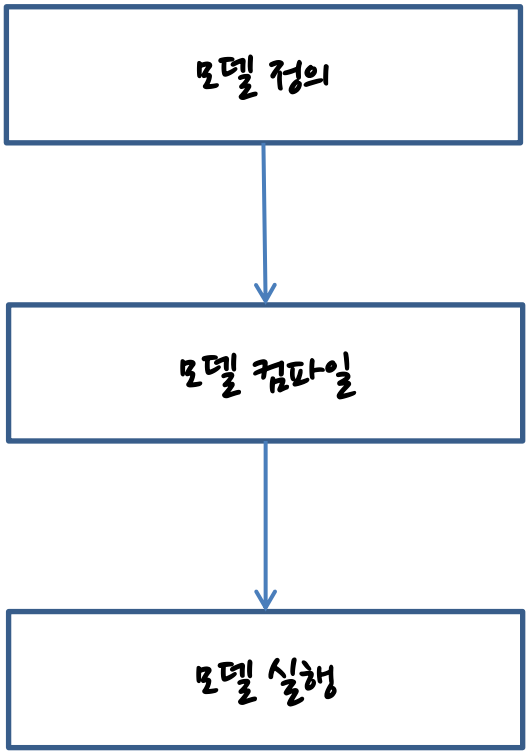
심층 신경망

2022.05

1. 신경망 Model

❖ 과정



- model이라는 함수를 선언하며 시작
- 입력층, 은닉층, 출력층 설계
- 각 층마다 활성화 함수를 설정

- 앞서 지정한 모델이 효과적으로 구현될 수 있게 여러 가지 환경을 설정해 주는 과정
- 오차 함수, 최적화 함수 설정

- 반복 횟수(Epoch) 설정
- 한번에 처리할 샘플 개수(Batch size) 설정

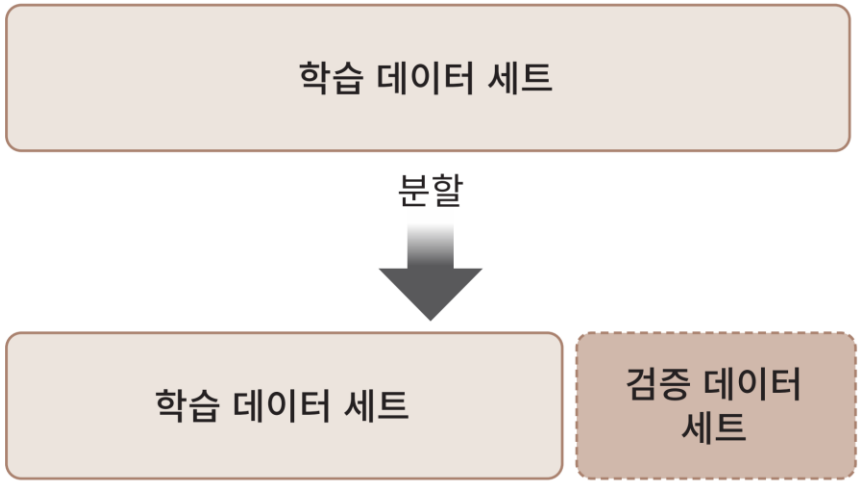
1. 신경망 Model

❖ 데이터 전처리 - 데이터셋 분할

- 데이터의 편중을 막기 위함
- 과적합을 피하고 성능 저하를 막기 위함

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, stratify=cancer.target,
    test_size=0.2, random_state=2022
)
```

학습 데이터를 다시 분할하여 학습 데이터와 학습된 모델의 성능을 일차 평가하는 검증 데이터로 나눔



모든 학습/검증 과정이 완료된 후 최종적으로 성능을 평가하기 위한 데이터 세트

1. 신경망 Model

❖ 데이터 전처리 - 피쳐 스케일링과 정규화

- 서로 다른 변수의 값 범위를 일정한 수준으로 맞추는 작업
- 표준화(Standardization)
  - 평균이 0이고, 표준편차가 1인 가우시안 정규분포로 변환
  - StandardScaler

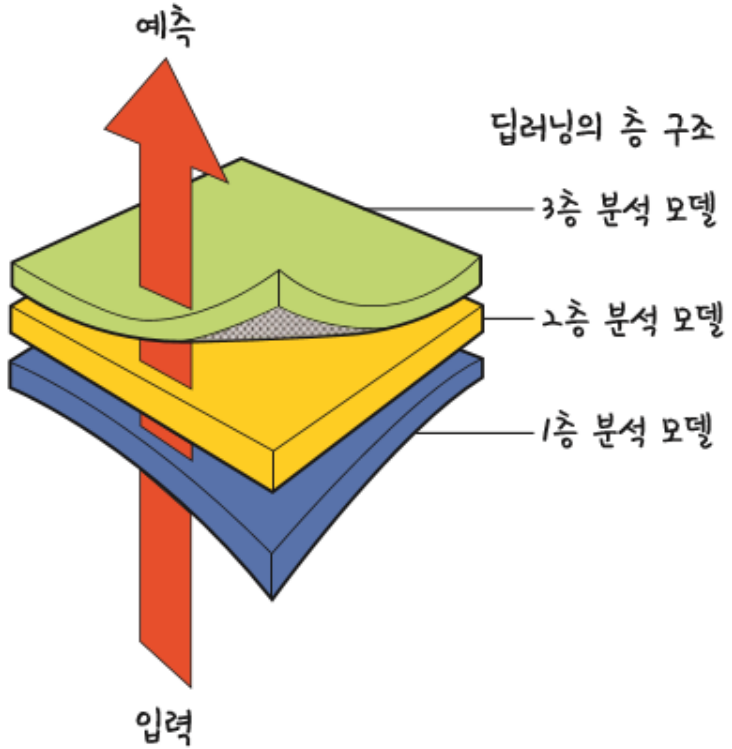
- 정규화(Normalization)
  - 최소 0 ~ 최대 1 값으로 변환
  - MinMaxScaler

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
cancer_std = scaler.fit_transform(cancer.data)
```

1. 신경망 Model

❖ 모델 정의

- 딥러닝은 아래 그림과 같이 여러 층이 쌓여 결과를 만들어 냄
- Sequential 함수는 딥러닝의 구조를 한 층 한 층 쉽게 쌓아올릴 수 있게 해 줌
- Sequential 함수를 선언할 때 필요한 층을 차례로 추가하면 됨



