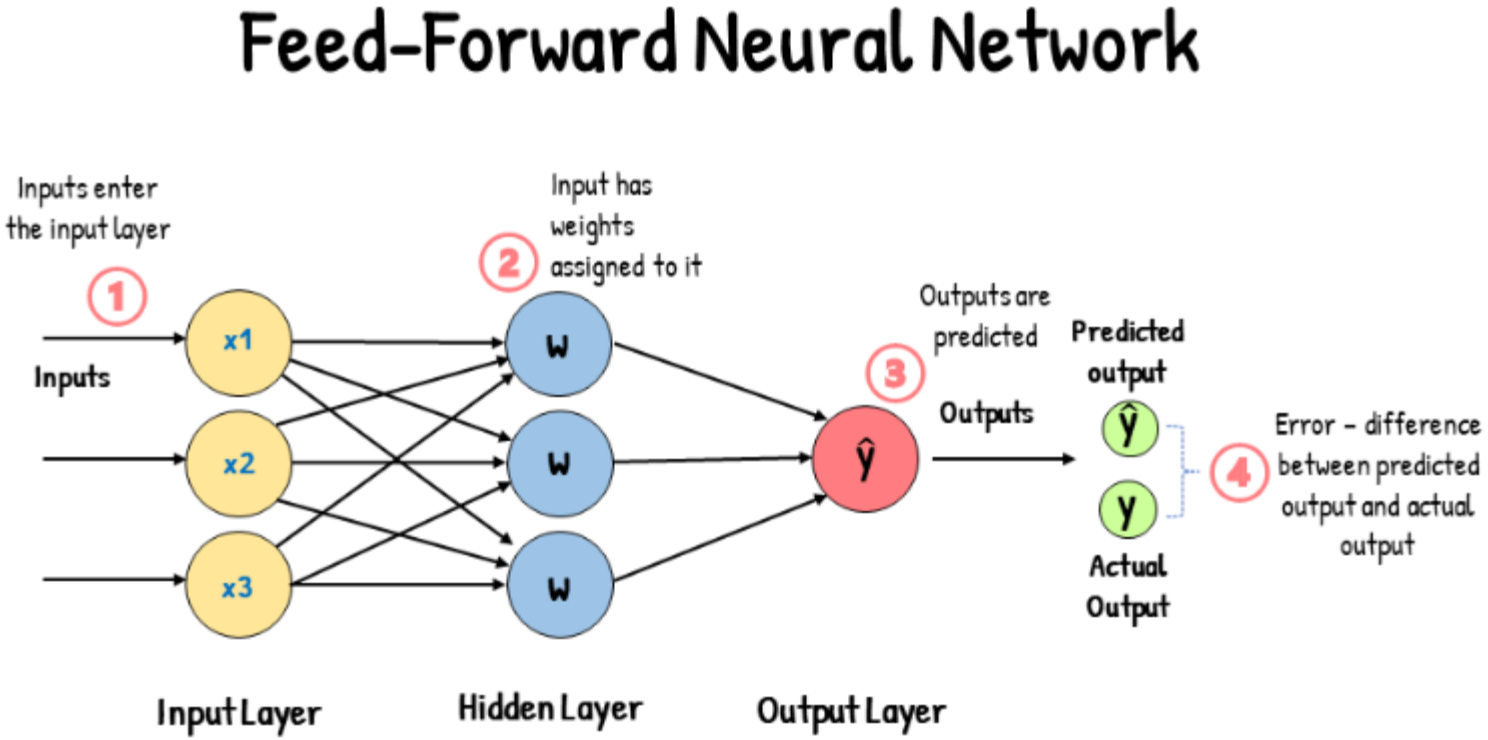
역전파 (Backpropagation)

순전파(활성화함수) → 오차 계산(손실함수) → 오류 역전파 → 가중치 업데이트



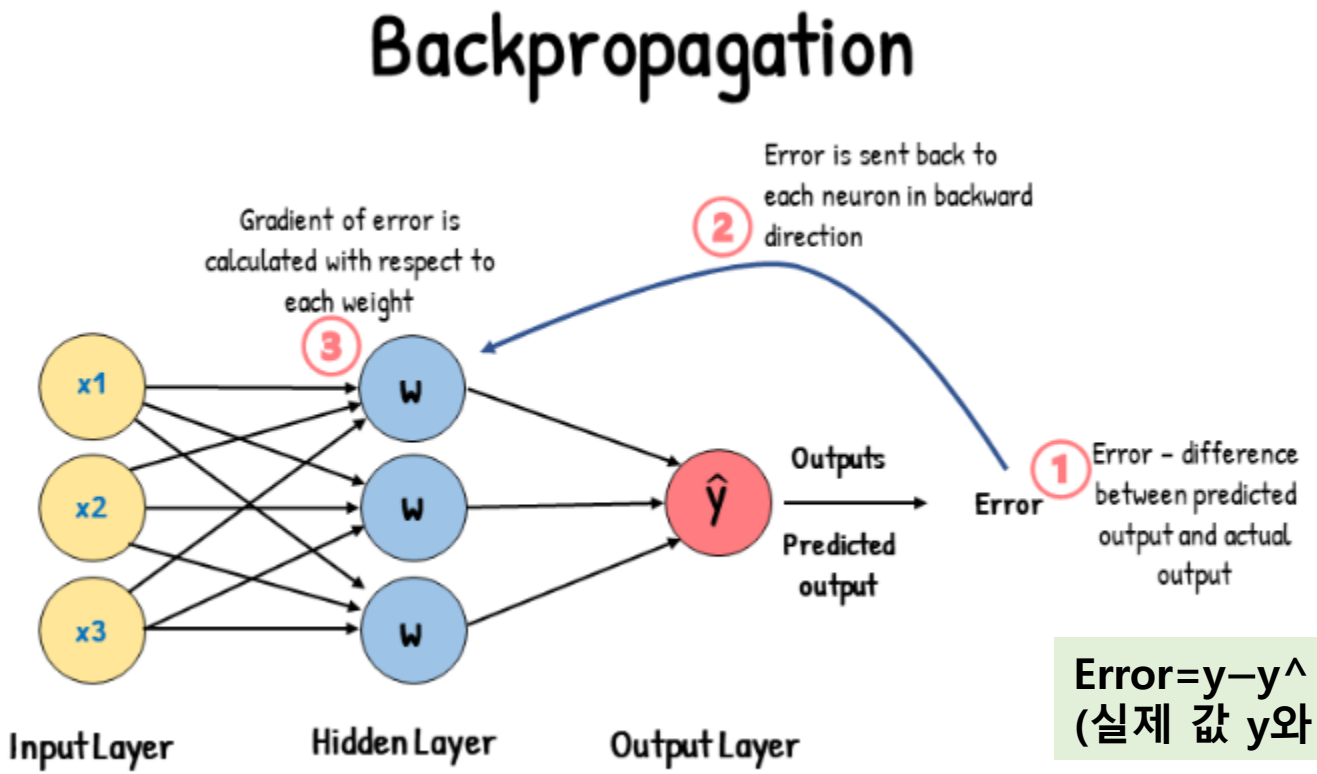
Neural Network

역전파 (Backpropagation)



Neural Network

역전파 (Backpropagation)



Error=y−y^
(실제 값 y와 예측 값 y^의 차이)

기울기 계산
-> 경사 하강법(Gradient Descent)을 통해 가중치 업데이트

Neural Network

역전파 (Backpropagation)

오류 역전파(Error Backpropagation)

: 네트워크를 통해 오류를 역방향으로 전파시켜 가중치를 조정하는 학습 알고리즘

역전파 과정

1) 순전파(Forward Propagation)

- 입력 데이터가 신경망의 각 층을 순차적으로 통과하면서, 각 뉴런의 출력값 계산
- 최종적으로 출력층에서 예측값 생성

2) 손실 계산(Loss Calculation)

- **예측값과 실제값(타깃) 사이의 오차 계산 -> 모델 성능을 수치적으로 평가**
- **이를 통해 신경망이 올바로 학습하고 있는지 판단**

Neural Network

역전파 (Backpropagation)

3) 오류 역전파(Backpropagation)

- 손실 함수로부터 계산된 오차를 출력층에서부터 입력층 방향으로 역전파

- 각 노드의 출력값에 대한 손실 함수의 미분값 계산, 오차가 각 가중치에 미치는 영향 파악

4) 가중치 업데이트(Weight Update)

- 계산된 오차의 영향을 바탕으로, **경사 하강법(Gradient Descent)** (or 그 외) 을 사용하여 네트워크의 가중치 조정
- 손실 함수의 값을 최소화

Neural Network

역전파 (Backpropagation) - 기울기(gradient)

- 오류 역전파 : 예측 값과 정답과의 차이를 줄이는 방향으로 가중치 수정

  → 가중치를 수정을 통한 오차 축소, How?

