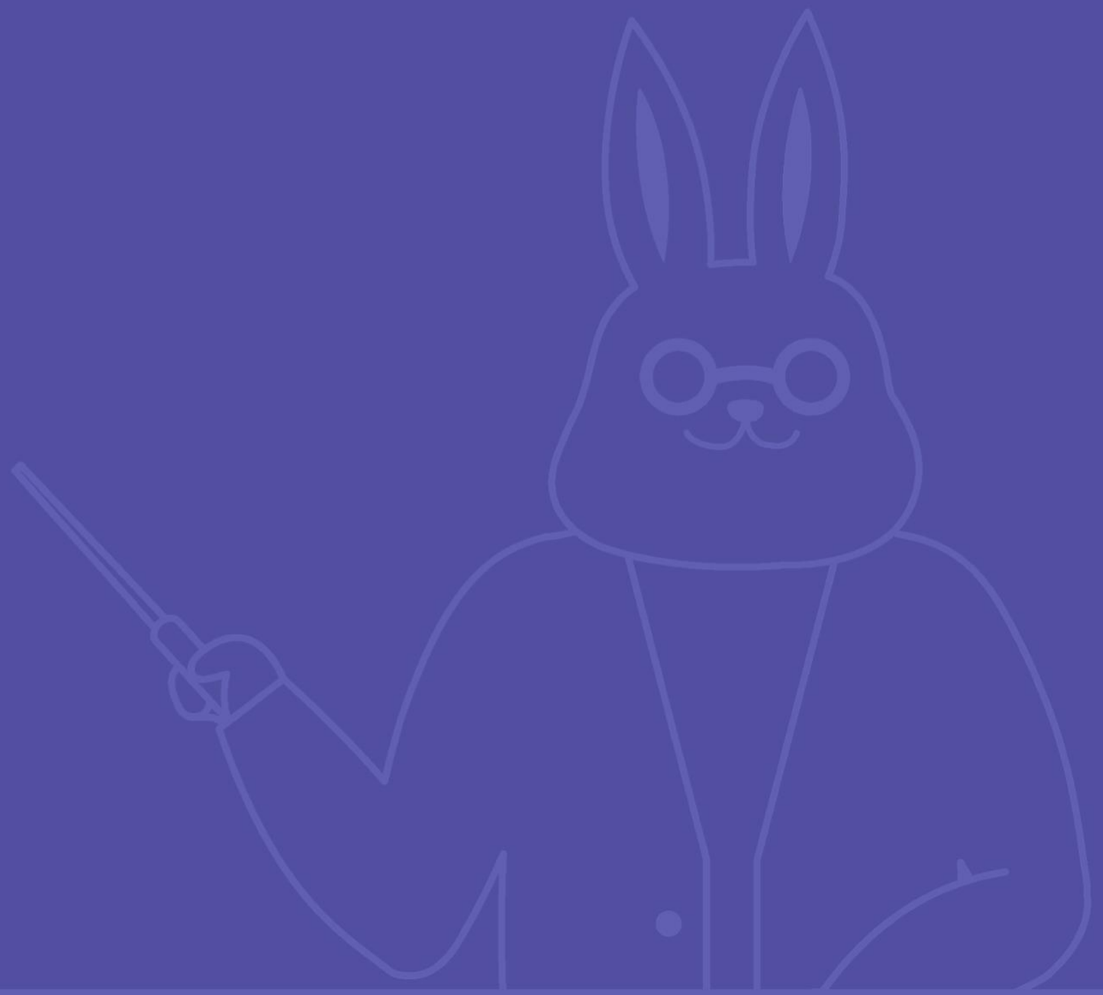
CNN/RNN 활용

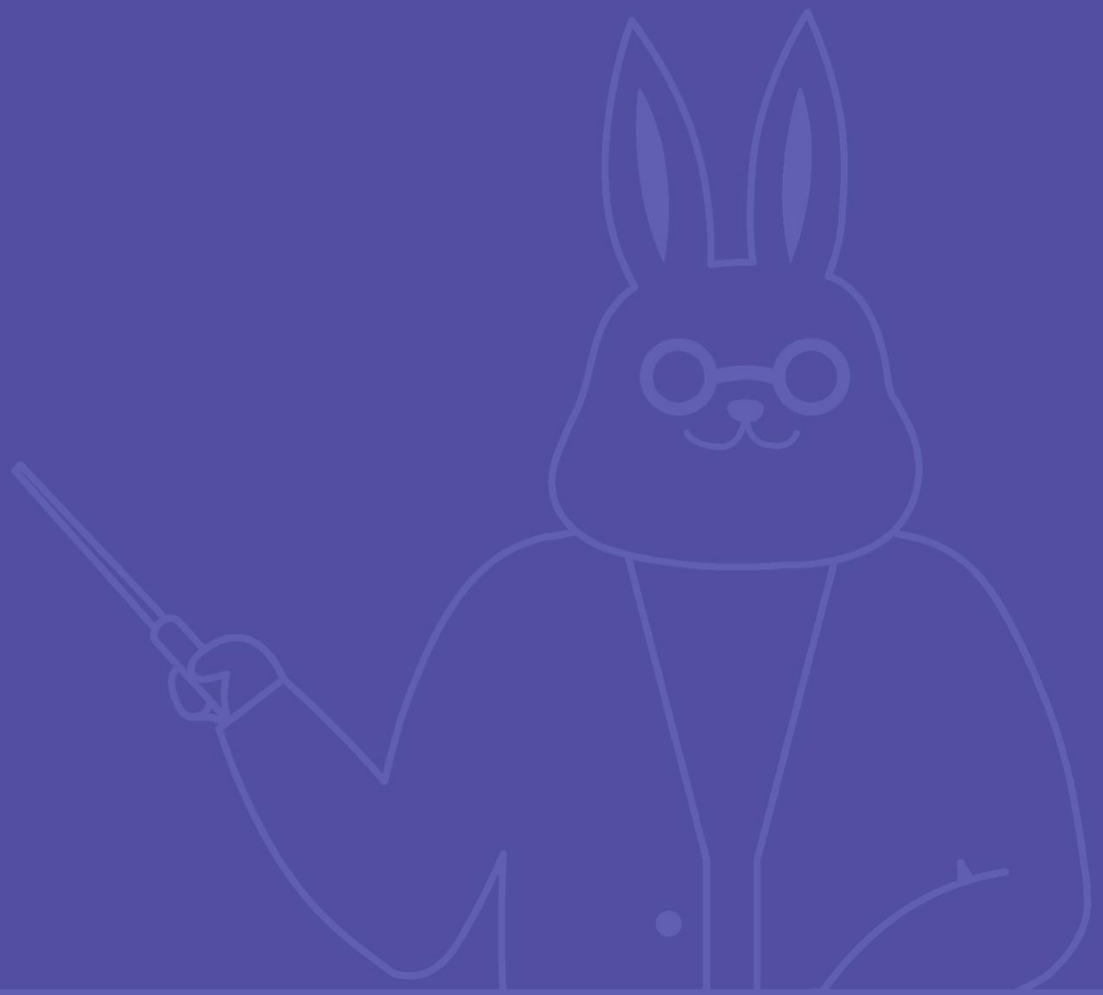
01 모델 학습



목차

01

Background



딥러닝 라이브러리의 종류

Tensorflow

- Google이 개발
- 사용이 복잡했으나 2.0이후 개선됨
- colab에서 간편하게 GPU 사용을 지원
- Tensorboard를 통한 강력한 시각화
- TF lite를 통한 간편한 모바일 변환

pytorch

- Facebook이 개발
- 일부 모델의 구현이나 데이터 처리에 더 적합
- 모델의 자세한 구조나 연산을 연구하는 용도로 적합

Tensorflow 1.x

Tensorflow

- **Session** 을 단위로 실행 코드를 작성하고 수동으로 모델을 구현
- 일부 동적인 길이의 입력 처리를 지원하지 않음
- gpu를 사용하는 버전과 gpu를 사용하지 않는 버전이 별도로 존재

Keras

- Tensorflow의 모델 구현과 학습을 편리하게 작성하기 위한 API
- layer.add 등을 통해 일반적인 구조의 모델은 간편하게 구현이 가능함