1. 신경망 Model

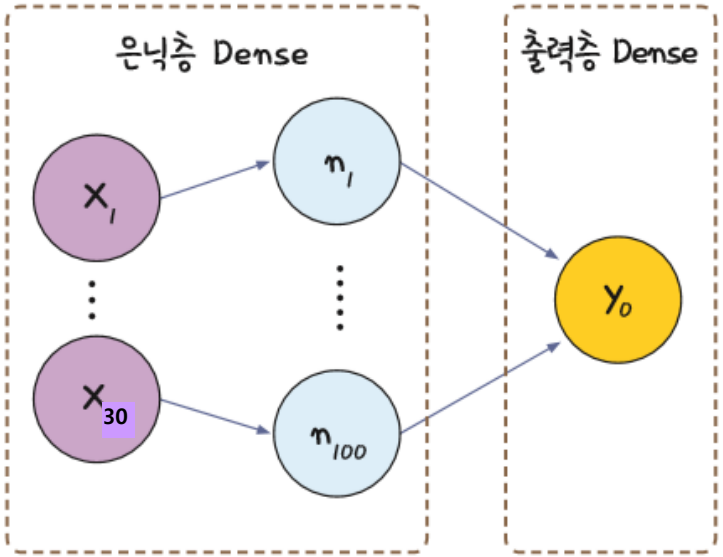
❖ 모델 정의(코드)

- Keras 라이브러리 임포트

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense              # Fully connected layer
```

- 딥러닝 모델 정의

```
model = Sequential([
    Dense(100, input_shape=(30,), activation='relu'),
    Dense(24, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