  → 가중치가 오차에 얼마나 영향을 주었는지 파악

  → 기울기(gradient)

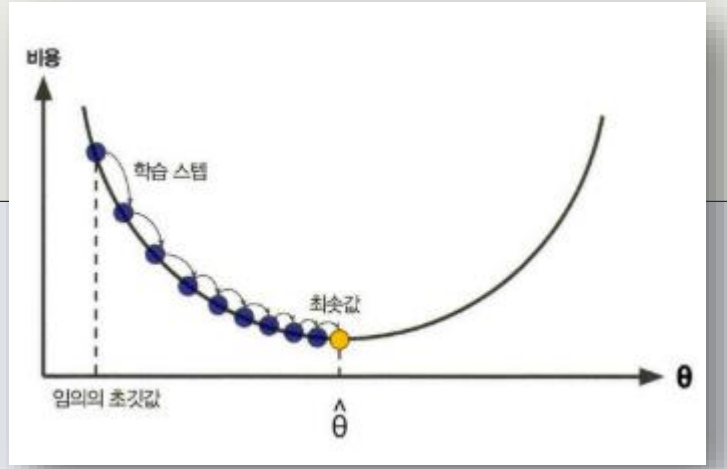
  → 기울기 = 변화율 = 미분값 = ∂오차/∂가중치

가중치가 바뀌었을 때 오차가 얼마나 바뀌는가?

Neural Network

역전파 (Backpropagation) – 경사 하강법 (Gradient Descent)

- 경사 하강법 : 오차를 최소화하기 위해 가중치를 수정해 나가는 방법(알고리즘)
  - Gradient : 기울기 (=미분값 = 변화율)
  - Descent : 내려간다 (오차를 줄임)
  - $w_{new} = w_{old} - \eta \cdot \partial L / \partial w$

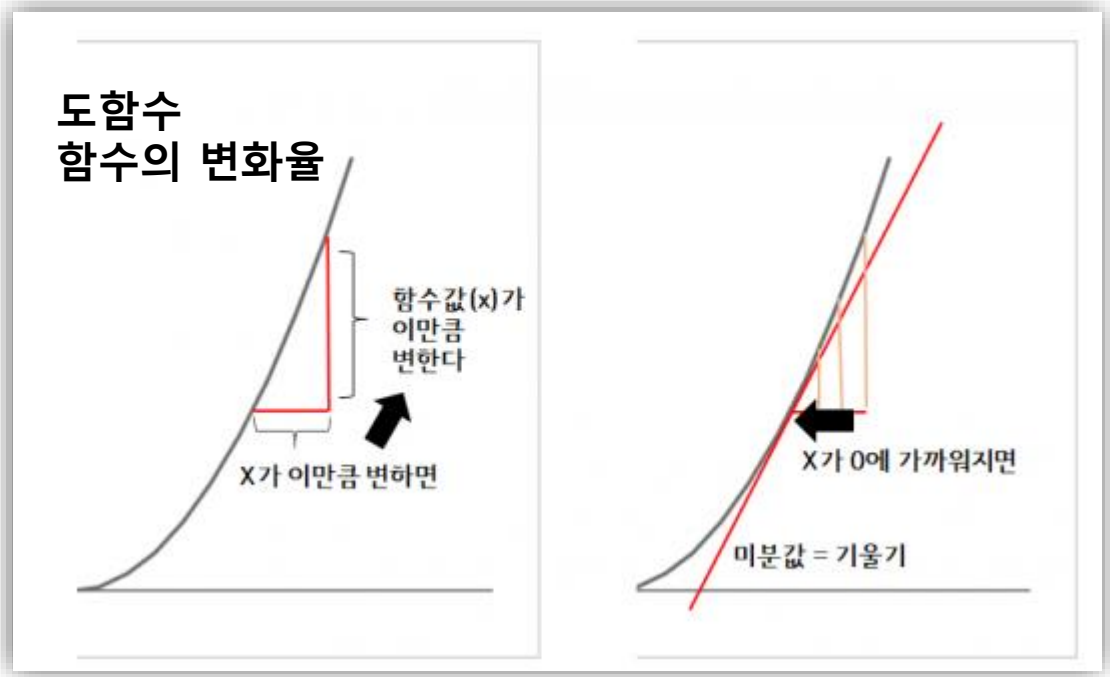


Neural Network

역전파 (Backpropagation)

오차의 미분 : 가중치가 변했을 때 오차 값이 얼마나 변하는지 확인

기울기 = 미분값



함수값의 변화량 / x의 변화량
= 기울기 = 변화율 = 미분값

곡선 위의 한 점에 그은 접선의 기울기
= (그 점에서의) 미분값 = gradient

Neural Network

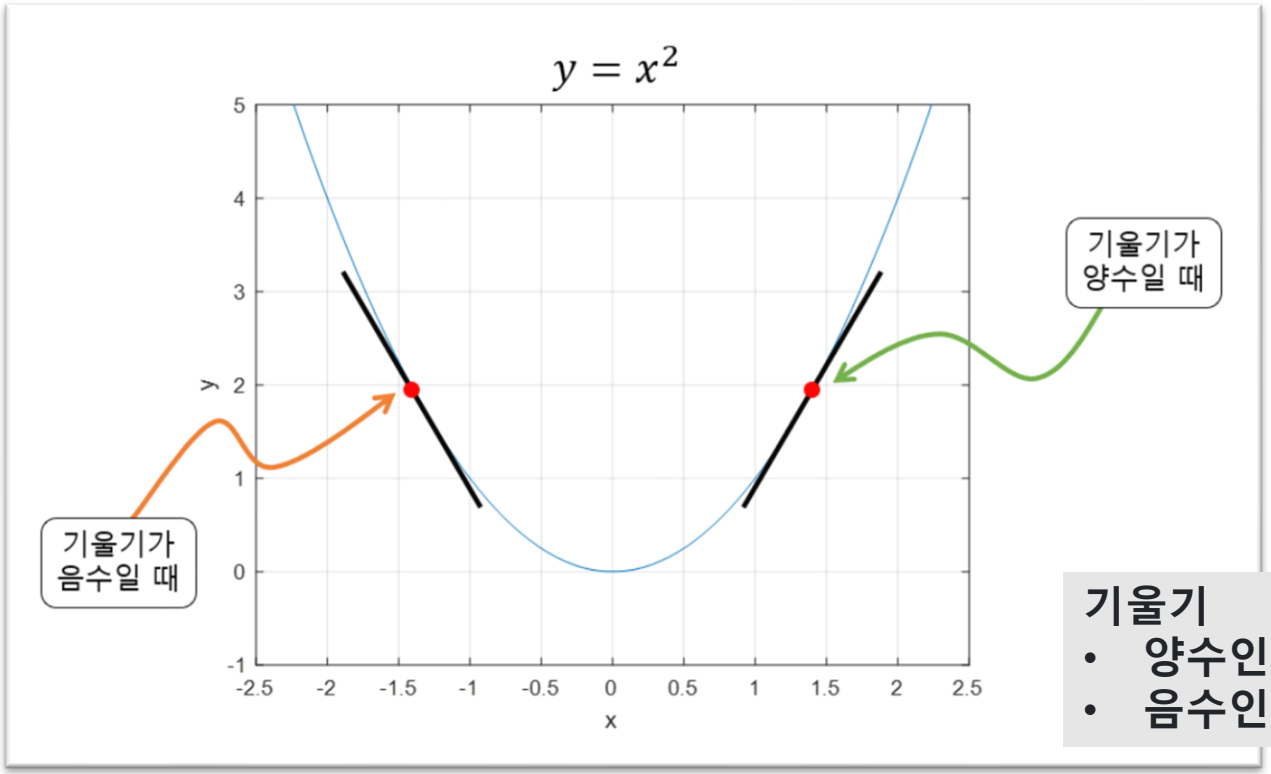
역전파 (Backpropagation)

각 노드의 가중치에 대한 오차의 기울기(gradient) 계산

4. 오류 역전파(Error Backpropagation)

