3장. 딥러닝 기본기 다지기

요약 정리

1. 머신러닝 프로세스는 간략하게 **[문제 정의 및 데이터 준비하기 → 학습하기 → 추론 및 평가]**로 나눌 수 있습니다.
2. **[문제 정의 및 데이터 준비하기]**는 명확한 문제 정의와 데이터 전처리가 매우 중요합니다.
3. **[학습하기]**는 본격적으로 모델을 선택하고, 학습시키는 단계입니다. 하이퍼파라미터 실험 환경 등을 고려하여 학습시간을 효율적으로 활용할 수 있도록 해야 합니다.
4. **[추론 및 평가]**는 올바른 지표를 통해 모델의 성능을 신뢰할 수 있어야 합니다. 주어진 상황에 맞는 지표를 선택하는 것은 매우 어렵고 중요합니다.
5. 세 가지 절로 나누어 여러 가지 용어를 알아보았습니다.
    - 데이터 준비하기: 클래스 불균형, 과소표집과 과대표집, 회귀와 분류, 원핫 인코딩, 교차 검증
    - 학습하기: 하이퍼파라미터, 배치와 배치크기, 에폭과 스텝, 지도 학습, 비지도 학습, 과대적합과 과소 적합
    - 평가하기: 혼동행렬, 정확도, 정밀도와 재현율, F1-Score, ROC 곡선

요약 정리

6. **구글 데이터셋 검색과 캐글**은 데이터셋을 탐색하고 수집할 최적의 장소입니다. 그 외에도 공공 데이터 포털, AI Hub가 있습니다.
7. 문제는 **공유와 소통**를 통해 더 빠르게 해결될 수 있습니다. 국내에 이를 위한 다양한 커뮤니티가 존재한다는 점을 잊지마세요.

3장의 내용?

- 텐서플로우를 사용해보고, 신경망의 기본 개념에 대해 알아보자
  - 경사하강법과 역전파는 신경망을 다루기 위한 필수 개념이므로 많은 글을 참고하여 정확히 이해해야 함

- 케라스의 개발 과정
  - 매우 간단하지만 강력한 결과를 얻을 수 있음

- 이 장에서 배울 내용
  - 텐서플로우를 통한 기본 연산 : 즉시 실행 모드
  - 신경망 살펴보기 : 경사하강법, 역전파
  - 케라스 개발 과정
    - 학습 데이터 정의 → 모델 구성(model, compile) → 학습(fit) → 평가(evaluate, predict)

Gradient Descent

Keras Develop Process

BackPropagation

TF Eager Mode

기본 연산 해보기

- 텐서플로우 2.x는 파이썬과 같은 실행 방식으로 연산을 쉽게 수행할 수 있음
  - 텐서플로우 1.x에서는 계산 그래프를 직접 생성, 조작하는 등 자세히 들여다봐야 정확히 사용할 수 있었음

- 텐서플로우가 다루는 자료형: **텐서(Tensor)**
  - C, JAVA 언어에서 볼 수 있는 int, float, string 등과 같은 자료형에 해당
  - 여러 형태를 가질 수 있는 넘파이 배열(NumPy Array)
  - 배열 차원을 **랭크(Rank)**로 표현
  - 아래 표의 '주로 사용하는 표현'으로 소통하지만, 코드에 Rank로 표현된 부분이 있어 알아두면 좋음

예	주로 사용하는 표현	텐서 표현	랭크
0, 1, 2, …	스칼라(Scalar)	0–D Tensor	0
[1, 2, 3, 4, 5]	벡터(Vector)	1–D Tensor	1
[[1, 2, 3, 4, 5], [1, 2, 3, 4, 5]]	행렬(Matrix)	2–D Tensor	2
[…[..]…]	n차원 배열	n–D Tensor	n

[표 3–1] 우리가 사용하는 표현과 텐서의 표현 비교

기본 연산 해보기

- 텐서 랭크 확인해보기
  - tf.rank() 함수

[함께 해봐요] **텐서의 차원과 기본 연산** basic_calc.ipynb

```
01  a = tf.constant(2)  # 텐서를 선언합니다.
02  print(tf.rank(a))   # 해당 텐서의 랭크를 계산합니다.
```

```
tf.Tensor(0, shape=(), dtype=int32)
```

- 즉시 실행 모드를 통한 연산
  - 텐서플로우 2.x의 큰 장점: **Eager Mode(즉시 실행 모드) 지원**
    → 파이썬처럼 사용

```
01  # 필요 모듈을 임포트합니다.
02  import tensorflow as tf
03  import numpy as np
04
05  a = tf.constant(3)
06  b = tf.constant(2)
07
08  # 기본 연산
09  # 텐서 형태로 출력해보기
10  print(tf.add(a, b))        # 더하기
11  print(tf.subtract(a, b))   # 빼기
12
13  # 넘파이 배열 형태로 출력해보기
14  print(tf.multiply(a, b).numpy())   # 곱하기
15  print(tf.divide(a, b).numpy())     # 나누기
```