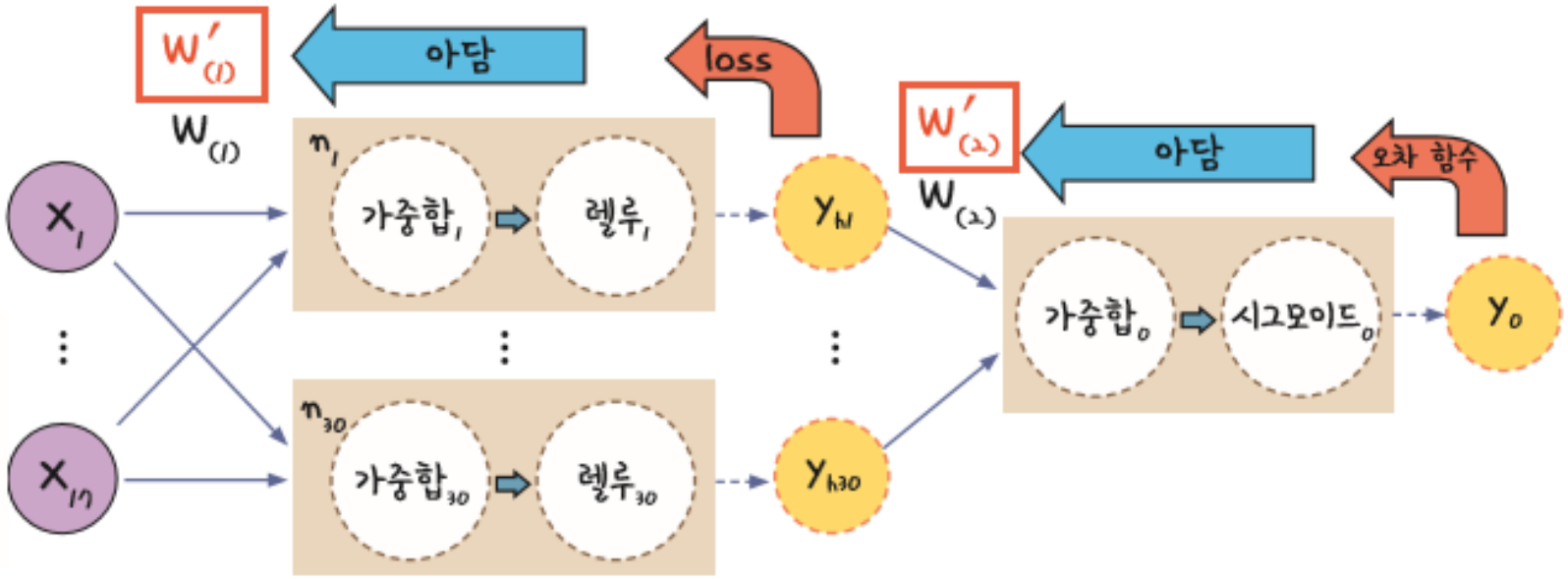


1. 신경망 Model

❖ 모델 컴파일

- 지정한 모델이 효과적으로 구현될 수 있게 여러 가지 환경을 설정

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



1. 신경망 Model

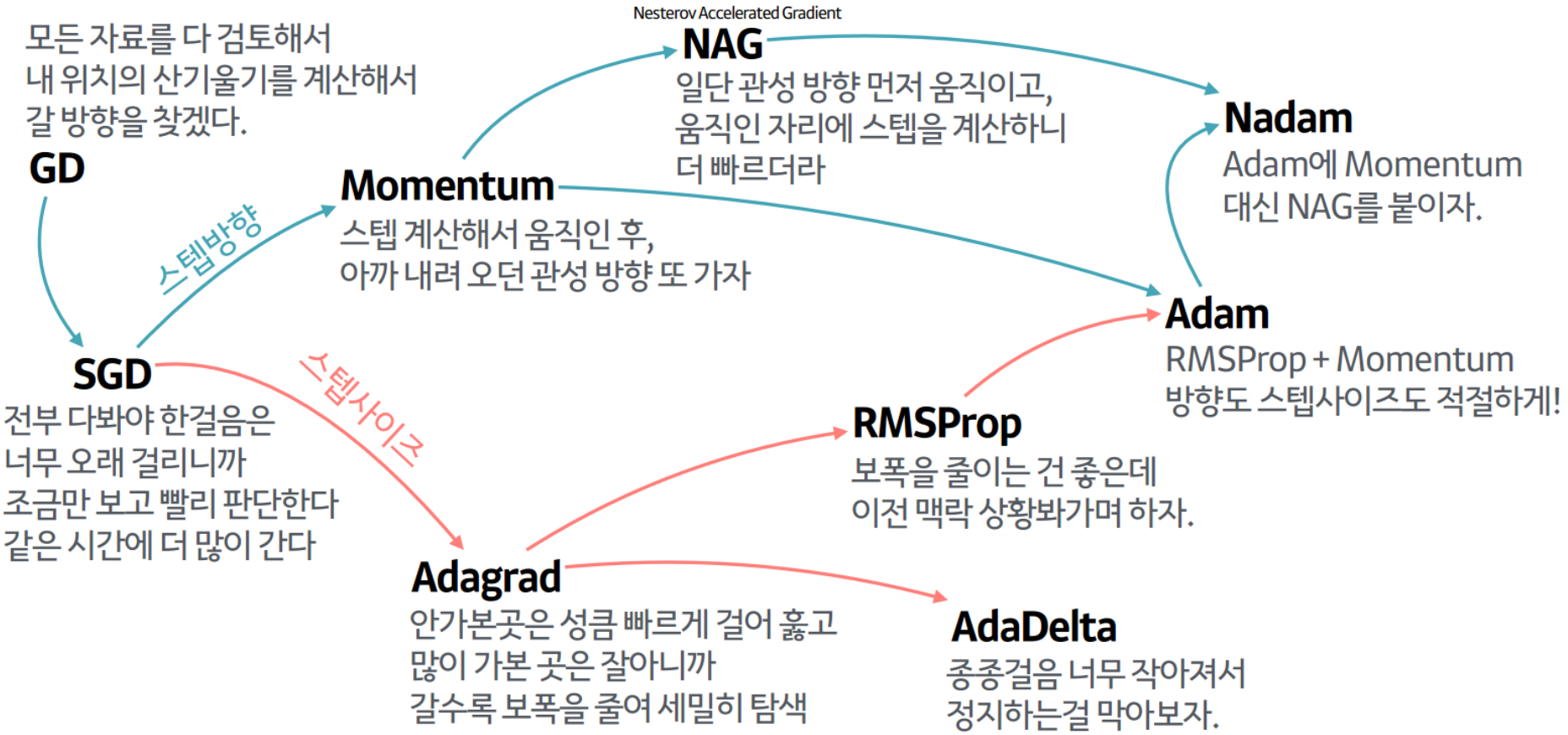
❖ 오차 함수

* 실제 값을 yt, 예측 값을 yo라고 가정할 때

평균 제곱 계열	mean_squared_error	평균 제곱 오차 계산: mean(square(yt − yo))
	mean_absolute_error	평균 절대 오차(실제 값과 예측 값 차이의 절댓값 평균) 계산: mean(abs(yt − yo))
	mean_absolute_percentage_error	평균 절대 백분율 오차(절댓값 오차를 절댓값으로 나눈 후 평균) 계산: mean(abs(yt − yo)/abs(yt)) (단, 분모 ≠ 0)
	mean_squared_logarithmic_error	평균 제곱 로그 오차(실제 값과 예측 값에 로그를 적용한 값의 차이를 제곱한 값의 평균) 계산: mean(square((log(yo) + 1) − (log(yt) + 1)))
교차 엔트로피 계열	categorical_crossentropy	범주형 교차 엔트로피(일반적인 분류)
	binary_crossentropy	이항 교차 엔트로피(두 개의 클래스 중에서 예측할 때)

1. 신경망 Model

❖ Optimizer



1. 신경망 Model

❖ 모델 실행

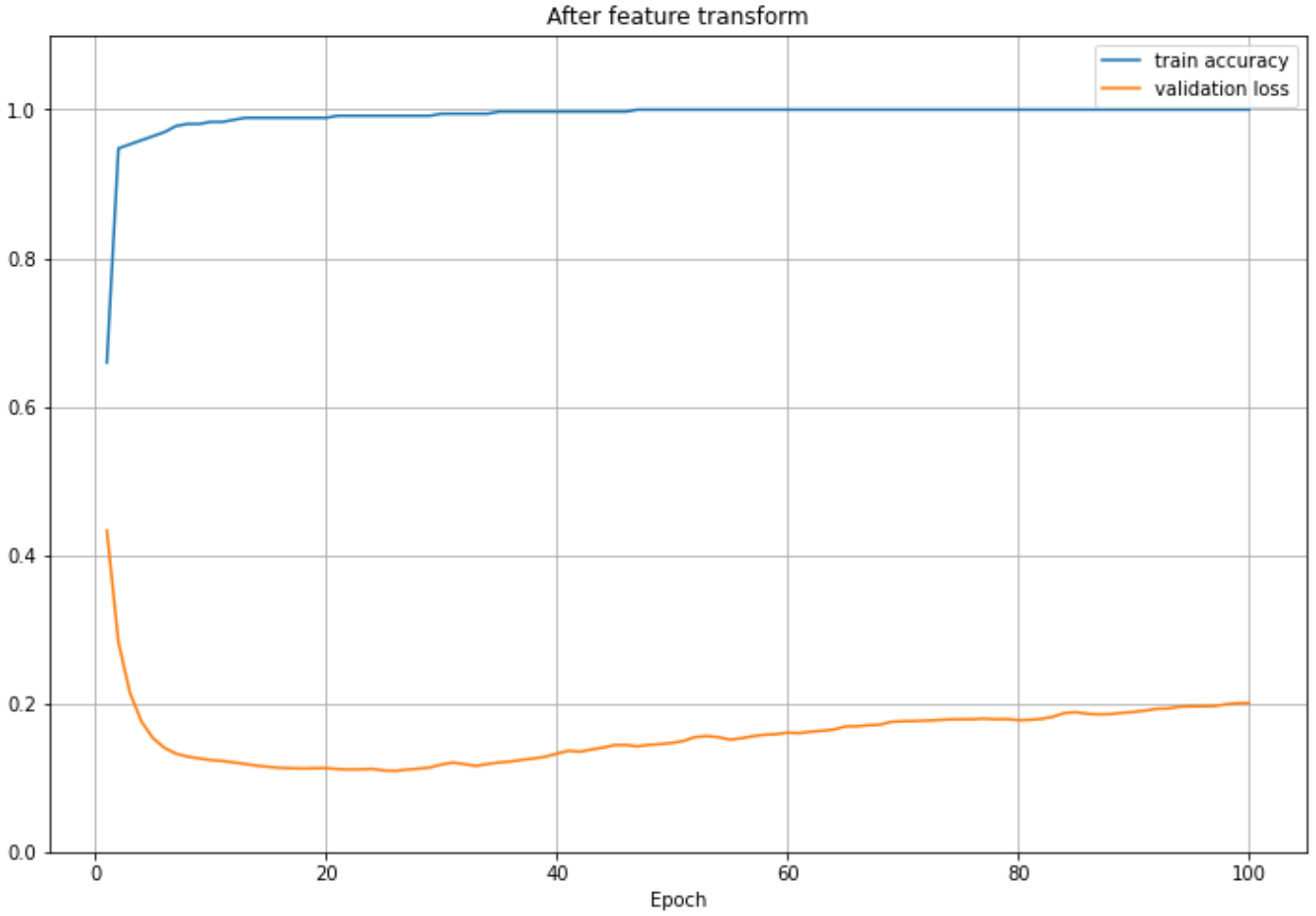
```
model.fit(X, Y, epochs=20, batch_size=10)
```