> 계산된 오차는 출력층에서부터 입력층 방향으로 역전파 되어 각 가중치가 오차에 얼마나 기여하는지 계산

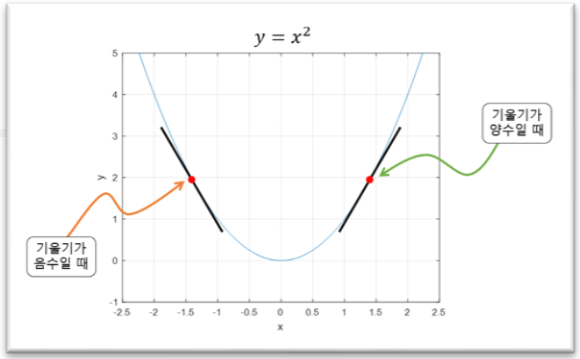
각 가중치가 오차에 미치는 영향 계산 -> 가중치 조정



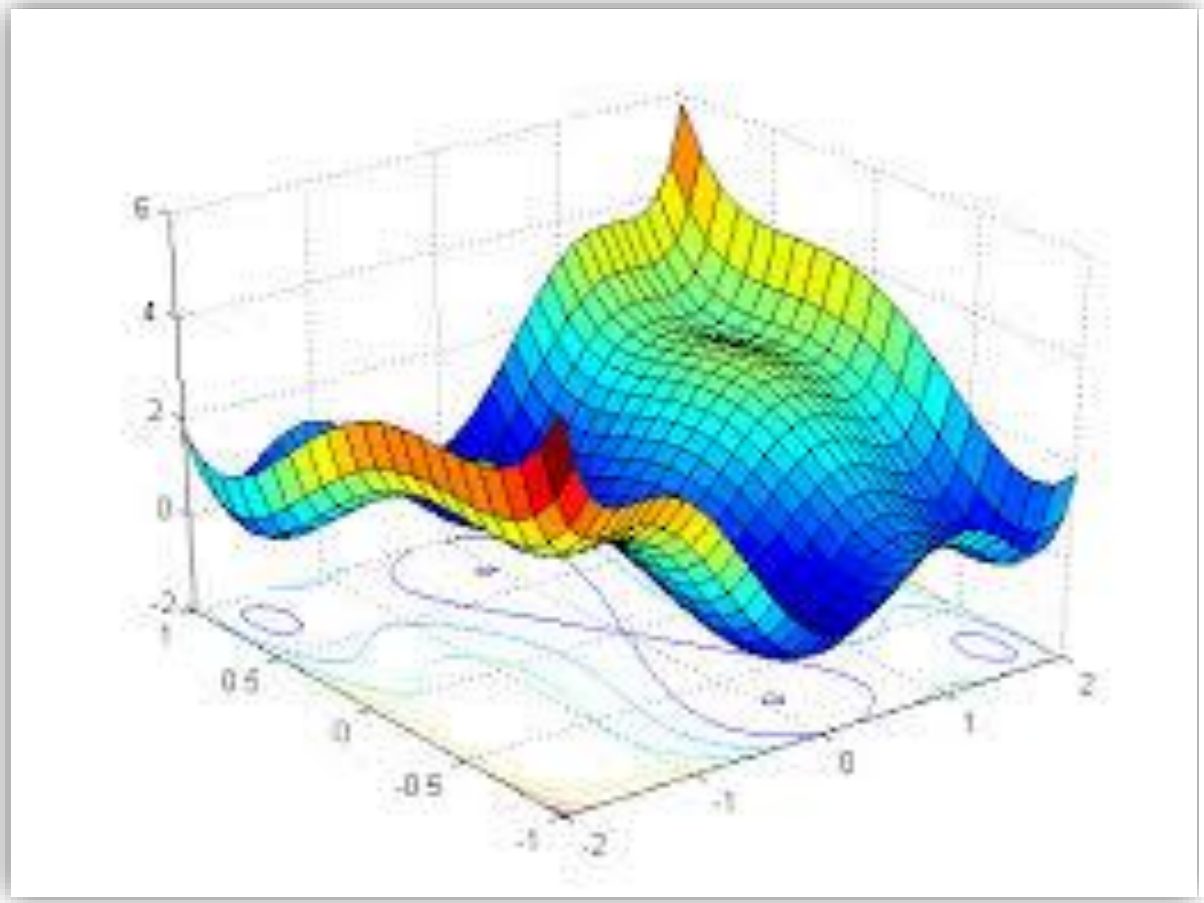
기울기
- 양수인 경우 : *x* 값이 증가할수록 함수 값 증가
- 음수인 경우 : *x* 값이 증가할수록 함수 값이 감소

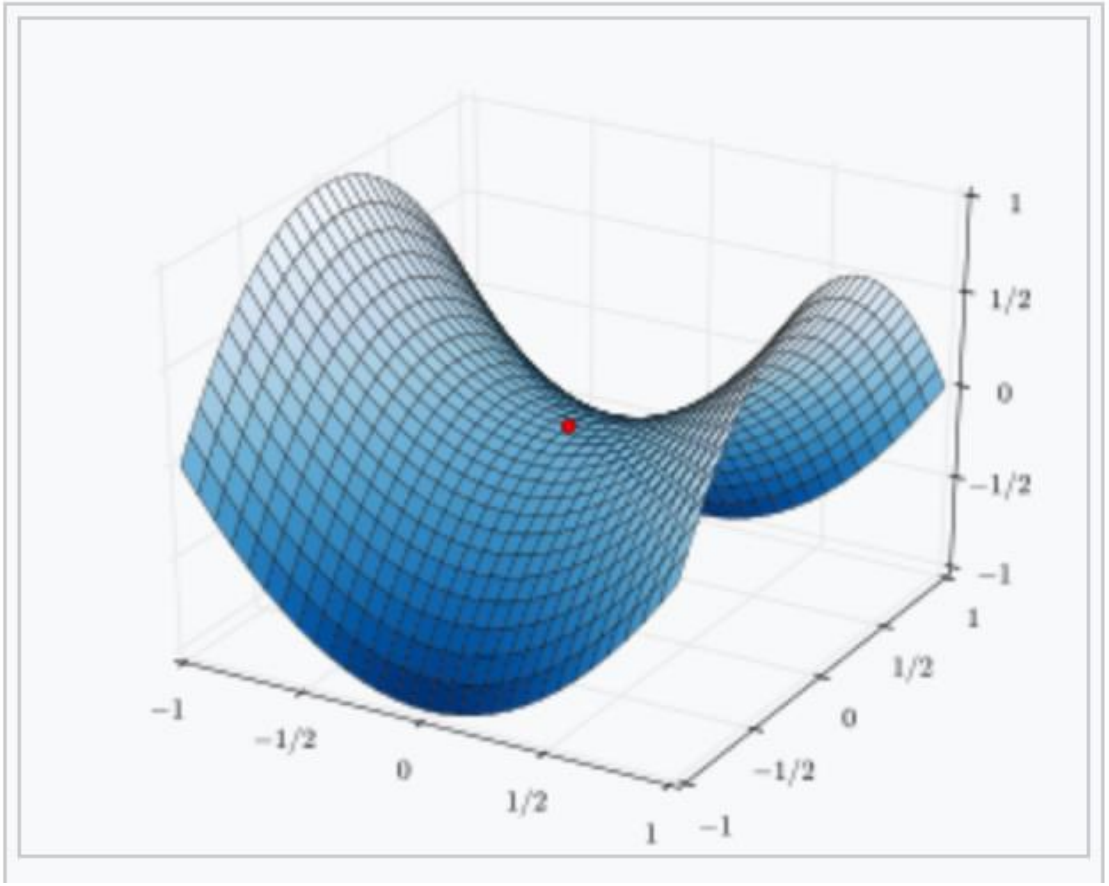
Neural Network

역전파 (Backpropagation)