텐서에서 넘파이, 넘파이에서 텐서

- 넘파이 형태 배열로 변환하여 사용해보기
  - numpy()와 convert_to_tensor() 함수
  - tensor와 numpy array간 변환이 매우 유연

[함께 해봐요] **텐서에서 넘파이로, 넘파이에서 텐서로** basic_calc.ipynb

```
01  import tensorflow as tf
02  import numpy as np
03
04  c = tf.add(a, b).numpy()  # a와 b를 더한 후 NumPy 배열 형태로 변환합니다.
05  c_square = np.square(c, dtype = np.float32)
    # NumPy 모듈에 존재하는 square 함수를 적용합니다.
06  c_tensor = tf.convert_to_tensor(c_square)  # 다시 텐서로 변환해줍니다.
07
08  # 넘파이 배열과 텐서 각각을 확인하기 위해 출력합니다.
09  print('numpy array : %0.1f, applying square with numpy : %0.1f,
          convert_to_tensor : %0.1f' % (c, c_square, c_tensor))
```

```
numpy array : 5.0, applying square with numpy : 25.0, convert_to_tensor : 5.0
```

@tf.function

- 텐서플로우에서 자동으로 그래프를 생성(Auto-Graph)
  - 파이썬으로 구성된 코드를 고효율의 텐서플로우 그래프로 변환
  - if → tf.cond
  - for/while → tf.while_loop
  - for _ in dataset → dataset.reduce
  - 이와 연관하여 **'jax'**를 검색하고, 공부해보세요.

```
03 @tf.function
04 def square_pos(x):
05     if x > 0:
06         x = x * x
07     else:
08         x = x * -1
09     return x
```

@tf.function을 사용하지 않은 함수	@tf.function을 사용한 함수

def square_pos(x):
  if x > 0:
  x = x * x
  else:
  x = x * -1
  return x
square_pos
	@tf.function
	def square_pos(x):
	  if x > 0:
	  x = x * x
	  else:
	  x = x * -1
	  return x
	square_pos

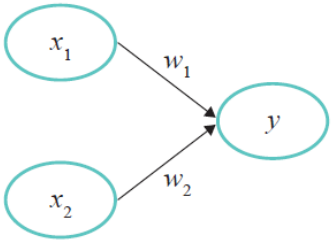
<function __main__.square_pos(x)>	<tensorflow.python.eager.def_function. Function at 0x1eefad00a58>

신경망

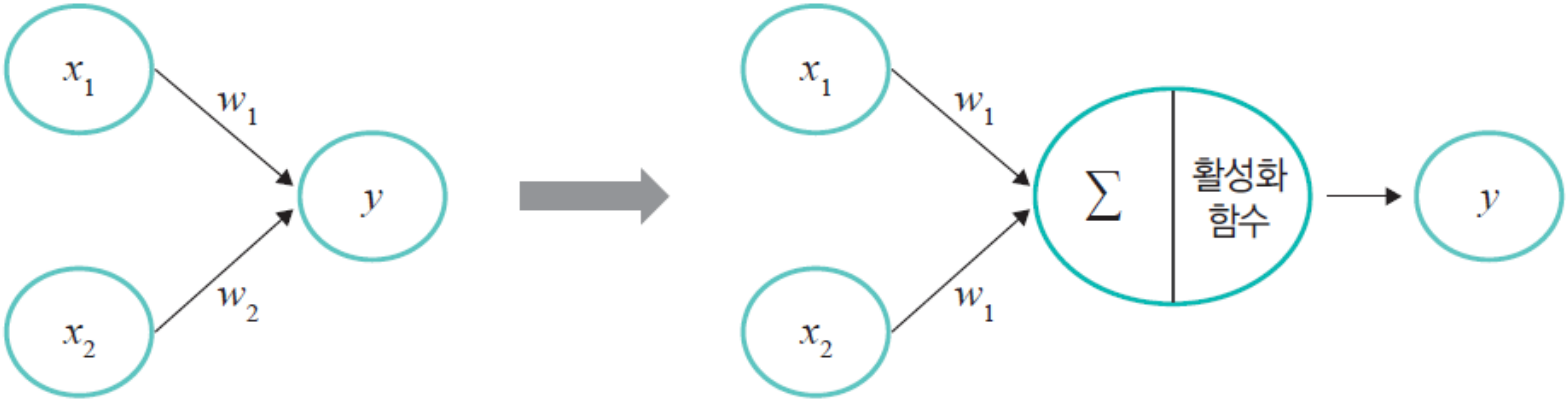
- 퍼셉트론 (Perceptron)
  - **여러 개의 신호를 입력으로 받아 하나의 값을 출력**
  - x는 입력, y는 출력, w는 가중치
  - x와 가중치 w를 곱한 값을 모두 더하여 하나의 값(y)로 만들어냄
  - 이때, 임곗값(threshold)과 비교하여 크면 1, 그렇지 않으면 0을 출력