- 에포크(epoch): 학습 프로세스가 모든 샘플에 대해 한 번 실행하는 것
- batch_size: 샘플을 한 번에 몇 개씩 처리할지를 정하는 부분
  - batch_size가 너무 크면 학습 속도가 느려지고,
    너무 작으면 각 실행 값의 편차가 생겨서 전체 결과값이 불안정해질 수 있음
  - 자신의 컴퓨터 메모리가 감당할 만큼의 batch_size를 찾아 설정

1. 신경망 Model

❖ 모델 평가 및 학습과정 시각화

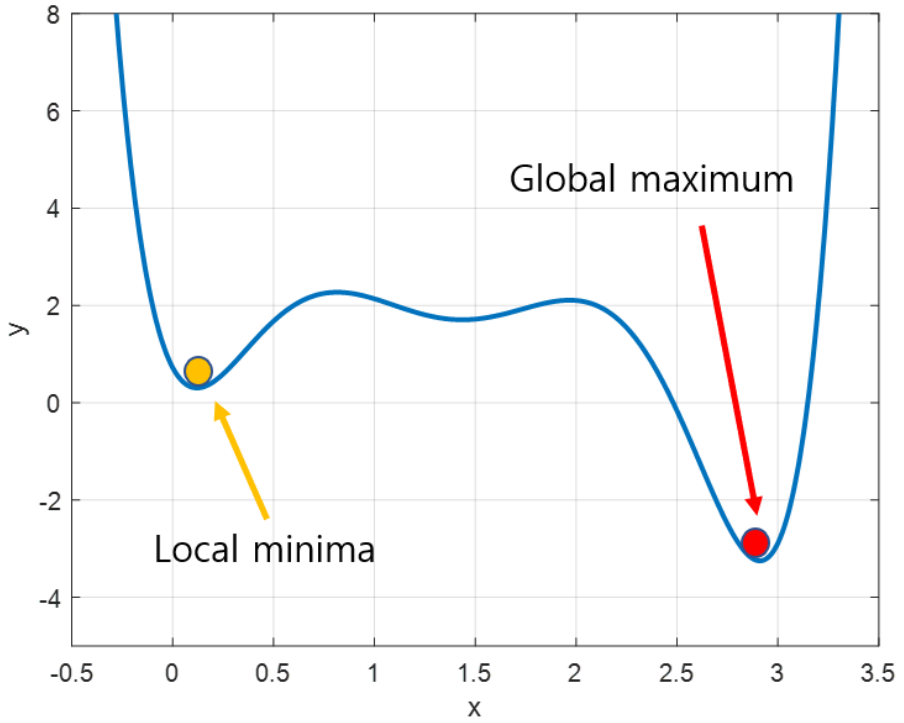
```
model.evaluate(X, Y)
```



❖ 경사 하강법의 문제

1. 계산량 증가
   - 학습용 데이터가 많아진다면 당연히 계산량도 매우 많아지고 이로 인해 학습 속도가 매우 느려지게 됨

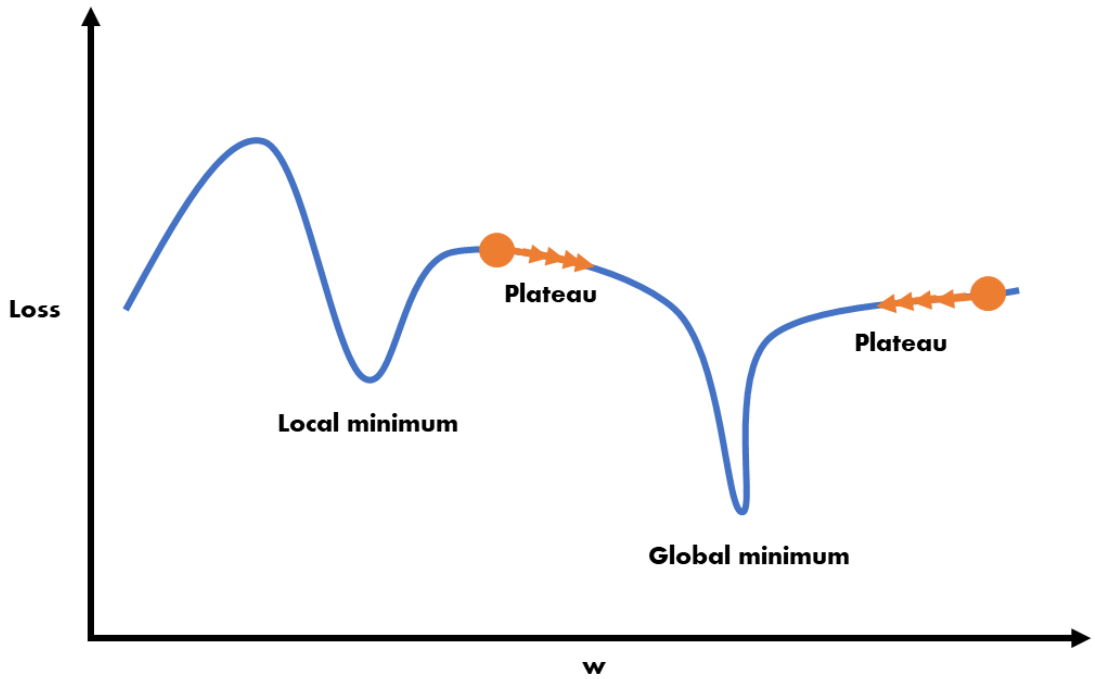
2. Local minimum(Optima) 문제
   - 실제 손실함수의 모양은 그림과 같이 울퉁불퉁한 정도가 심함
   - 랜덤하게 선택된 가중치가 Local minimum 가까이에 있고, Local minimum에 수렴해버리면 최종 목표인 Global minimum을 찾지 못하는 문제 발생