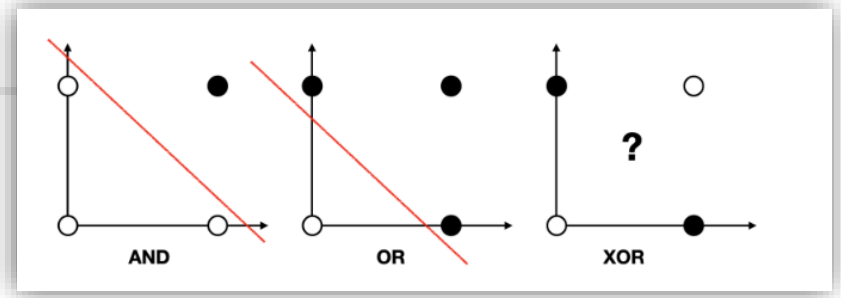


손실 함수 그래프





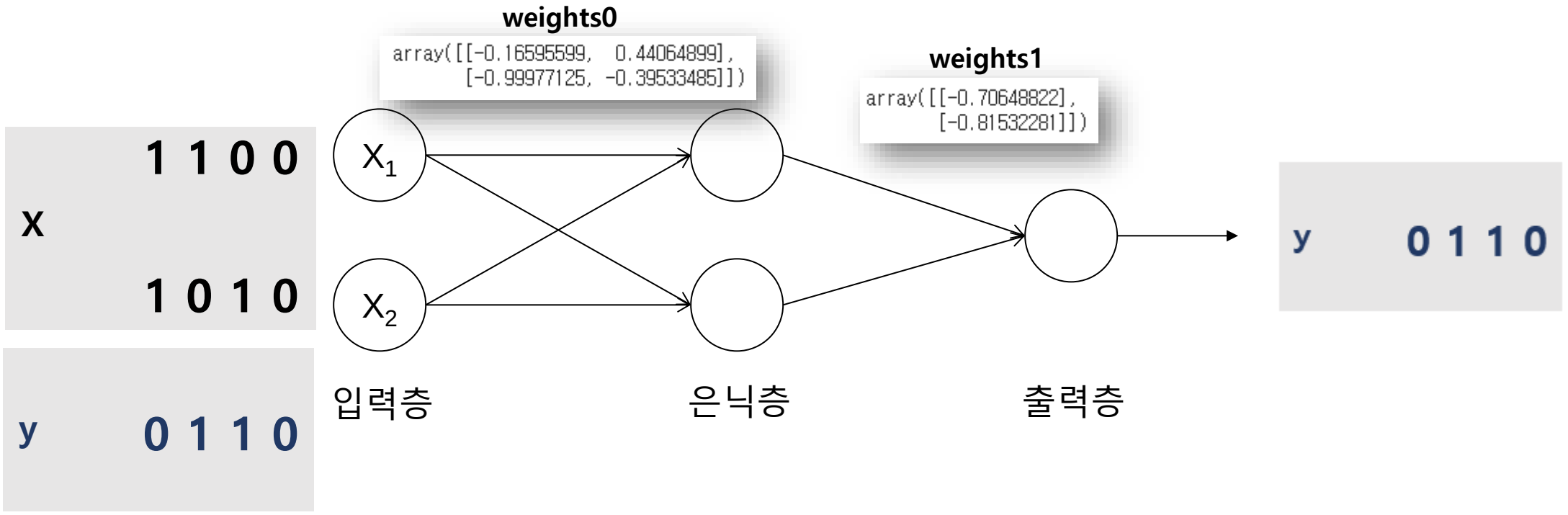
Neural Network



역전파 (Backpropagation)

- 실습

  4.03.NN.Backpropagation.ipynb



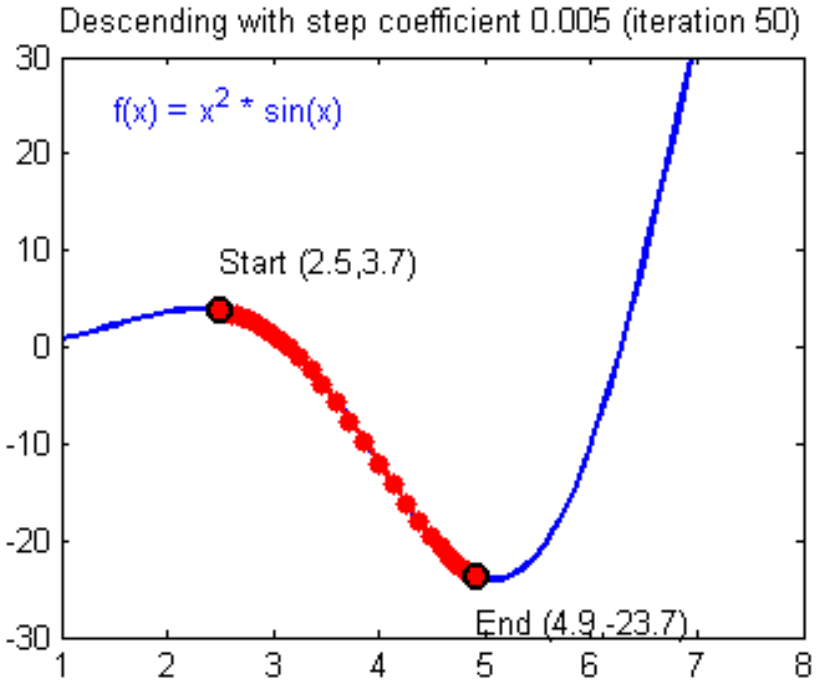

```
# 입력 데이터와 타깃 데이터
X = np.array([[0,0], [0,1], [1,0], [1,1]]) # 2차원 입력
y = np.array([[0], [1], [1], [0]]) # XOR 문제
```

Neural Network

경사하강법 (Gradient Descent)

순전파(활성화함수) → 오차 계산(손실함수) → 오류 역전파 → 가중치 업데이트

- 최적화 문제(Optimization problem)를 풀기 위한 알고리즘
- 비용 함수(Cost Function)를 최소화하는 파라미터를 찾는 데 사용



- 최적화 문제 : 주어진 조건(제약 조건) 내에서 특정 함수 (목적함수 또는 비용 함수)의 최대값이나 최소값을 찾는 문제
- 딥러닝에서의 최적화 문제 : 모델의 성능을 나타내는 비용 함수나 손실 함수의 최소값을 찾아내는 것

https://hackernoon.com/life-is-gradient-descent-880c60ac1be8

Neural Network

경사하강법 (Gradient Descent)

- 함수의 기울기(gradient)를 사용하여 함수의 최소값을 찾는 반복적인 방법
  (함수의 기울기: 함수가 가장 가파르게 증가하는 방향)
- 기울기의 반대 방향으로 반복적으로 조금씩 이동함으로써 함수의 **최소값을** 탐색