    → **활성화 함수(Activation Function)**

    → 위에서 사용한 것은 계단 함수(Step Function)



[그림 3-1] 퍼셉트론



[그림 3-2] 퍼셉트론과 퍼셉트론의 기본 단위

신경망: OR 게이트 문제

[함께 해봐요] OR 게이트 구현해보기 perceptron.ipynb

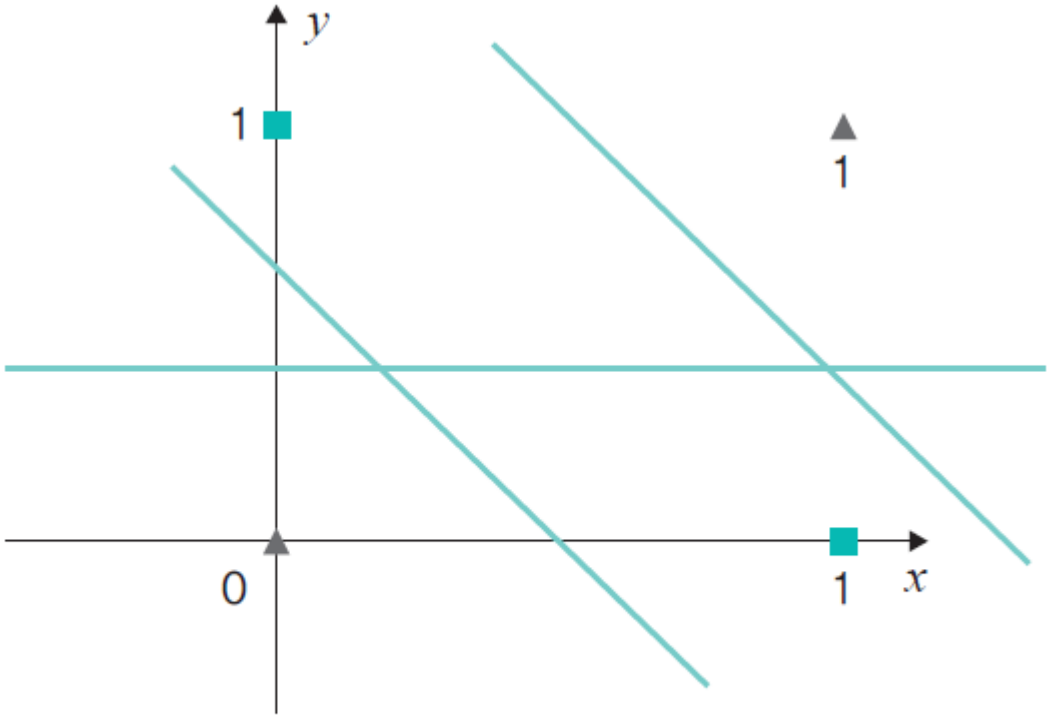
```
01 import tensorflow as tf
02 tf.random.set_seed(777) # 시드를 설정합니다.
03
04 import numpy as np
05 from tensorflow.keras.models import Sequential
06 from tensorflow.keras.layers import Dense
07 from tensorflow.keras.optimizers import SGD
08 from tensorflow.keras.losses import mse
09
10 # 데이터 준비하기
11 x = np.array([[0, 0], [1, 0], [0, 1], [1, 1]])
12 y = np.array([[0], [1], [1], [1]])
13
14 # 모델 구성하기
15 model = Sequential()
16 # 단층 퍼셉트론을 구성합니다.
17 model.add(Dense(1, input_shape = (2, ), activation = 'linear'))
18
19 # 모델 준비하기
20 model.compile(optimizer = SGD(),
21               loss = mse,
22               metrics = ['acc'])  # list 형태로 평가지표를 전달합니다.
23
24 # 학습시키기
25 model.fit(x, y, e
```

```
Epoch 194/500
4/4 [==============================] - 0s 2ms/sample - loss: 0.1164 - acc: 1.0000
... 생략 ...
```

- tf.random.set_seed(777)
  - **실험의 재생산성**
- Dense층
  - 퍼셉트론 생성
  - 밀집층, 다층 퍼셉트론, 완전 연결층 등
- Dense(1, input_shape = (2,))
  - 두 개의 특성을 가지는 1차원 데이터를 입력으로 받고, 한 개의 출력을 가지는 Dense층
  - '1'은 **은닉 유닛(hidden unit)**이라고 표현
- AND, NAND 게이트도 해결해보세요!