❖ 경사 하강법의 문제

3. Plateau 문제

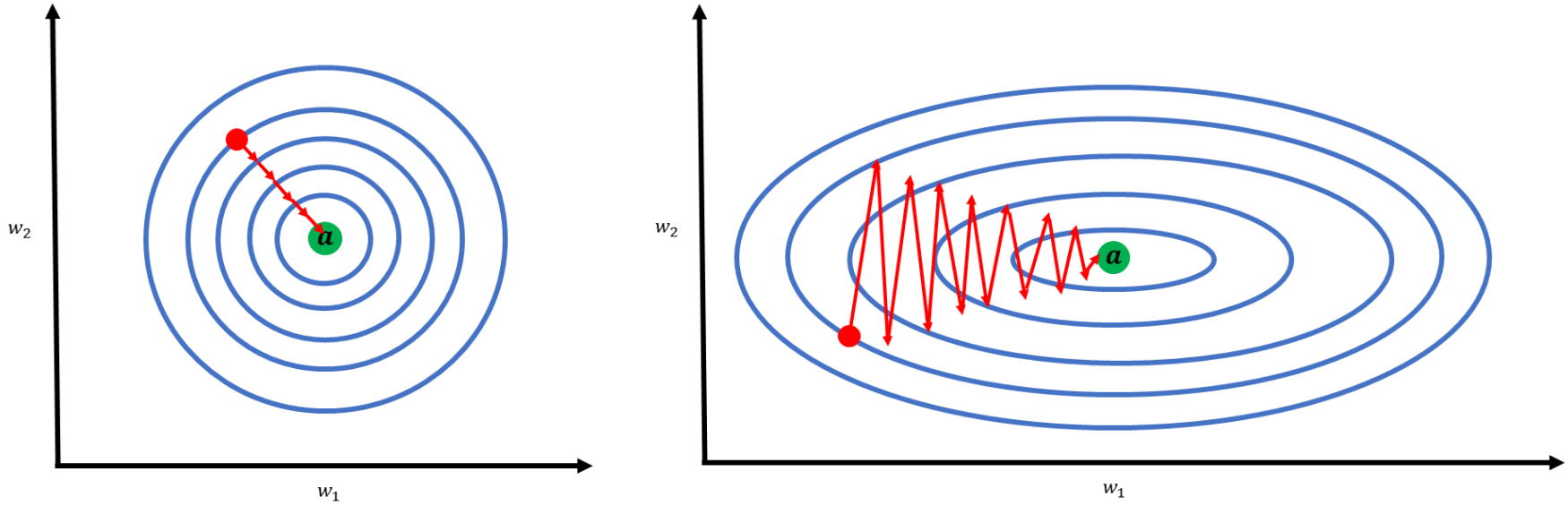
- 플래튜(Plateau)라고 불리는 평탄한 영역에서는 학습 속도가 매우 느려지며, 심지어 정지해버릴 위험이 존재함
- 가중치 소실 현상이 발생할 수 있고, 최적해를 찾는 알고리즘이 제대로 작동하지 못하게 됨



❖ 경사 하강법의 문제

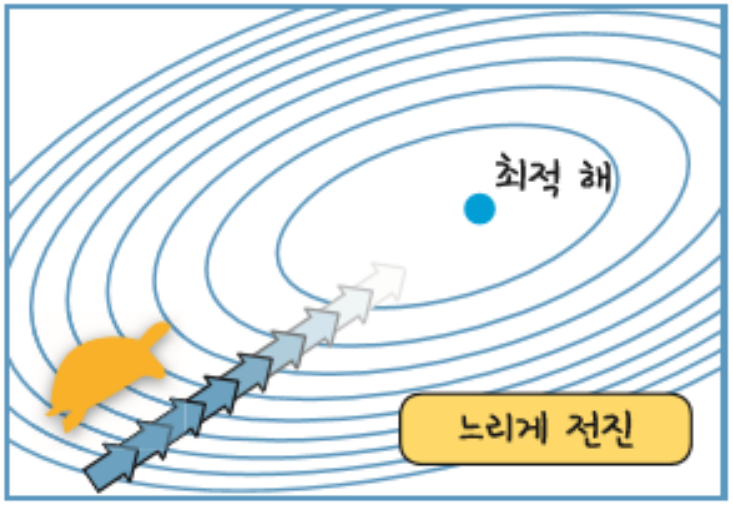
4. Zigzag 문제

- 가중치의 스케일이 동일하다면 최적해로 바로 찾아갈 수 있음
- 가중치의 스케일이 다르다면 오른쪽 그림과 같이 지그재그 현상이 발생하게 되고, 이로 인해 최적해를 찾아가기가 어려워지고 학습 시간 역시 길어지게 됨

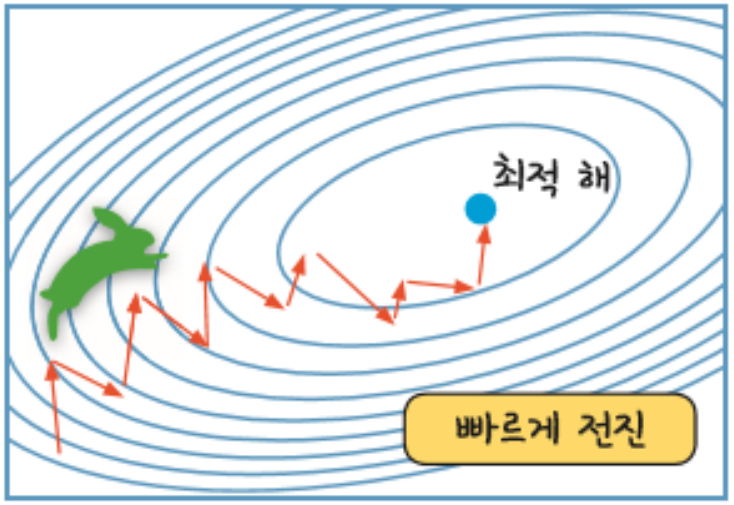


❖ 확률적 경사 하강법(SGD)

- 전체 데이터를 사용하는 것이 아니라, 랜덤하게 추출한 일부 데이터를 사용
- 일부 데이터를 사용하므로 더 빨리 그리고 자주 업데이트를 하는 것이 가능해짐