Tensorflow 2.0

- Keras를 공식 api로 채택하고 Tensorflow내에 포함시킴
- 기존 keras의 기능 대부분도 **tf.keras**를 통해 사용이 가능
- GPU 사용 여부와 관계없이 하나의 라이브러리를 유지

- 1.15 버전과 2.0이 같이 사용되던 시기
  - 일부 tensorflow 1.15의 문제점이나 버전 관리가 어려운 문제점이 있음

Tensorflow 2.7

- keras의 기능들이 안정적으로 포함됨
- 다양한 구조의 모델을 간편하게 구현 가능하고 사용자 정의 구조를 정의가 개선됨
- GPU를 사용할 수 있는 환경에서 자동으로 데이터를 GPU에 업로드하는 과정을 수행
- LSTM을 포함한 구조의 연산을 최적화함

- Google에서 제공하는 문서들이 2.7 버전으로 작성되어 있음

과적합이란 무엇인가

처음 모델학습을 진행할 때 가장 많이 마주치는 문제

모델이 데이터의 일반적인 특징이 아닌 학습 데이터에만 특화되어 학습

- 문제집 답을 외운 학생과 유사한 특징
- 외운 문제집을 풀면 100점에 가까운 점수를 달성하지만 시험을 보면 점수가 매우 떨어짐

과적합의 원인

- 데이터의 수나 다양성이 부족하다 => 문제집의 문제가 적어서 잠깐만 봐도 외울 수 있다
- 데이터에 비해 모델이 너무 **크다** => 난이도에 비해 학생의 **암기력**이 너무 좋다
- 너무 많은 epoch을 학습했다 => 같은 문제집을 너무 오래 보게 했다

과적합을 막기 위한 방법

데이터를 늘린다

- 가장 근본적인 해결 방법중 하나
- 충분히 많은 데이터를 확보한다면 과적합이 잘 일어나지 않음

과적합이 일어나기 전에 학습을 정지한다 (Early stop)