다층 퍼셉트론

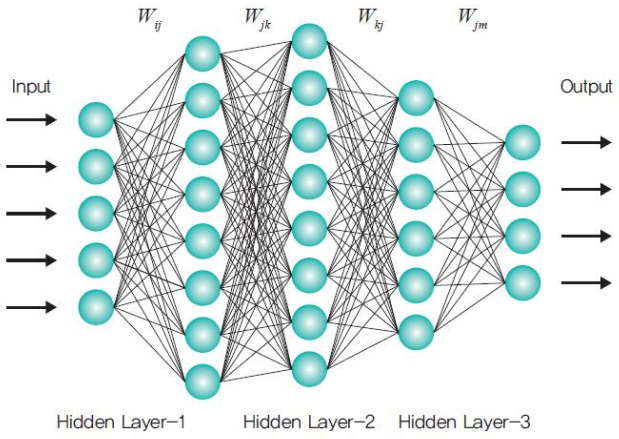
- 네모 점과 세모 점을 구분지을 수 있나요?



[그림 3–3] 네모 점과 세모 점으로 구분짓기

다층 퍼셉트론

- 전과 같은 문제를 해결한 것이 **다층 퍼셉트론(Multi-Layer Perceptron)**
  - 그림의 선이 전부 가중치에 해당함
  - 실제로 사용한 퍼셉트론은 굉장히 많음
    → 연산 비용이 큼
    → **벡터화(Vectorization)**을 이용



[그림 3-4] 다층 퍼센트론

m : 데이터의 개수
n : 데이터 특성의 개수
k : 층의 유닛 수

$$\begin{bmatrix} X_{11} & \cdots & X_{1n} \\ \vdots & \ddots & \vdots \\ X_{m1} & \cdots & X_{mn} \end{bmatrix} \cdot \begin{bmatrix} W_{11} & \cdots & W_{1k} \\ \vdots & \ddots & \vdots \\ W_{n1} & \cdots & W_{nk} \end{bmatrix} = \begin{bmatrix} Z_{11} \end{bmatrix}$$

(m, n) (n, k) (m, k)

[그림 3-5] 벡터의 내적

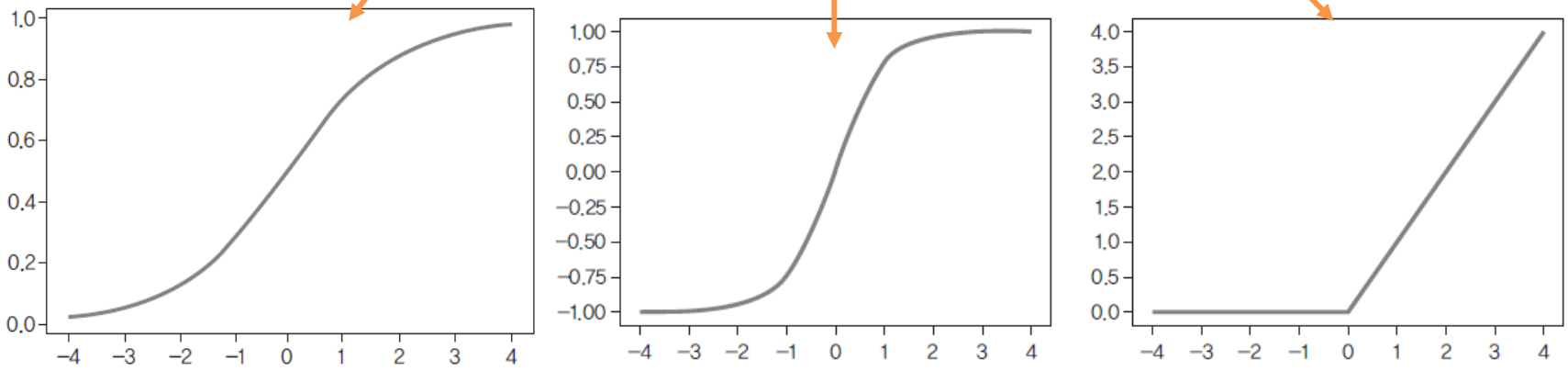
다층 퍼셉트론(실습)

- 제공되는 코드를 통해 XOR 게이트 문제를 해결해보세요!