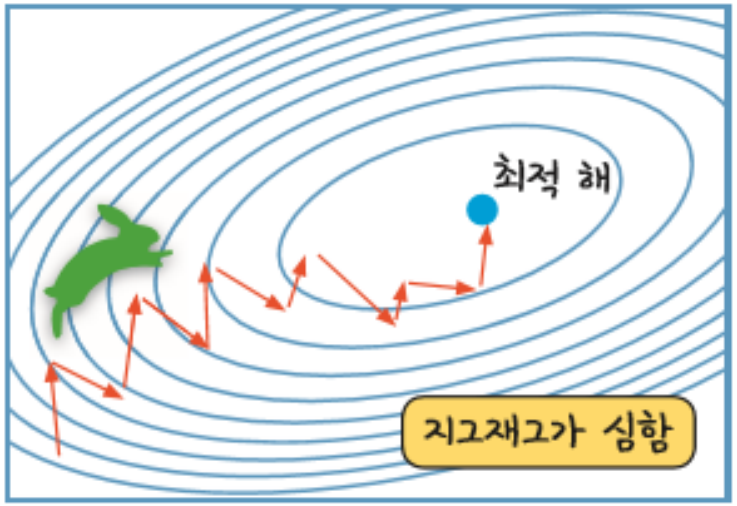
경사 하강법



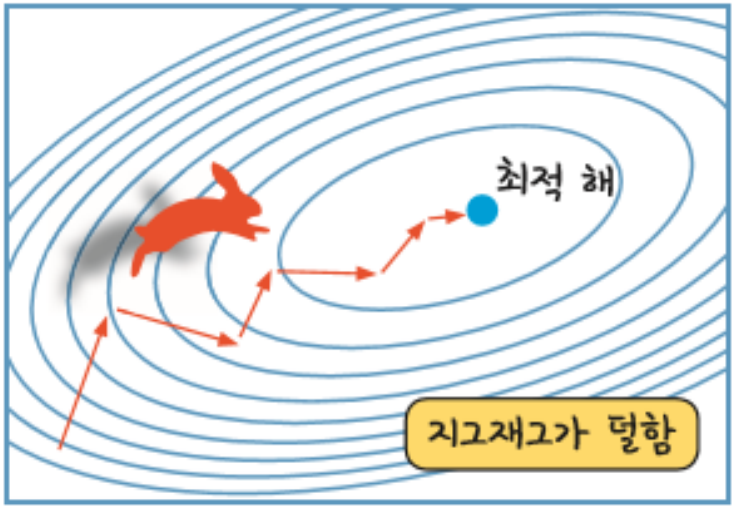
확률적 경사 하강법

❖ 모멘텀

- 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



확률적 경사 하강법



모멘텀을 적용한 확률적 경사 하강법

2. 옵티마이저

❖ 고급 경사 하강법

고급 경사 하강법	개요	효과	케라스 사용법
1. 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 이용합니다.
2. 모멘텀 (Momentum)	관성의 방향을 고려해 진동과 폭을 줄이는 효과	정확도 개선	keras.optimizers.SGD(lr = 0.1, momentum = 0.9) 모멘텀 계수를 추가합니다.
3. 네스테로프 모멘텀 (NAG)	모멘텀이 이동시킬 방향으로 미리 이동해서 그레이디언트를 계산. 불필요한 이동을 줄이는 효과	정확도 개선	keras.optimizers.SGD(lr = 0.1, momentum = 0.9, nesterov = True) 네스테로프 옵션을 추가합니다.