1.초기화: 파라미터를 임의의 값(또는 지정된 시작점)으로 설정

2.기울기 계산: 현재 파라미터에서 비용 함수의 기울기 계산

* 기울기 : 파라미터에 대한 비용 함수의 변화율

어느 방향으로 파라미터를 조정해야 비용이 감소하는지를 알려줌

3. 파라미터 업데이트: 계산된 기울기에 학습률(learning rate)을 곱한 값으로 파라미터를 조정

* 학습률: 한 번의 반복으로 파라미터를 얼마나 업데이트할지를 결정하는 하이퍼파라미터

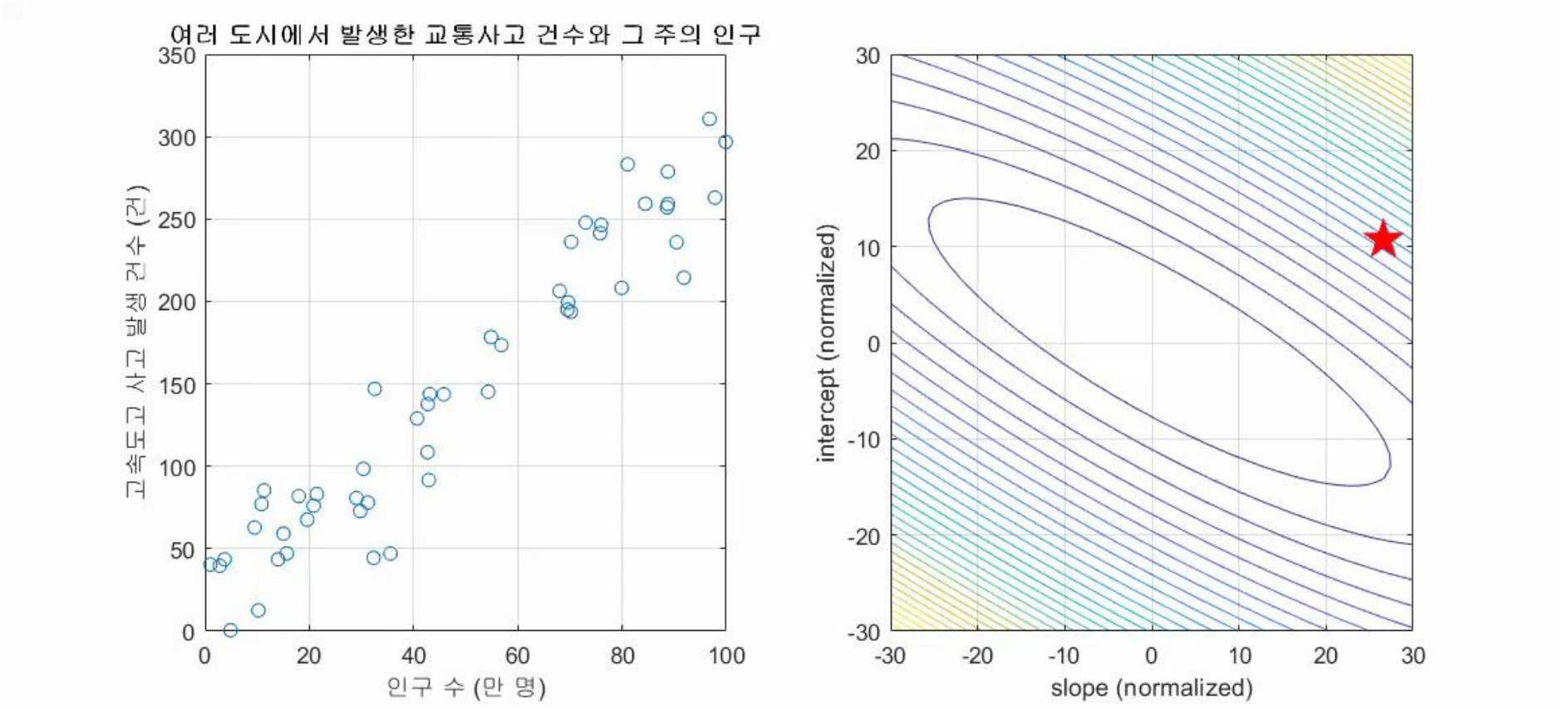
각 단계에서 얼마나 큰 걸음으로 이동할지 결정

4. 반복: 새로운 파라미터로 다시 기울기를 계산하고 파라미터를 업데이트하는 과정을 반복

Neural Network

경사하강법 (Gradient Descent)

경사하강법(gradient descent)을 이용해 비용함수의 최솟값을 찾는 과정

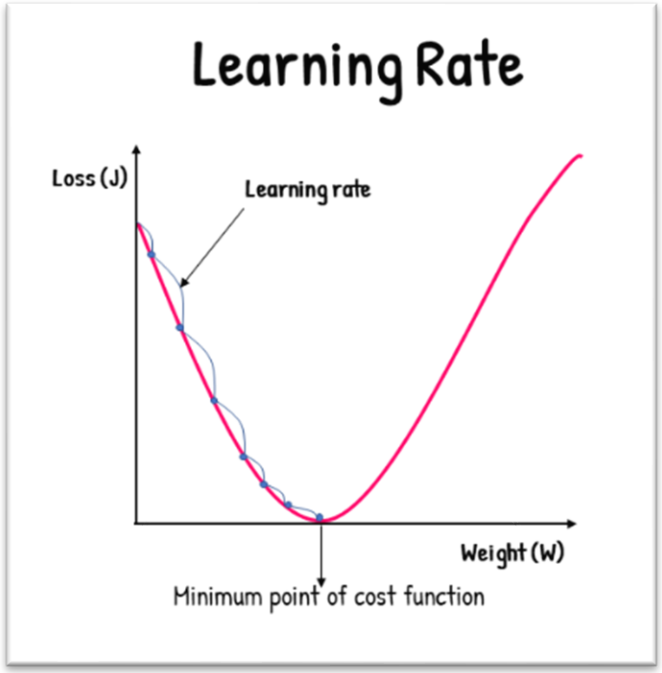


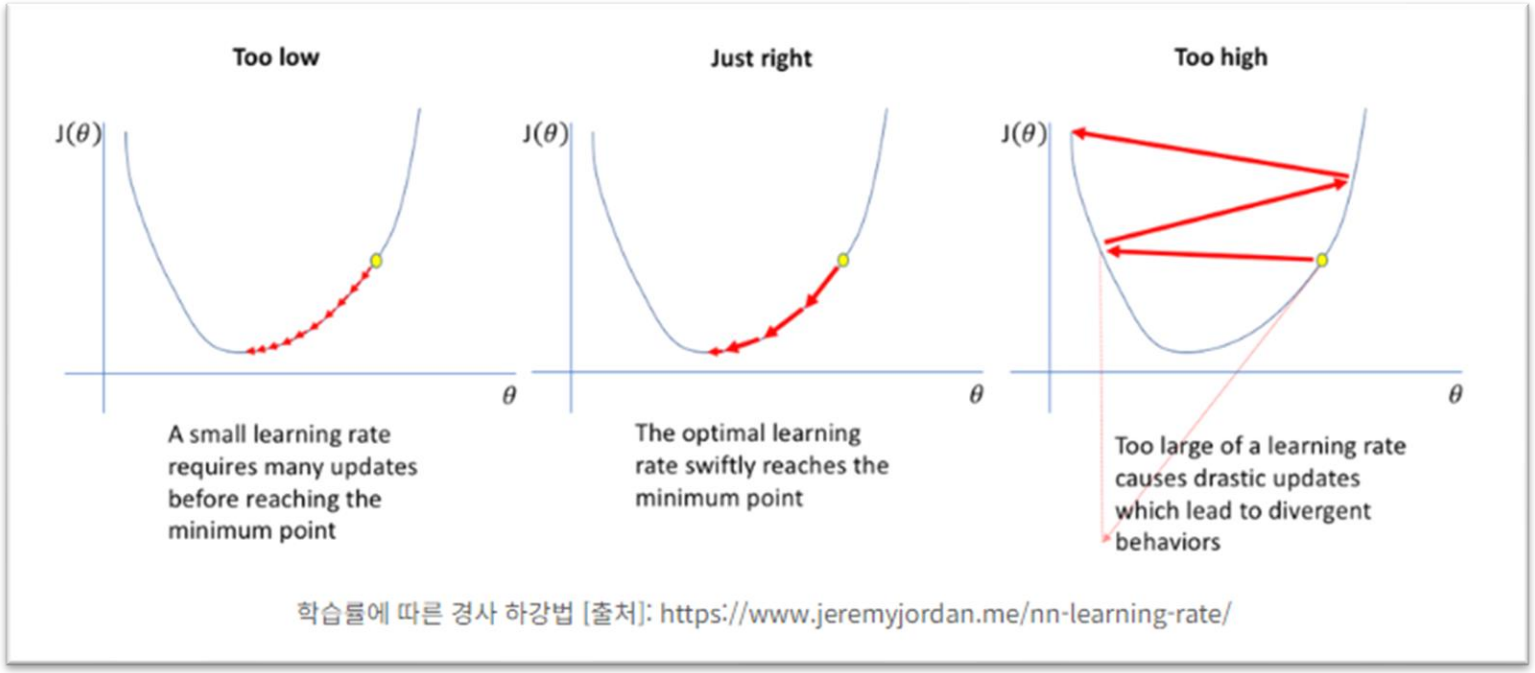
Neural Network

경사하강법 (Gradient Descent)

학습률: 한 번의 반복으로 파라미터를 얼마나 업데이트할지를 결정하는 하이퍼파라미터

각 단계에서 얼마나 큰 걸음으로 이동할지 결정





학습률에 따른 경사 하강법 [출처]: https://www.jeremyjordan.me/nn-learning-rate/

Neural Network

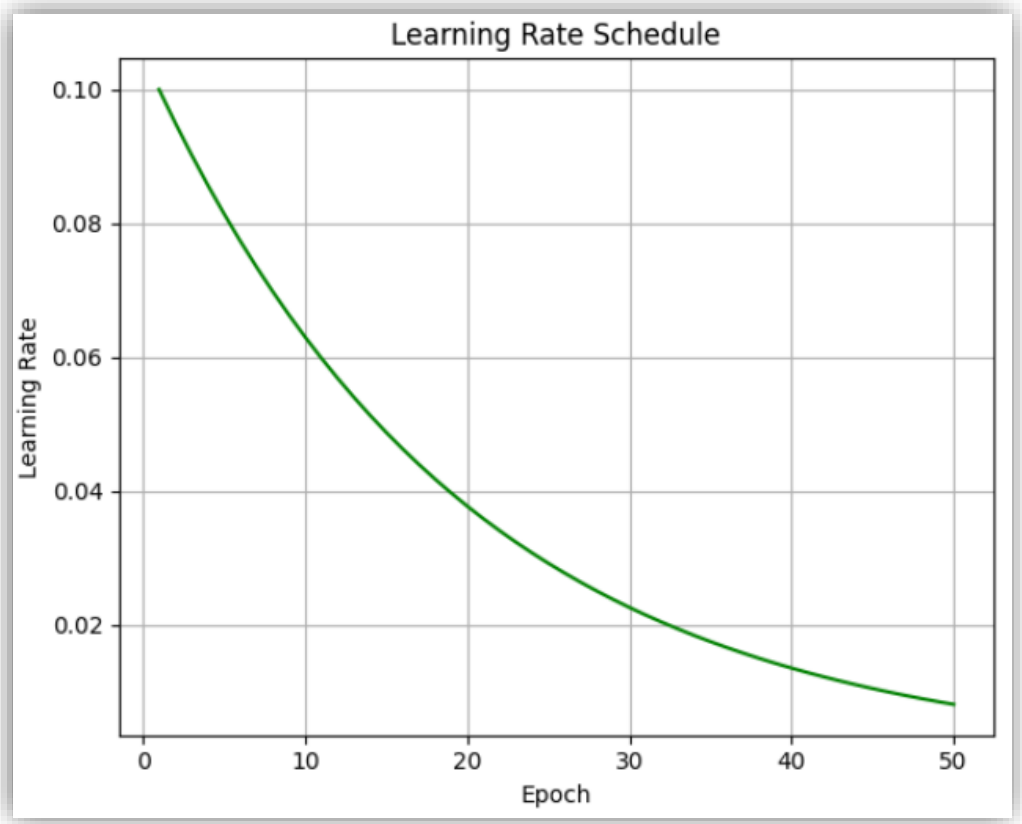
경사하강법 (Gradient Descent)