활성화 함수

- XOR 게이트 문제에서 ReLU 활성화 함수를 사용
  - 비선형 활성화 함수
  - **선형 활성화 함수를 쓰면, f(f(f(x))) → f(x)와 동일**
  - 층을 쌓는 의미가 없어져... → 비선형 활성화 함수 사용으로 해결

- 대표적으로 사용되는 **시그모이드(Sigmoid), 하이퍼볼릭 탄젠트(tanh), ReLU** 활성화 함수
  - 그 외에 PReLU, ELU, Leaky-ReLU, swish 등이 존재

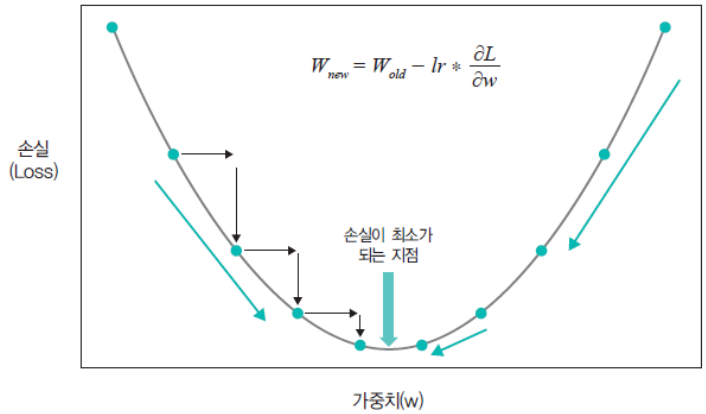


활성화 함수(실습)

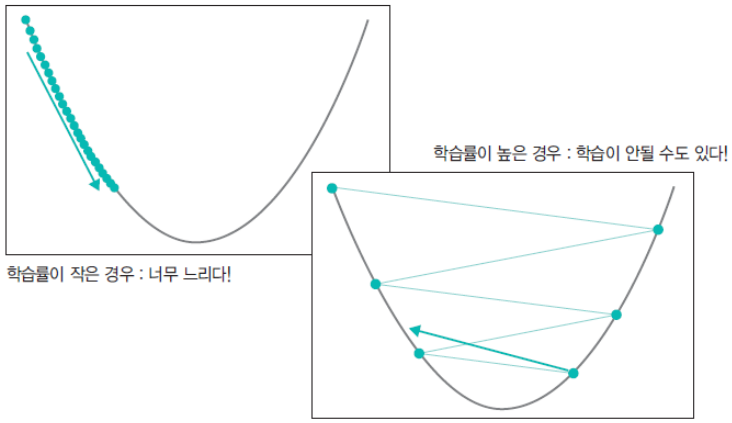
- 제공되는 코드를 통해 활성화 함수를 직접 그려보세요!

경사하강법

- **Gradient Descent Algorithm**
  - 특정 함수에서의 미분을 통해 얻은 기울기를 활용하여 최적의 값을 찾아가는 방법



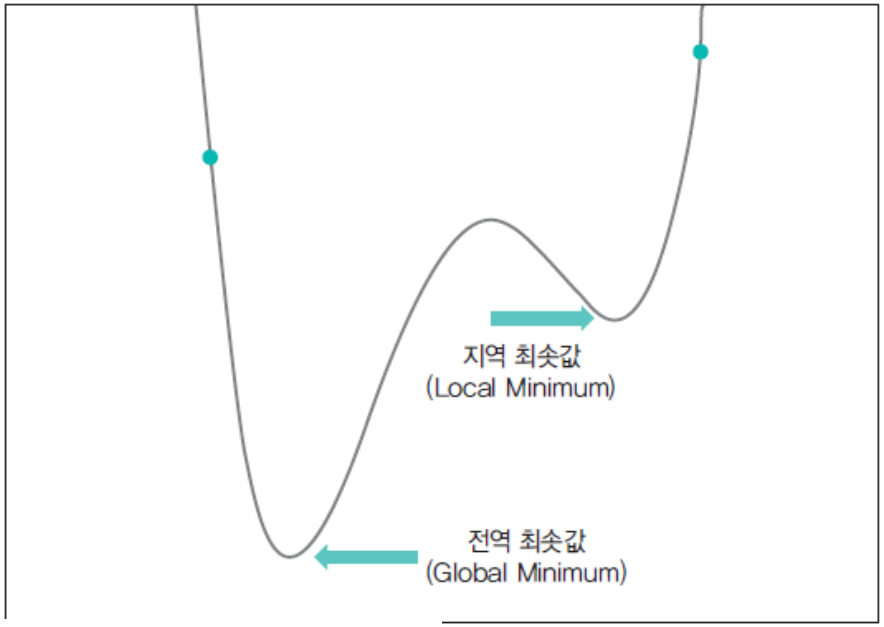
[그림 3-8] 경사하강법($y=x^2$)