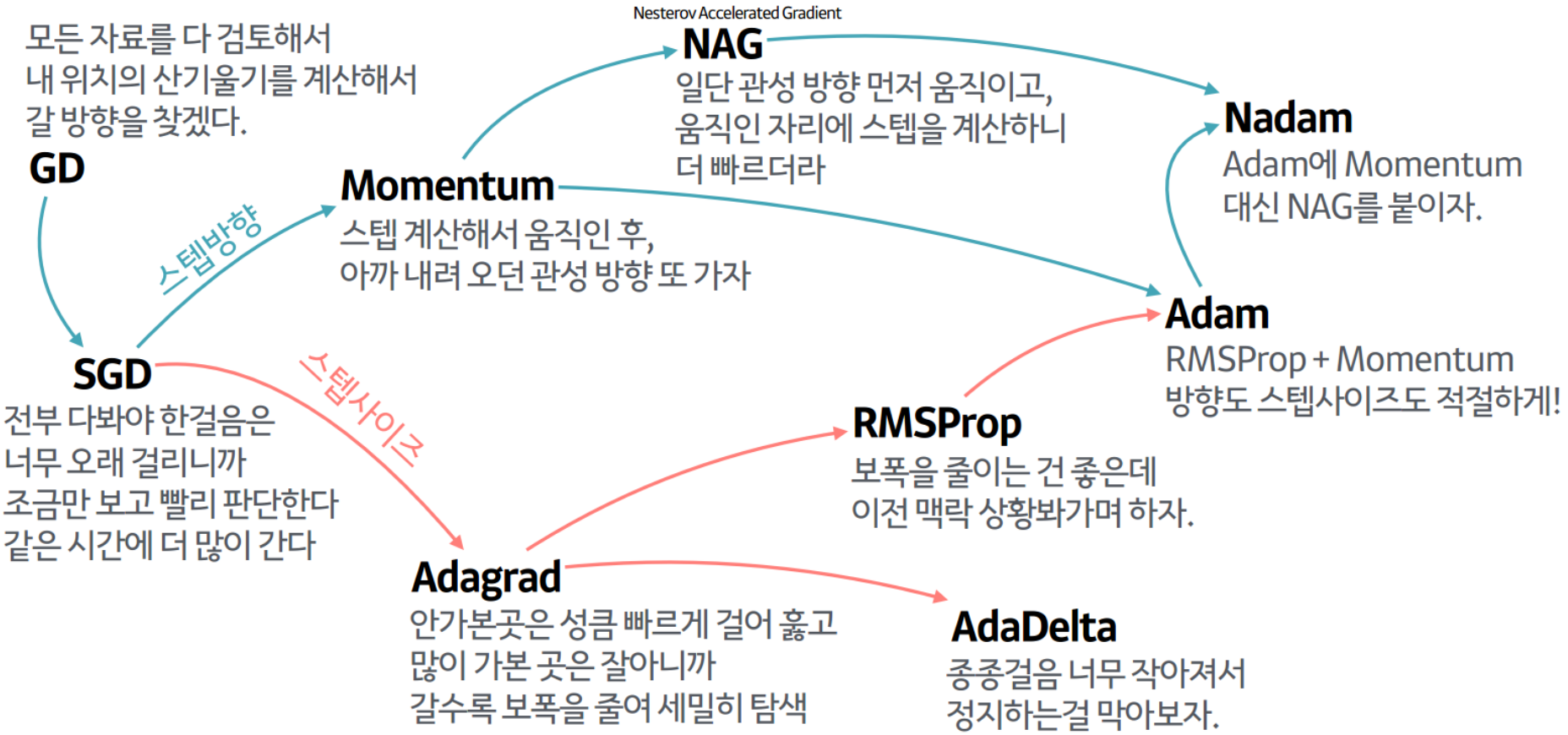
2. 옵티마이저

❖ 고급 경사 하강법

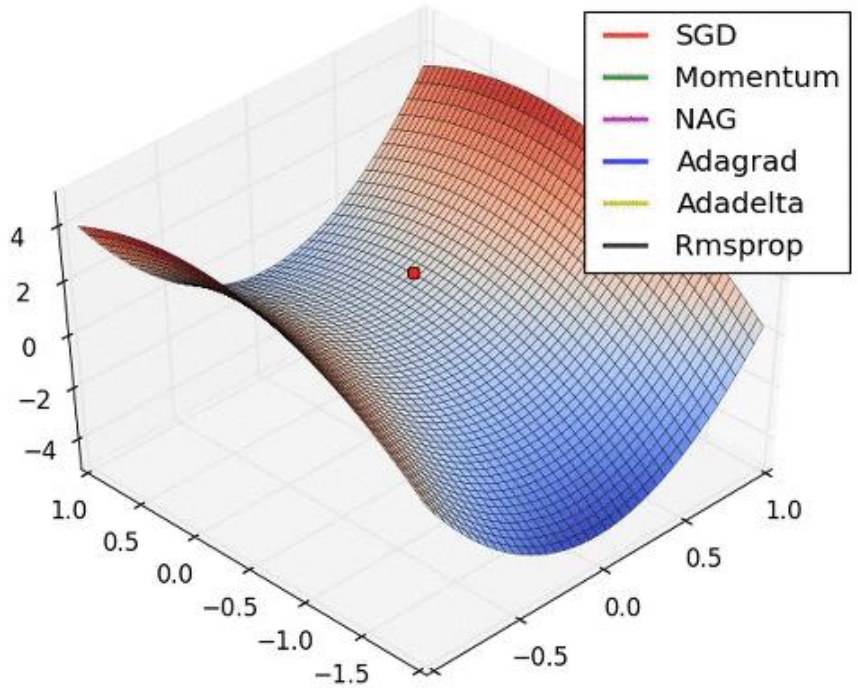
4. 아다그라드 (Adagrad)	변수의 업데이트가 잦으면 학습률을 적게 하여 이동 보폭을 조절하는 방법	보폭 크기 개선	keras.optimizers.Adagrad(lr = 0.01, epsilon = 1e − 6) 아다그라드 함수를 사용합니다. ※ 참고: 여기서 epsilon, rho, decay 같은 파라미터는 바꾸지 않고 그대로 사용하기를 권장하고 있습니다. 따라서 lr, 즉 learning rate(학습률) 값만 적절히 조절하면 됩니다.
5. 알엠에스프롭 (RMSProp)	아다그라드의 보폭 민감도를 보완한 방법	보폭 크기 개선	keras.optimizers.RMSprop(lr = 0.001, rho = 0.9, epsilon = 1e − 08, decay = 0.0) 알엠에스프롭 함수를 사용합니다.
6. 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.

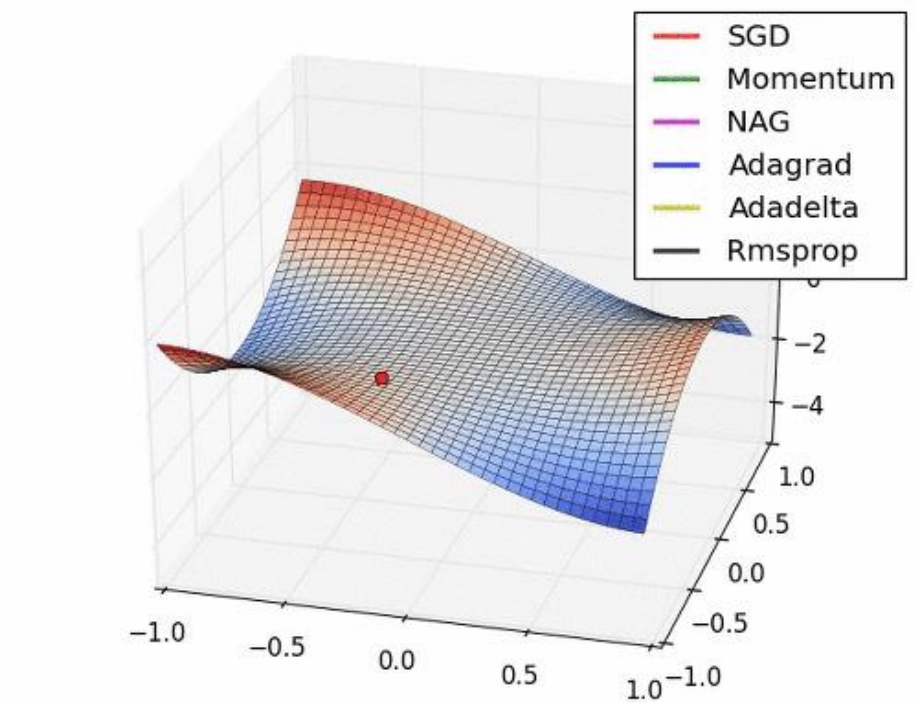
2. 옵티마이저

❖ 경사 하강법 계보



❖ 경사 하강법 계보

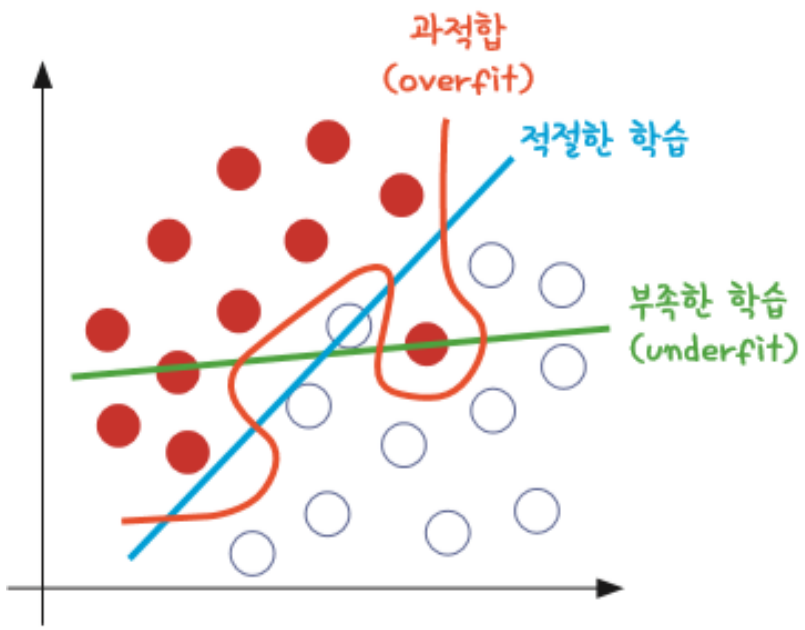

SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
4
2
0
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.5
1.0
0.5
0.0
−0.5


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0

3. 과적합 피하기

❖ 과적합(overfitting)이란?