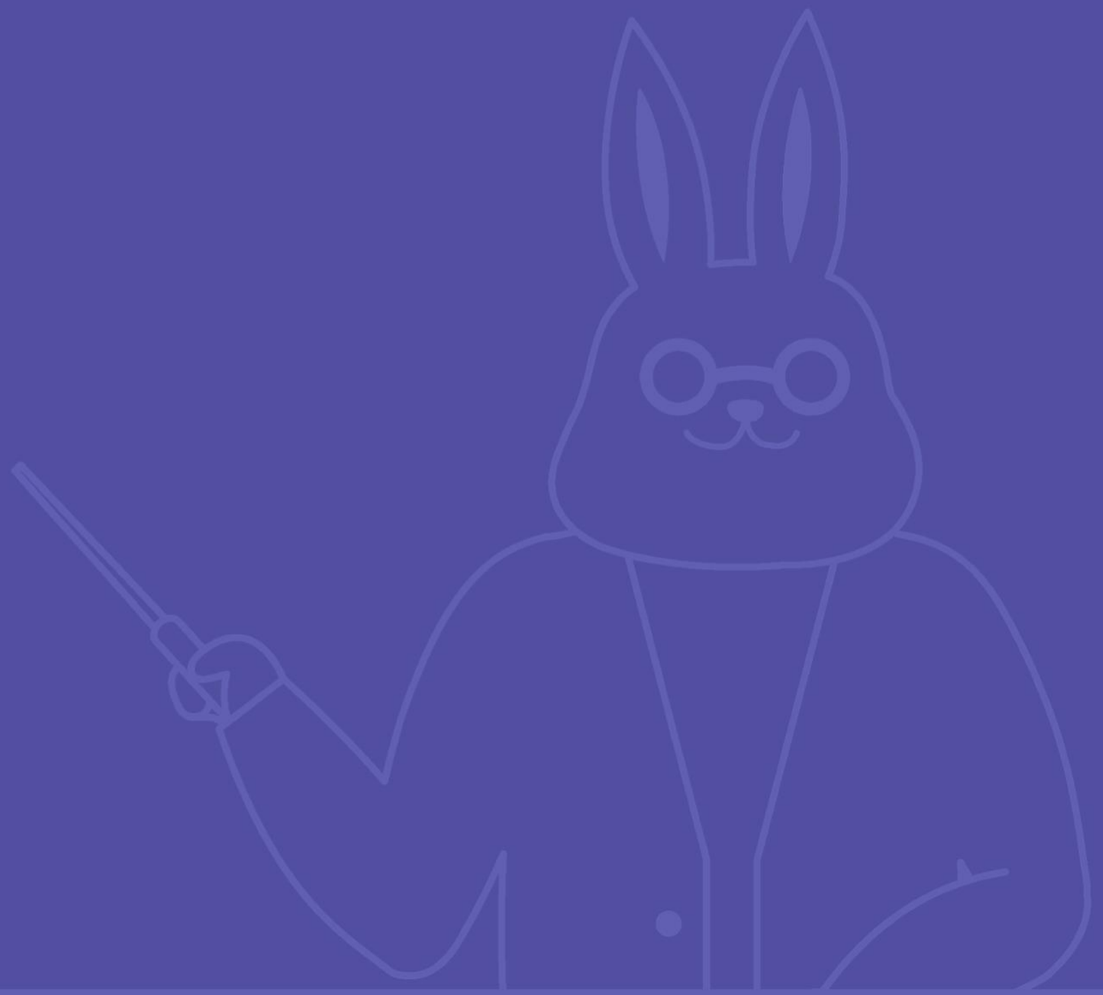
- 검증 성능이 가장 높은 시점에서 학습을 중단하고 모델을 사용

데이터 증강 (Data augmentation)

- 같은 데이터도 반전시키거나 약간 변형해서 입력
- 적은 수의 데이터를 이용하여 좀더 일반적인 특징을 학습하도록 유도

02

Fit 함수



학습과정의 핵심 fit 함수

tf.keras.Model.fit ( )

```
fit( x=None, y=None, batch_size=None, epochs=1, verbose='auto',
callbacks=None, validation_split=0.0, validation_data=None, shuffle=True,
class_weight=None, sample_weight=None, initial_epoch=0, steps_per_epoch=None,
validation_steps=None, validation_batch_size=None, validation_freq=1,
max_queue_size=10, workers=1, use_multiprocessing=False )
```

모델을 학습하는 핵심 함수

- 다양한 옵션을 제공하는 인수를 포함하고 있음
- 모델 학습을 위해서는 각 인수의 의미를 파악할 필요가 있음
- 이전 버전에서 사용하던 fit_generator 함수도 fit함수에 포함되었음

학습 데이터

tf.keras.Model.fit ( )

```
fit( x=None, y=None, batch_size=None, epochs=1, verbose='auto',
callbacks=None, validation_split=0.0, validation_data=None, shuffle=True,
class_weight=None, sample_weight=None, initial_epoch=0, steps_per_epoch=None,
validation_steps=None, validation_batch_size=None, validation_freq=1,
max_queue_size=10, workers=1, use_multiprocessing=False )
```

x

- 학습 세트의 입력 데이터

y

- Label, Ground truth
- 학습 세트의 입력 데이터를 넣었을 때 정답 값