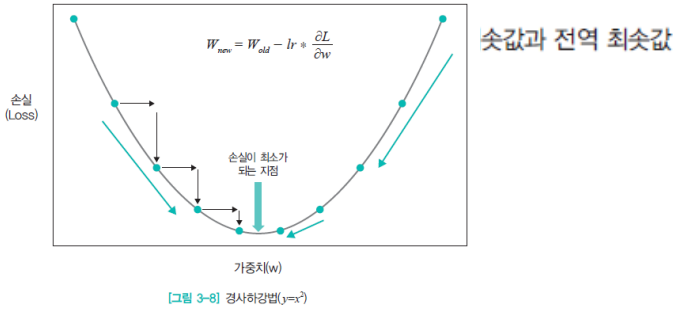


[그림 3-10] 학습에 영향을 미치는 학습률의 크기

  - 그림의 **학습률(lr, learning rate)**은 성능, 학습 속도에 중요한 영향을 끼치는 하이퍼파라미터
    - 학습률이 너무 높으면 학습이 되지 않을 수 있음
    - 그렇다고 너무 낮으면, 최적값에 도달하기 전에 학습이 종료
    - 주로 **1e-3(0.001, 또는 1e-4)**을 기본값으로 사용
  - 위 함수는 어느 지점에서 출발해도 경사를 따라가다 보면 최적값(손실이 최소가 되는 지점)에 도달함
  - 하지만 우리가 만날 함수 공간(space)은?

경사하강법





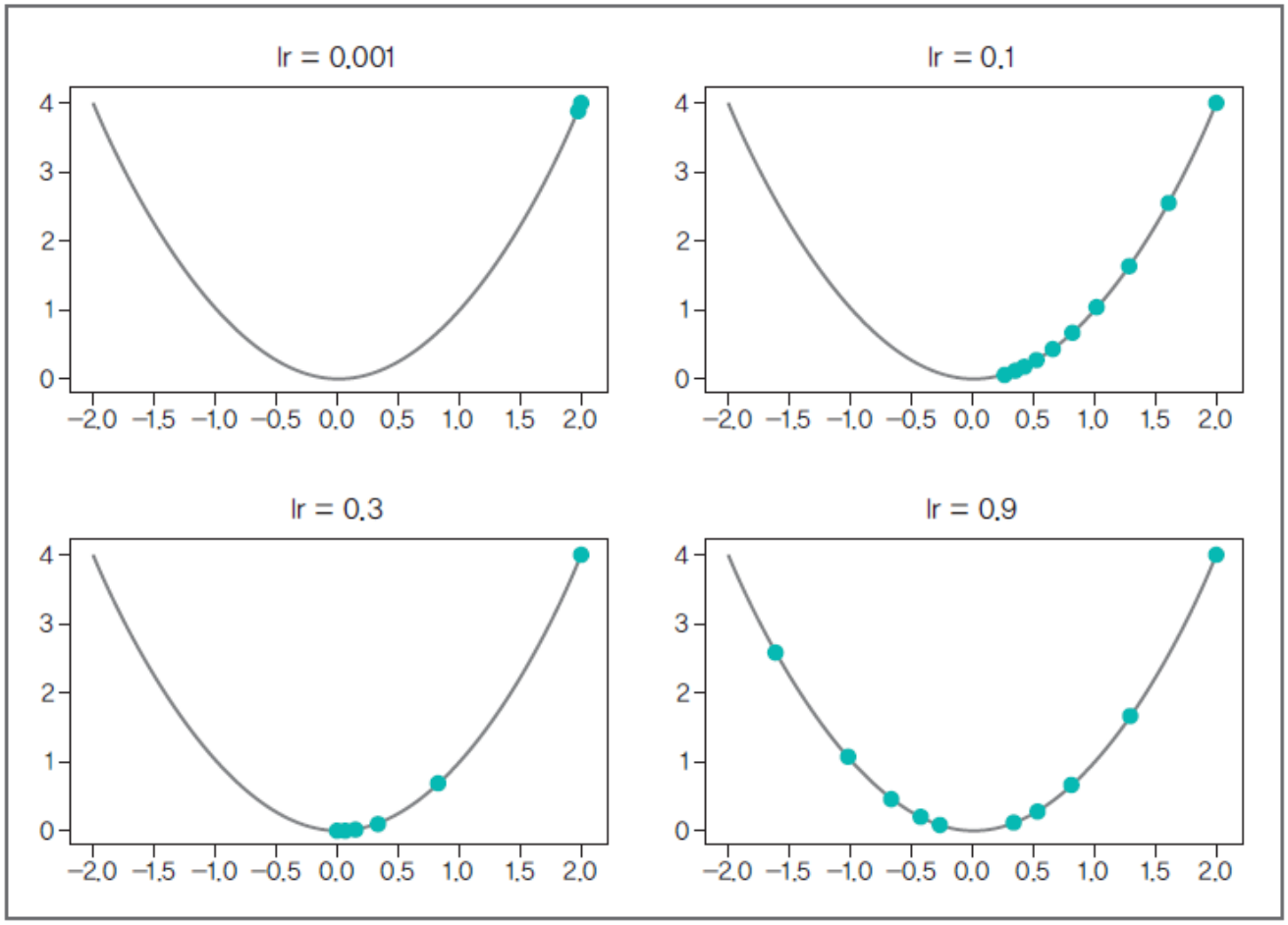
[그림 3-8] 경사하강법($y=x^2$)