학습률: 한 번의 반복으로 파라미터를 얼마나 업데이트할지를 결정하는 하이퍼파라미터

각 단계에서 얼마나 큰 걸음으로 이동할지 결정

- 실습

  4.03.lr.tracking.ipynb



Neural Network

경사하강법 (Gradient Descent)

- **배치 경사 하강법 (Batch Gradient Descent)**:

  전체 훈련 데이터셋을 사용하여 한 번에 비용 함수의 기울기를 계산

- **확률적 경사 하강법 (Stochastic Gradient Descent, SGD)**:

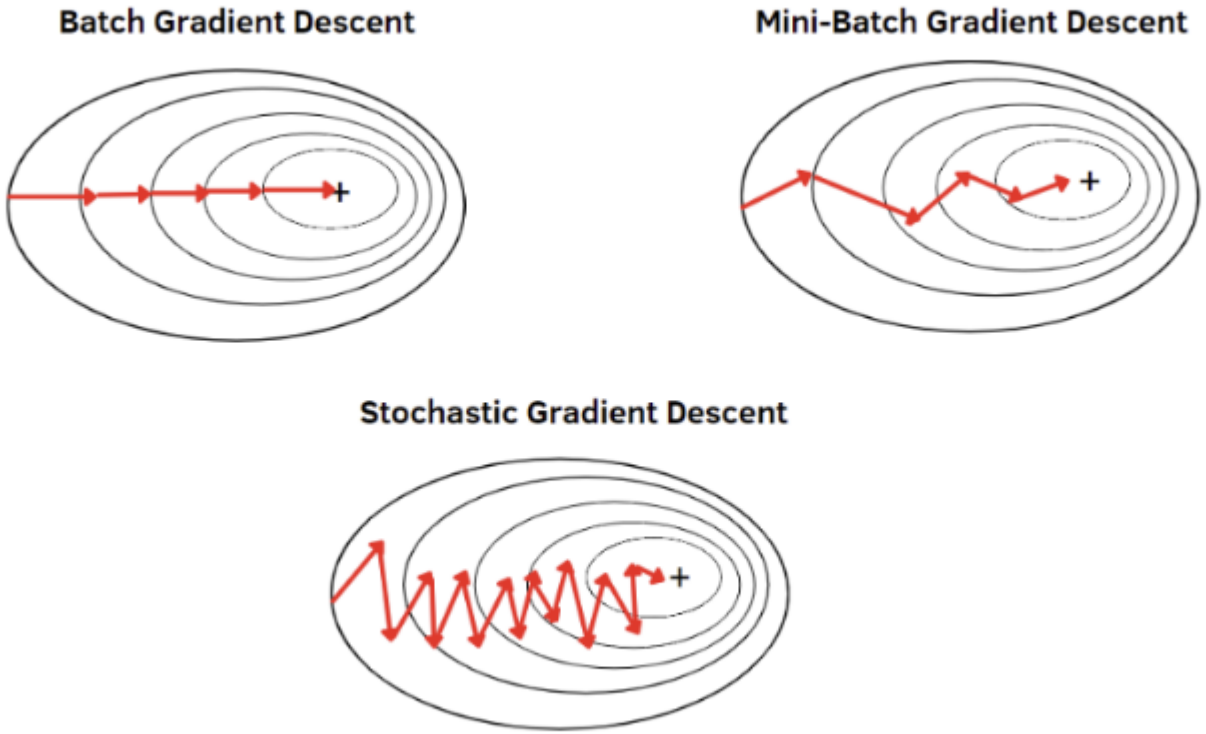
  각 반복에서 **단일 훈련 샘플**을 무작위로 선택하여 기울기를 계산

- **미니 배치 경사 하강법 (Mini-Batch Gradient Descent)**:

  훈련 데이터셋을 **미니 배치**로 나누고, 각 반복에서 하나의 미니 배치를 사용하여 기울기 계산

Neural Network

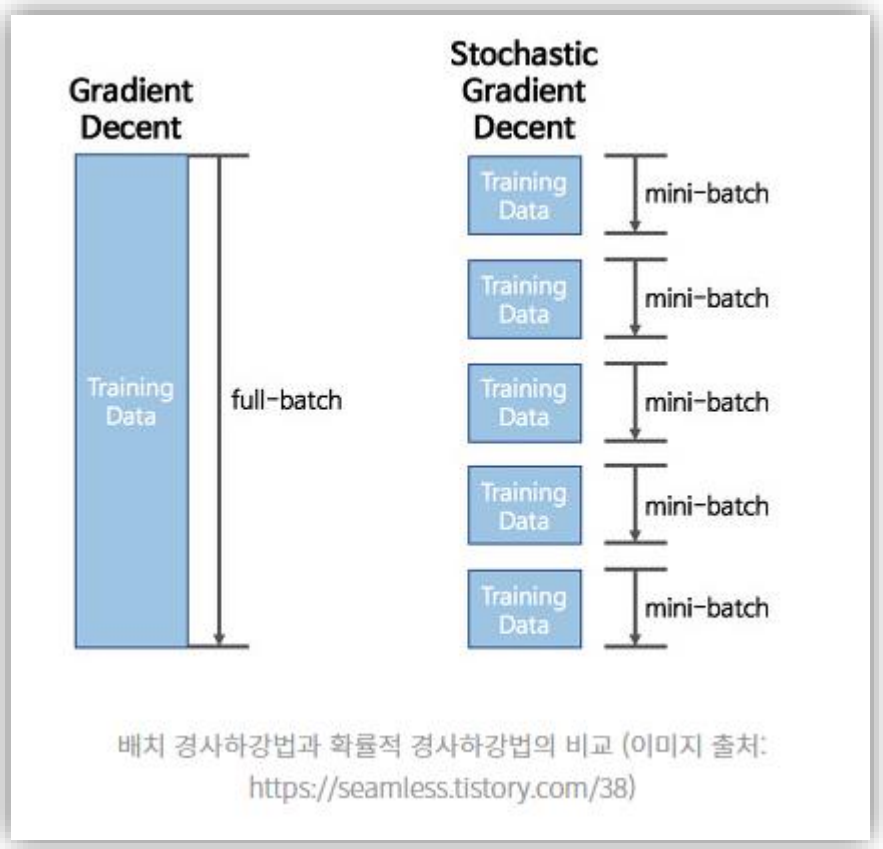
경사하강법 (Gradient Descent)



배치 경사 하강법, 확률적 경사 하강법, 미니배치 경사 하강법 [출처]: https://www.analyticsvidhya.com/blog/2022/07/gradient-descent-and-its-types/

Neural Network

경사하강법 (Gradient Descent)



배치 경사하강법과 확률적 경사하강법의 비교 (이미지 출처: https://seamless.tistory.com/38)