Epoch과 batch

tf.keras.Model.fit ( )

```
fit( x=None, y=None, batch_size=None, epochs=1, verbose='auto',
callbacks=None, validation_split=0.0, validation_data=None, shuffle=True,
class_weight=None, sample_weight=None, initial_epoch=0, steps_per_epoch=None,
validation_steps=None, validation_batch_size=None, validation_freq=1,
max_queue_size=10, workers=1, use_multiprocessing=False )
```

batch_size:한번에 입력할 데이터의 수

- 모델의 input_shape도 이 값을 고려하여 설정

epochs: 학습 데이터를 몇 번 학습할 것인지

steps_per_epoch: 한 epoch은 몇 번의 입력을 수행하는지

- None으로 전달하면 batch_size에 따라 자동으로 설정됨

batch_size * steps_per_epoch = 학습 데이터의 수

학습 진행도를 출력하는 방법

tf.keras.Model.fit ( )

```
fit( x=None, y=None, batch_size=None, epochs=1, verbose='auto',
callbacks=None, validation_split=0.0, validation_data=None, shuffle=True,
class_weight=None, sample_weight=None, initial_epoch=0, steps_per_epoch=None,
validation_steps=None, validation_batch_size=None, validation_freq=1,
max_queue_size=10, workers=1, use_multiprocessing=False )
```

verbose : 학습 과정을 출력하는 방법

- 0: 어떤 출력도 하지 않는다
- 1: 진행바를 표시하여 진행상황을 표시 (기본값)

```
Epoch 17/20
219/219 [==============================] - 4s 17ms/step - loss: 0.0015
```