숏값과 전역 최솟값

- **경사하강법은 항상 최적값을 반환한다는 보장을 할 수 없음**
  - 왼쪽 점에서 시작할 경우
    - 전역 최솟값 ← Good!!
  - 오른쪽 점에서 시작할 경우
    - 지역 최솟값 ← Bad!!
- **가중치 초기화 문제**
  - weight initialization
  - 왼쪽 점? 오른쪽 점?
  - 특별한 경우가 아닌 이상, 케라스가 제공하는 기본값(default)을 사용해도 무방
  - Glorot(Xavier), he, Lecun 초기화
- 배치 단위를 사용하여 진행
  - **확률적 경사 하강법**
  - SGD; Stochastic Gradient Descent

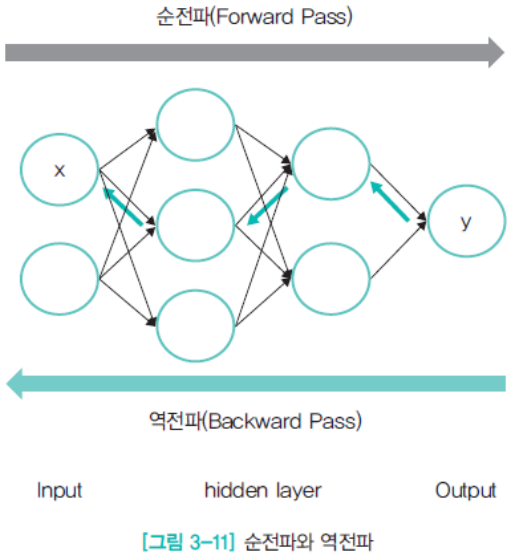
경사하강법(실습)

- 제공되는 코드를 통해 학습률을 변경하면서 경사하강법을 실행해보세요!



역전파

- 신경망을 학습시킬 멋진 방법
  - 역전파 알고리즘, Backpropagation Algorithm
  - **가중치를 무작위로 설정한 모델에서 결괏값을 도출하고, 이를 정답과 비교하여 가중치를 재조정하는 과정에서 사용**
  - ex) 햄버거 만드는 과정 → 순전파(Forward-Pass),
    햄버거에 고객의 피드백을 반영 → 역전파(Backward-Pass)



[그림 3-11] 순전파와 역전파

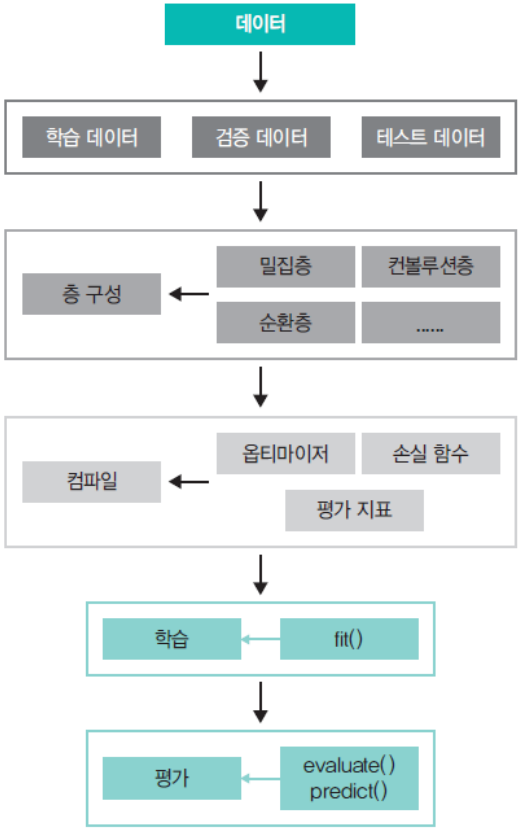
- 효율적인 계산을 위한 **체인룰(Chain-Rule)**을 사용
  - 미분 개념이 필요함

$$\frac{\partial f}{\partial x} = \frac{\partial f(g)}{\partial g} \cdot \frac{\partial g(x)}{\partial x}$$

  - 하지만 딥러닝 라이브러리가 다 해주므로,
    '신경망의 가중치가 역전파를 통해 업데이트되는구나!'를 파악!