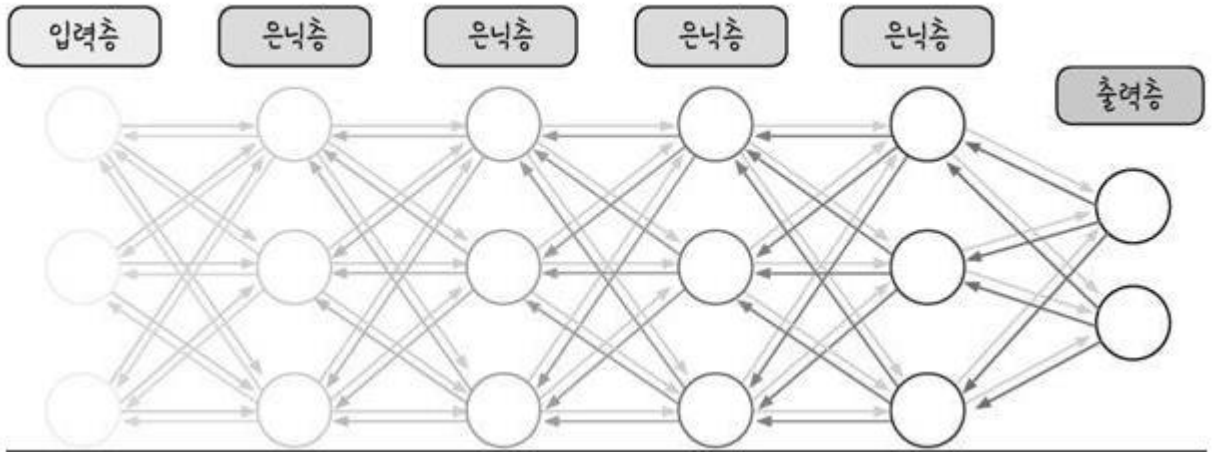
```
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=50,
          batch_size=32)
```

Neural Network

경사하강법 (Gradient Descent) - 경사 소실 문제

Vanishing Gradient Problem

- 신경망의 깊이가 깊어질수록(많은 층을 가질수록) 네트워크의 초기 층으로 가면서 **경사(오류의 변화율, 가중치에 대한 손실 함수의 미분 값)가 점점 작아지는 현상**
- 네트워크의 초기 층에 있는 가중치는 거의 업데이트되지 않게 되어, 학습이 효과적으로 이루어지지 않는 상태



https://cbjsena.blogspot.com/2018/12/blog-post_25.html

Neural Network

경사하강법 (Gradient Descent) - 경사 소실 문제

Vanishing Gradient Problem

발생 원인

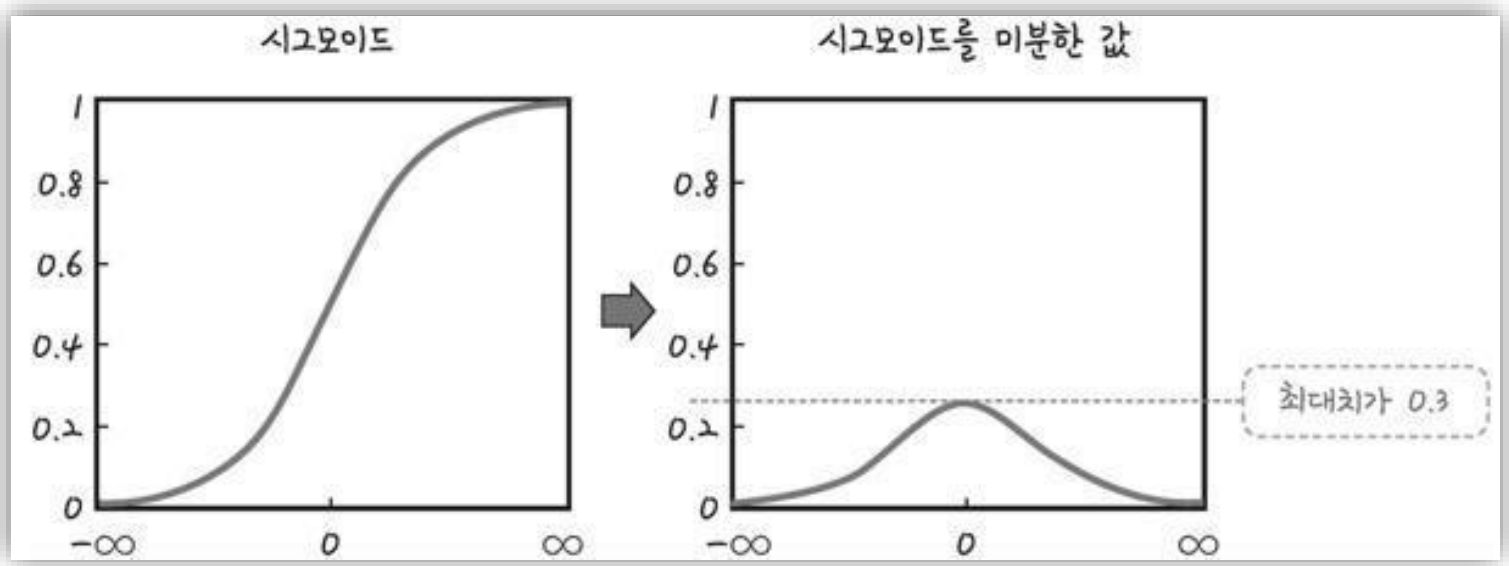
- **활성화 함수**:
  > 시그모이드(Sigmoid)나 하이퍼볼릭 탄젠트(Tanh) 같이 **출력 값의 범위가 제한**된 활성화 함수 사용 시 (시그모이드 함수 출력 범위 0 ~ 1)
  > 위 함수들의 **미분 값은 입력 값이 0에 가까워질수록 매우 작아지며**, 네트워크가 깊어질수록 경사가 점점 더 작아짐

- **연쇄 법칙(Chain Rule)과 그래디언트 전파**:
  > 역전파 알고리즘에서는 **연쇄 법칙**을 사용하여 그래디언트(경사값)를 계산하고 전파
  > 출력층으로부터 입력층으로 '기울기' 전파 시, 각 층의 경사값을 이전 층의 경사값에 곱함
  > 네트워크의 깊이가 깊어질수록, 매우 작은 미분 값들(경사값)이 계속 곱해지면서 전체 경사값이 점점 더 작아지는 결과 초래

Neural Network

경사하강법 (Gradient Descent) - 경사 소실 문제

미분 값은 입력 값이 0에 가까워질수록 매우 작아지며 ...



https://cbjsena.blogspot.com/2018/12/blog-post_25.html

Neural Network

경사하강법 (Gradient Descent) - 경사 소실 문제

$$ReLU(x) = max(0,x)$$

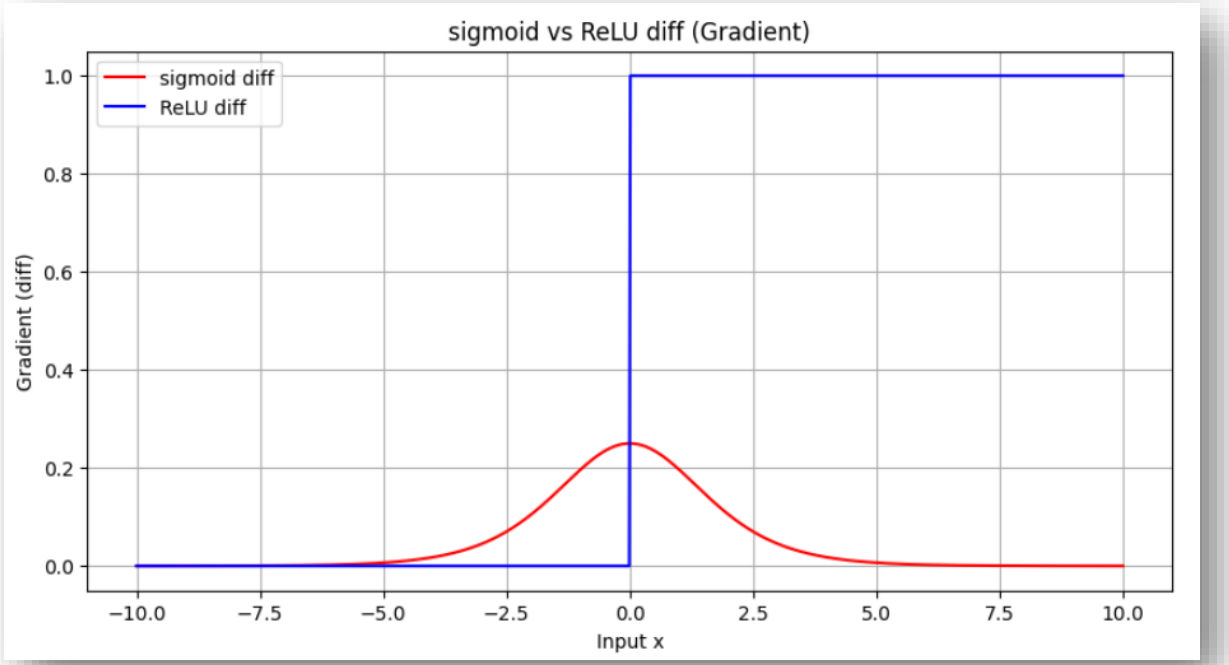
$$ReLU'(x) = \begin{cases} 1 & (if\ x>0) \\ 0 & (if\ x<0) \end{cases}$$