- 2: 진행바는 표시하지 않고 수치 정보만 표시

```
Epoch 18/20
219/219 - 1s - loss: 0.0094
```

검증 데이터

tf.keras.Model.fit ( )

```
fit( x=None, y=None, batch_size=None, epochs=1, verbose='auto',
callbacks=None, validation_split=0.0, validation_data=None, shuffle=True,
class_weight=None, sample_weight=None, initial_epoch=0, steps_per_epoch=None,
validation_steps=None, validation_batch_size=None, validation_freq=1,
max_queue_size=10, workers=1, use_multiprocessing=False )
```

validation_data : 검증 데이터 셋

- (x_val, y_val) 의 형태로 전달하거나 Generator로 전달
- 이 데이터는 학습에 반영하지는 않고 현재 epoch의 성능을 평가하기 위해 사용

validation_freq : 검증하는 주기

- 1(기본값): 매 epoch마다 검증
- 자주 수행할수록 걸리는 시간은 증가하지만, 세밀하게 성능을 평가할 수 있음

학습 데이터의 순서를 섞기

tf.keras.Model.fit ( )

```
fit( x=None, y=None, batch_size=None, epochs=1, verbose='auto',
callbacks=None, validation_split=0.0, validation_data=None, shuffle=True,
class_weight=None, sample_weight=None, initial_epoch=0, steps_per_epoch=None,
validation_steps=None, validation_batch_size=None, validation_freq=1,
max_queue_size=10, workers=1, use_multiprocessing=False )
```

shuffle: 학습 데이터를 섞을지 여부

- 정해진 학습 데이터의 입력 순서에 맞춰 학습되지 않도록 순서를 변경
- Generator를 사용할 경우 Generator의 옵션에 관련 내용이 이미 존재하므로 이 값은 무시됨

검증 과정의 epoch, batch

tf.keras.Model.fit ( )

```
fit( x=None, y=None, batch_size=None, epochs=1, verbose='auto',
callbacks=None, validation_split=0.0, validation_data=None, shuffle=True,
class_weight=None, sample_weight=None, initial_epoch=0, steps_per_epoch=None,
validation_steps=None, validation_batch_size=None, validation_freq=1,
max_queue_size=10, workers=1, use_multiprocessing=False )
```

- validation_batch_size: 한번 입력에 입력할 데이터의 수
- validation_steps: 한번 검증할 때 데이터 입력의 횟수
- validation_batch_size * validation_steps = 검증 데이터의 수

데이터 불균형을 위한 가중치

tf.keras.Model.fit ( )

```
fit( x=None, y=None, batch_size=None, epochs=1, verbose='auto',
callbacks=None, validation_split=0.0, validation_data=None, shuffle=True,
class_weight=None, sample_weight=None, initial_epoch=0, steps_per_epoch=None,
validation_steps=None, validation_batch_size=None, validation_freq=1,
max_queue_size=10, workers=1, use_multiprocessing=False )
```

class_weight: 클래스별 반영 정도

- 데이터 불균형 해결하는 방법 중 하나
- 적은 수의 클래스에는 더 많은 관심을 기울이는 개념
- 클래스별 데이터의 수가 차이날 경우, 적은 수의 클래스는 더 많이 반영하여 학습
- {클래스index:반영비율} 의 딕셔너리 문제를 형태로 전달