케라스에서의 개발 과정

1. 학습 데이터를 정의합니다.
2. 데이터에 적합한 모델을 정의합니다.
3. 손실 함수, 옵티마이저, 평가지표를 선택하여 학습 과정을 설정합니다.
4. 모델을 학습시킵니다.
5. 모델을 평가합니다.



[그림 3-12] 케라스에서의 개발 과정

케라스에서의 개발 과정

- 데이터 준비
  - 학습, 검증, 테스트 데이터로 분리

- 모델 구성
  - Sequential(), Functional API(7장) 방법

예시: Sequential()을 사용한 모델 구성

```
01  model = Sequential()
02  model.add(Dense(32, input_shape = (2, ), activation = 'relu'))
03  model.add(Dense(1, activation = 'sigmoid'))
```

  - 항상 모델의 첫 번째 층은 데이터의 형태(위 코드에서 input_shape 인자)를 전달해주어야 함

케라스에서의 개발 과정

- **compile()** 함수를 통한 학습 과정 설정

예시: model.compile()

```
01  # 평균 제곱 오차 회귀 문제
02  model.compile(optimizer = RMSprop(),
03                loss = 'mse',
04                metrics = [ ])
05
06  # 이항 분류 문제
07  model.compile(optimizer = RMSprop(),
08                loss ='binary_crossentropy',
09                metrics = ['acc'])
10
11  # 다항 분류 문제
12  model.compile(optimizer = RMSprop(),
13                loss ='categorical_crossentropy',
14                metrics = ['acc'])
```

  - **옵티마이저(optimizer)** : 최적화 방법을 설정, SGD(), RMSProp(), Adam(), NAdam() 등
    - 'sgd', 'rmsprop', 'adam'과 같이 문자열로 지정하여 사용 가능
    - tf.keras.optimizers
  - **손실 함수(loss function)** : 학습 과정에서 최적화시켜야 할 손실 함수를 설정
    - mse(mean_squared_error), binary_crossentropy, categorical_crossentropy
    - tf.keras.losses
  - **평가 지표(metrics)**: 학습 과정을 모니터링하기 위해 설정
    - tf.keras.metrics

케라스에서의 개발 과정

- **fit()** 함수를 통한 모델 학습

예시: model.fit()

```
01  model.fit(data, label, epochs = 100)
02
03  model.fit(data, label, epochs = 100, validation_data = (val_data, val_label))
```

  - 에폭(epochs): 전체 학습 데이터를 몇 회 반복할지 결정
  - 배치 크기(batch_size): 전달한 배치 크기만큼 학습 데이터를 나누어 학습을 진행
  - 검증 데이터(validation_data): 모델 성능을 모니터링하기 위해 사용

- 평가 진행
  - evaluate(), predict()

예시: model.evaluate(), model.predict()

```
01  model.evaluate(data, label)
```

```
[0.21061016619205475, 1.0] 손실과 평가지표
```

```
01  result = model.predict(data)
02  print(result)
```

```
array([[0.48656905],
       [0.5464304 ],
       [0.552116  ],
       [0.4465039 ]], dtype=float32)
```

요약 정리

1. 텐서플로우 2.x는 기본적으로 **즉시 실행 모드**를 사용하며, **텐서**라는 개념을 통해 연산을 수행합니다.
2. **@tf.function**은 파이썬 함수를 텐서플로우 그래프 모드로 변경해줍니다.
3. 신경망은 **퍼셉트론 알고리즘**에서부터 출발합니다. **다층 퍼셉트론**을 사용하면 XOR 게이트 문제를 해결할 수 있습니다.
4. **경사하강법과 역전파**는 학습을 위해 사용되는 주요 개념입니다. 경사하강법에서는 **학습률, 가중치 초기화**에 대해 알아보았으며, **역전파**에서는 **체인 룰**을 사용하는 것을 배웠습니다.
5. 케라스에서의 개발 과정은 **[데이터 정의 → 모델 정의 → 손실함수, 옵티마이저, 평가지표 선택 → 모델 학습]**으로 이루어집니다.
6. 케라스 모델의 **첫 번째 층은 항상 입력 데이터의 형태를 전달**해주어야 합니다.
7. 대표적으로 손실 함수에는 **['mse', 'binary_crossentropy', 'categorical_crossentropy']**, 옵티마이저에는 **['sgd', 'rmsprop', 'adam']**이 있으며, 문자열로 지정하여 사용할 수 있습니다.