입력이 양수면 기울기 그대로

Neural Network

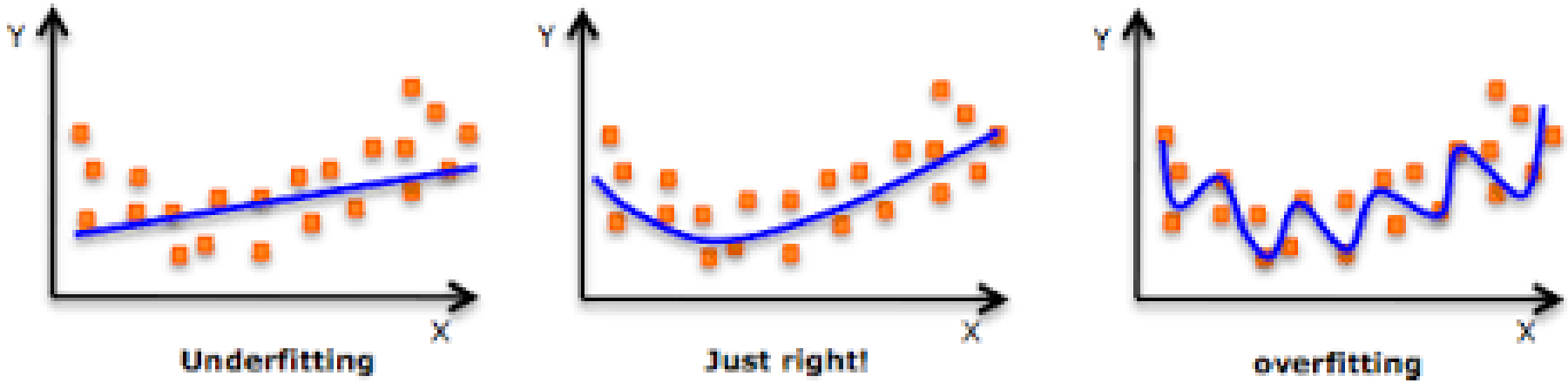
경사하강법 (Gradient Descent) - 경사 소실 문제

ReLU *vs* Sigmode



Neural Network

Regularization - **Overfitting**



Neural Network

Regularization - **Overfitting**

- **과적합**
  - Overfitting means that the model performs well on the training data, but it does not generalize well
  - 모델이 훈련 데이터에서 좋은 성능을 보이지만 일반화 성능은 떨어짐
    - 새로운 데이터를 잘 예측하지 못함
    - 학습 데이터를 얼마나 잘 설명하느냐 보다 새로운 데이터를 잘 예측하는 것이 중요
- **언제 발생?**
  - 모델이 복잡한 경우
    - 모델에 포함된 독립변수의 수가 많은 경우
  - 학습 데이터에 존재하는 독립변수의 값에 민감하게 반응하는 경우

Q. 해결 방안?

Neural Network

Regularization - **정규화**

주요 정규화 기법들:

1.L1 규제(Lasso Regularization)

- 가중치의 절대값에 비례하는 비용을 추가

  -> **가중치를 정확히 0으로 만들어** 해당 특성의 영향을 제거 -> 모델 단순화

- 모델의 희소성을 증가시키는 효과

2.L2 규제(Ridge Regularization)

- 가중치의 제곱에 비례하는 비용을 추가

  -> 가중치의 **값을 줄여** 모델의 복잡도를 낮춤

- 가중치가 너무 크게 되어 과적합을 일으키는 것을 방지

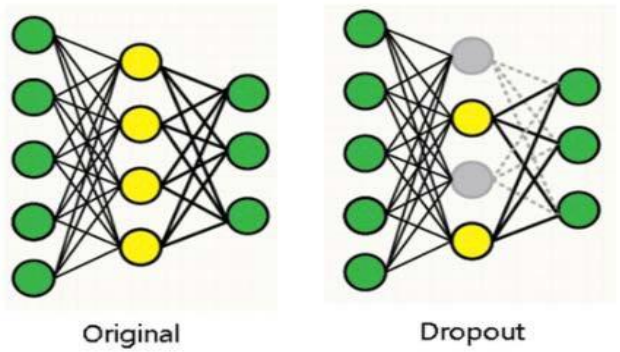
Neural Network

Regularization - **정규화**

3. 드롭아웃(Dropout)

- 학습 과정에서 무작위로 일부 **노드를 비활성**, 학습에 기여하는 것을 일시적으로 중단시킴
- 신경망이 특정 노드나 노드의 그룹에 과도하게 의존하는 것을 방지하여 과적합을 줄임

(test 시에는 모든 노드 사용)



4. 조기 종료(Early Stopping)

- 검증 세트의 성능이 더 이상 개선되지 않을 때 학습을 조기에 종료하는 기법
- 학습 과정에서 모델의 일반화 성능이 최적점에 도달하면 그 시점에서 학습을 멈춤

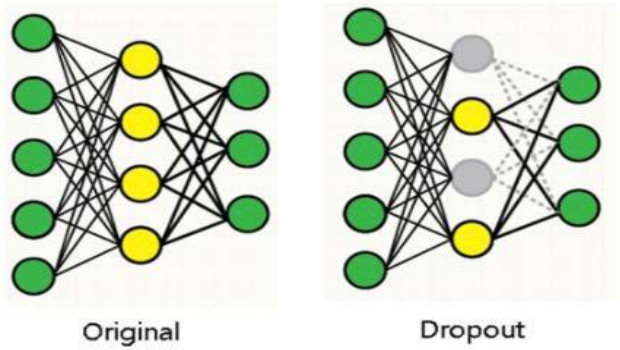
Neural Network

Regularization - **정규화**

3. 드롭아웃(Dropout)

- 학습 과정에서 무작위로 일부 **노드를 비활성**, 학습에 기여하는 것을 일시적으로 중단시킴
- 신경망이 특정 노드나 노드의 그룹에 과도하게 의존하는 것을 방지하여 과적합을 줄임

(test 시에는 모든 노드 사용)




```
model = Sequential([
    Dense(64, activation='relu', input_shape=(100,)),
    Dropout(0.5),  # 50% 드롭아웃
    Dense(10, activation='softmax')
])
```

Neural Network

Regularization - **정규화**

4. 조기 종료(Early Stopping)

- 검증 세트의 성능이 더 이상 개선되지 않을 때 학습을 조기에 종료하는 기법
- 학습 과정에서 모델의 일반화 성능이 최적점에 도달하면 그 시점에서 학습을 멈춤

```
from tensorflow.keras.callbacks import EarlyStopping

early_stop = EarlyStopping(
    monitor='val_loss',      # 검증 손실을 모니터링
    patience=5,              # 5 에포크 연속으로 개선되지 않으면 중단
    restore_best_weights=True  # 가장 성능 좋았던 가중치로 복원
)

model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=100,
    batch_size=32,
    callbacks=[early_stop]
)
```

NN 학습 프로세스

Q. Optimizer?

Regularization?

Generalization?

Neural Network

NN 학습 프로세스

- **Regularization (정규화)**
  - **과적합 방지:** 모델이 학습 데이터에 너무 정확하게 맞춰져 새로운 데이터에 대한 성능이 떨어지는 현상 방지

    L1, L2 정규화, 드롭아웃(Dropout) 등의 기법 사용
  - **모델 복잡도 조절:** 일반화된 모델 생성에 기여

순전파(활성화함수) → 오차 계산(손실함수) → 오류 역전파 → 가중치 업데이트

(과적합 방지 -> 정규화(Regularization))

Neural Network

NN 학습 프로세스

- **Generalization (일반화)**
  - 모델이 새로운 데이터에 대해 얼마나 잘 예측하는지를 나타내는 능력
  - **Generalization > Regularization**
  - **Regularization 기법 +**
    - ✓ **데이터 확장(Data Augmentation):**

      학습 데이터의 다양성을 높여 새로운 데이터에 대한 모델의 노출을 증가 -> 일반화
    - ✓ **교차 검증(Cross-validation):**

      학습 데이터셋을 여러 부분으로 나누고, 각 부분을 검증 데이터셋으로 사용하여 모델 평가

Neural Network

NN 학습 프로세스 – review

- **Key Word**
  - 순전파
  - 활성화 함수
  - 손실함수
  - 역전파
  - 경사하강법
  - 과적합
  - 정규화(Regularization) 일반화 (Generalization)

THANK YOU