중간부터 이어서 학습하기

tf.keras.Model.fit ( )

```
fit( x=None, y=None, batch_size=None, epochs=1, verbose='auto',
callbacks=None, validation_split=0.0, validation_data=None, shuffle=True,
class_weight=None, sample_weight=None, initial_epoch=0 steps_per_epoch=None,
validation_steps=None, validation_batch_size=None, validation_freq=1,
max_queue_size=10, workers=1, use_multiprocessing=False )
```

initial_epoch: 학습을 시작하는 epoch

- 중간에 중단된 학습을 이어서 진행하는 경우에는 0이 아닌 값을 사용
- 복잡한 모델을 학습하는 경우 epoch에 따라 변하는 수치들이 존재

학습은 initial_epoch+1부터 epochs까지 진행

- initial_epoch=20, epochs = 40 일 때 학습하는 epoch
  - [21/40],[22/40]…[40/40] : 20번의 epoch 학습

데이터 병렬 처리

tf.keras.Model.fit ( )

```
fit( x=None, y=None, batch_size=None, epochs=1, verbose='auto',
callbacks=None, validation_split=0.0, validation_data=None, shuffle=True,
class_weight=None, sample_weight=None, initial_epoch=0, steps_per_epoch=None,
validation_steps=None, validation_batch_size=None, validation_freq=1,
max_queue_size=10, workers=1, use_multiprocessing=False )
```

workers, use_multiprocessing: 병렬 처리를 위한 인수들

- GPU를 사용할 경우 데이터를 읽고 변환하는 과정이 학습하는 시간보다 길게 소모
- GPU는 그동안 기다리는 idle time이 발생하면서 효율이 저하
- 여러 Workers가 동시에 데이터를 불러오고 학습을 진행
- Generator를 사용하는 경우에만 사용

콜백함수의 리스트

tf.keras.Model.fit ( )

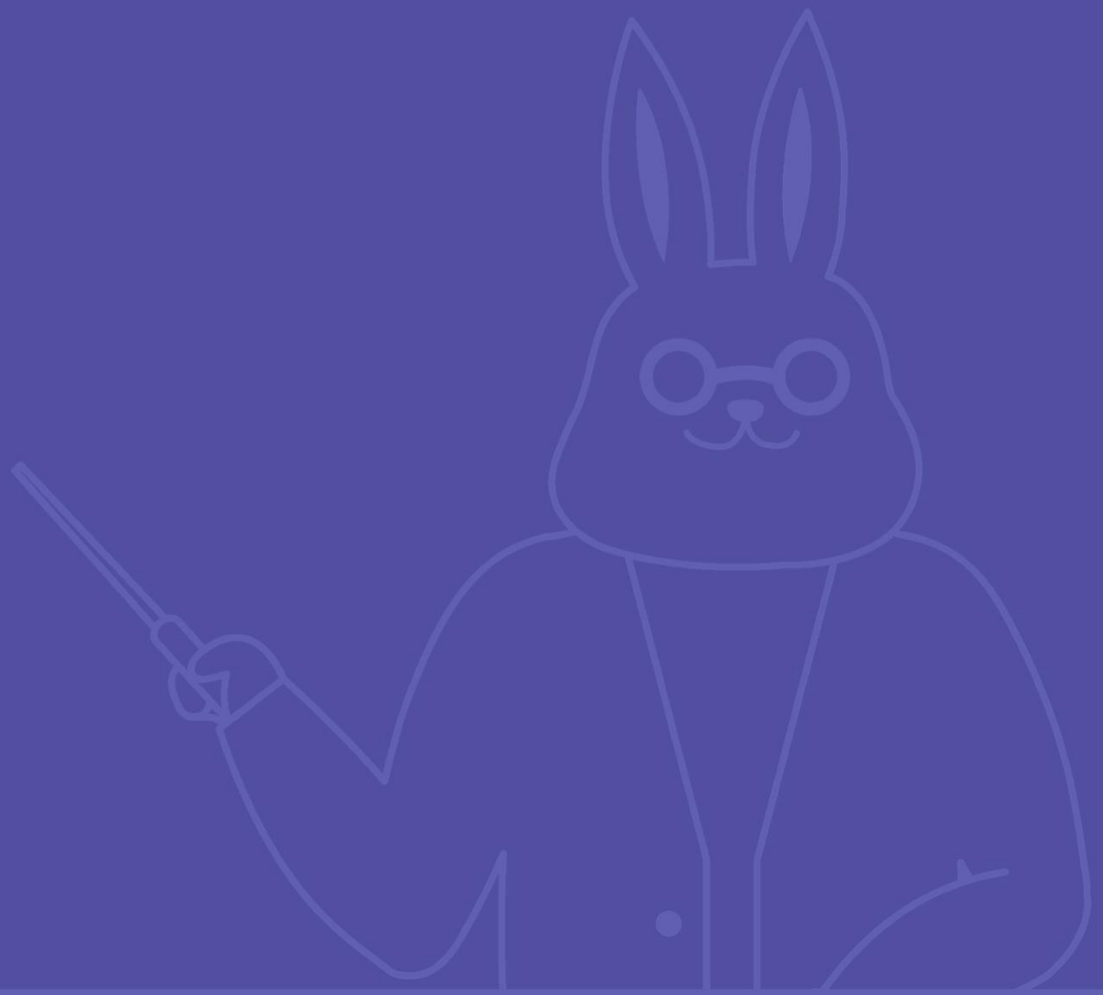
```
fit( x=None, y=None, batch_size=None, epochs=1, verbose='auto',
callbacks=None, validation_split=0.0, validation_data=None, shuffle=True,
class_weight=None, sample_weight=None, initial_epoch=0, steps_per_epoch=None,
validation_steps=None, validation_batch_size=None, validation_freq=1,
max_queue_size=10, workers=1, use_multiprocessing=False )
```