- 모델이 학습 데이터셋 안에서는 일정 수준 이상의 예측 정확도를 보이지만, 새로운 데이터에 적용하면 잘 맞지 않는 것을 말함



3. 과적합 피하기

❖ 과적합 방지 방법

1. 데이터의 양 늘리기
   - 모델은 데이터의 양이 적을 경우, 해당 데이터의 특정 패턴이나 노이즈까지 쉽게 암기하게 되므로 과적합 현상이 발생할 확률이 늘어남
   - 데이터 증강(Data Augmentation)
     - 이미지 : 기존 이미지를 조금씩 변형(확대, 축소, 이동, 회전, 반사 등)
     - 텍스트 : 번역후 재번역 (Back Translation)

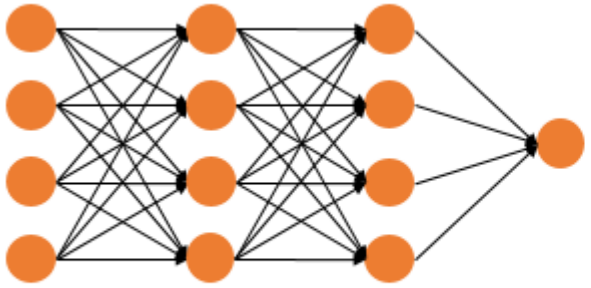
2. 모델의 복잡도 줄이기
   - 모델의 파라메터를 모델의 수용력(capacity)이라 함
   - 인공신경망 모델의 은닉층 수나 노드의 개수를 줄임

3. 과적합 피하기

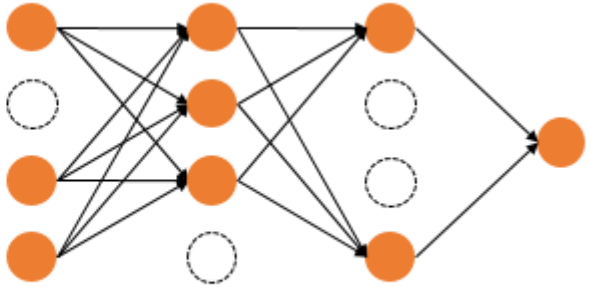
❖ 과적합 방지 방법

3. 가중치 규제(Regularization) 적용
   - L1 규제 : 가중치 w들의 절대값 합계를 비용 함수에 추가
   - L2 규제 : 모든 가중치 w들의 제곱합을 비용 함수에 추가 (가중치 감쇠: weight decay)
   - 이 두 식 모두 비용 함수를 최소화하기 위해서는 가중치 w들의 값이 작아져야 함

4. Dropout
   - 학습 과정에서 신경망의 일부를 사용하지 않는 방법



드롭아웃 적용 전

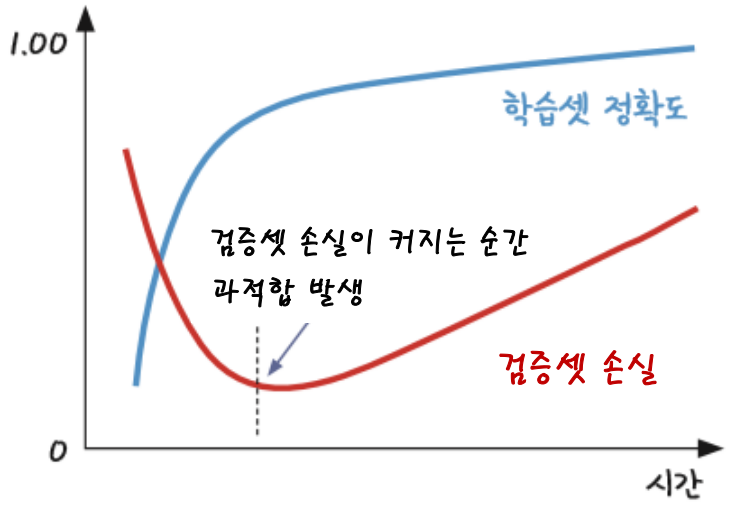


드롭아웃 적용 후

4. 베스트 모델 만들기

❖ 학습셋과 검증셋

- 학습이 깊어져서 학습셋 내부에서의 성공률은 높아져도 검증셋에서는 효과가 없다면 과적합이 일어나고 있는 것



- 학습을 진행해도 테스트 결과가 더 이상 좋아지지 않는 지점에서 학습을 멈춰야 함
- 이때의 학습 정도가 가장 적절한 것으로 볼 수 있음

4. 베스트 모델 만들기

❖ 모델 저장과 재사용

- 학습이 끝난 후 테스트해 본 결과가 만족스러울 때 이를 저장하여 사용할 수 있음
- 학습한 결과를 저장

```
from tensorflow.keras.models import load_model
model.save('my_model.h5')
```

- 불러오는 방법

```
model = load_model('my_model.h5')
```

4. 베스트 모델 만들기

❖ 모델 업데이트

- 에포크(epoch)마다 검증셋의 손실을 평가 → ModelCheckpoint 콜백(callback) 함수
- 베스트 모델(validation loss가 이전 best보다 좋아진 모델)만 저장

```
from tensorflow.keras.callbacks import ModelCheckpoint

modelpath = 'best_model.h5'
mc = ModelCheckpoint(filepath=modelpath, monitor='val_loss',
                     verbose=1, save_best_only=True)

model.fit(X_train, y_train, validation_split=0.2, epochs=200, batch_size=200,
          verbose=0, callbacks=[mc])
```

❖ 학습의 자동 중단

- 학습이 진행될수록 학습셋의 정확도는 올라가지만 과적합으로 인해 검증셋의 실험 결과는 점점 나빠지게 됨
- 학습이 진행되어도 검증셋 오차가 일정 epoch동안 줄지 않으면 학습을 멈추게 함

  → EarlyStopping 콜백 함수

```
from tensorflow.keras.callbacks import EarlyStopping

es = EarlyStopping(monitor='val_loss', patience=30)

model.fit(X_train, y_train, validation_split=0.2, epochs=1000, batch_size=500,
          verbose=0, callbacks=[es, mc])
```