콜백함수(Callback function)

- 다른 코드의 인수로 함수를 넘겨주면, 그 코드가 필요에 따라 실행하는 함수
- callbacks 옵션으로 함수들의 리스트를 전달하면 학습 과정 중에 Tensorflow가 실행
- 다양한 함수로 학습과정을 세밀하게 조정하거나, 중간 경과를 살펴볼 수 있음

03

콜백함수



콜백함수 정의하기

keras.callbacks.Callback을 상속받아서 클래스 정의

호출될 타이밍에 따라 해당 함수를 재정의

- on_train_end(self, logs=None): 학습이 종료될 때 호출
- on_epoch_end(self, epoch, logs=None): 한 epoch이 끝날 때 호출
- on_predict_end(self, logs=None): 예측이 끝날 때 호출
- …

정의한 클래스의 인스턴스를 리스트에 포함시켜 callbacks에 전달

코드

```
class MyCallback(keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs=None):
        keys = list(logs.keys())
        print("End epoch {} of training; got log keys: {}".format(epoch, keys))

model.fit( ... , callbacks=[MyCallback()])
```

내장 콜백함수

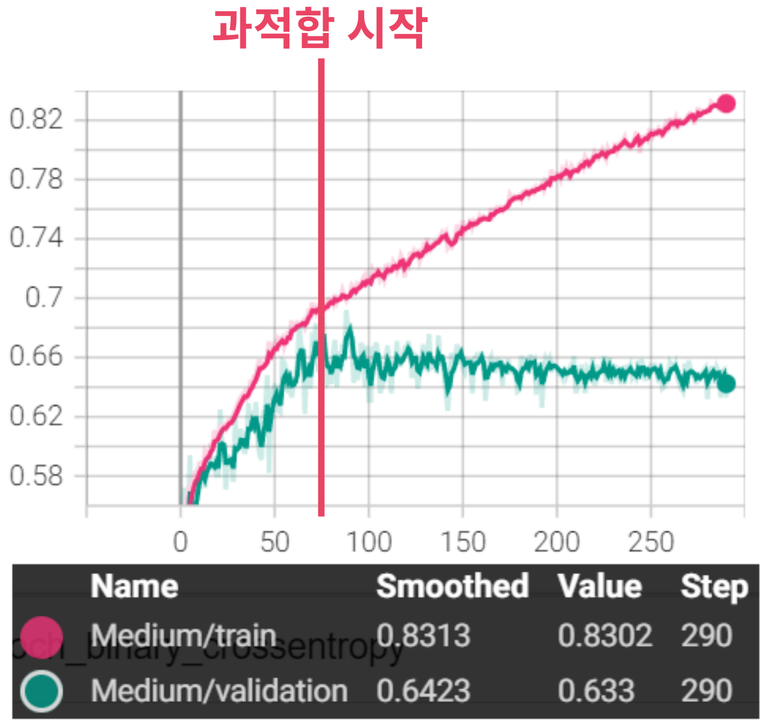
Tensorflow는 다양한 콜백함수를 제공

tf.keras.callbacks에 포함된 주요 콜백함수

- EarlyStopping: 학습이 진전이 없을 경우, 조기에 학습을 종료
- ModelCheckpoint: 모델을 주기적으로 자동 저장
- TensorBoard: 학습 과정이나 모델의 정보를 시각화 할 수 있도록 로그를 기록

EarlyStopping의 개념

- 일정 시점부터 학습 성능이 증가하지만, 검증 성능이 감소하거나 유지되는 과적합
- 과적합이 시작되면 더 학습을 진행할수록 성능이 감소함
- 과적합이 발생하는 Epoch을 감지하고 자동으로 학습을 종료하는 방법



EarlyStopping

- 모델 학습의 목표는 Loss 값을 최소화하는 과정
- Loss값을 감시하고 있다가 더 이상 감소하지 않는다면 학습이 종료됨
- 과적합을 방지하고 무의미한 학습과정을 생략하여 연산자원 절약

tf.keras.callbacks.EarlyStopping( )

```
tf.keras.callbacks.EarlyStopping(
    monitor='val_loss', min_delta=0, patience=0, verbose=0,
    mode='auto', baseline=None, restore_best_weights=False
)
```

EarlyStopping

monitor: 감시할 수치의 이름

Ex)'val_loss’: 검증 Loss값

mode: 원하는 방향

- monitor값이 어떻게 되어야 하는 값인지 결정
- ‘min’: monitor값이 최소가 되어야 성능이 좋아지는 모델일 때(Loss, Error)
- ‘max’: monitor값이 최대가 되어야 성능이 좋아지는 모델일 때(정확도, 성능)

tf.keras.callbacks.EarlyStopping( )

```
tf.keras.callbacks.EarlyStopping(
    monitor='val_loss', min_delta=0, patience=0, verbose=0,
    mode='auto', baseline=None, restore_best_weights=False
)
```

ModelCheckpoint

- 일정 주기마다 모델이나 모델의 가중치를 자동으로 저장
- 과적합이 발생했을 때 처음부터 수행하지 않고 중간부터 다시 학습이 가능
- 성능 그래프를 보면서 가장 좋은 성능의 가중치를 서비스에 사용
- 가장 성능이 좋은 버전만 저장하는 기능이 포함

tf.keras.callbacks.ModelCheckpoint( )

```
tf.keras.callbacks.ModelCheckpoint(
    filepath, monitor='val_loss', verbose=0, save_best_only=False,
    save_weights_only=False, mode='auto', save_freq='epoch'    ,
    options=None, **kwargs
)
```

ModelCheckpoint

monitor, mode

- 감시할 지표와 모드: EarlyStopping과 동일

save_best_only : 가장 좋은 버전만 저장

- True: 성능이 가장 좋은 버전만 남기고 다른 버전의 체크포인트는 모두 삭제
- False: 모든 버전을 저장하고 삭제하지 않음

save_weights_only: 모델의 가중치의 값만 저장

- True: 학습 진행상황, 모델의 구조에 대한 데이터를 빼고 저장하여 공간이 절약되지만 불러오기 위해서는 모델의 구조를 저장하고 있어야 함

tf.keras.callbacks.ModelCheckpoint( )

```
tf.keras.callbacks.ModelCheckpoint(
    filepath, monitor='val_loss', verbose=0, save_best_only=False,
    save_weights_only=False, mode='auto', save_freq='epoch'    ,
    options=None, **kwargs
)
```

ModelCheckpoint의 주의할 점

filepath: 체크 포인트를 저장할 디렉토리

- 고정된 문자열을 전달하면 하나의 디렉토리에 계속 저장
- Epoch에 따라 이름을 다르게 하도록 설정
- 포맷 문자열을 사용가능

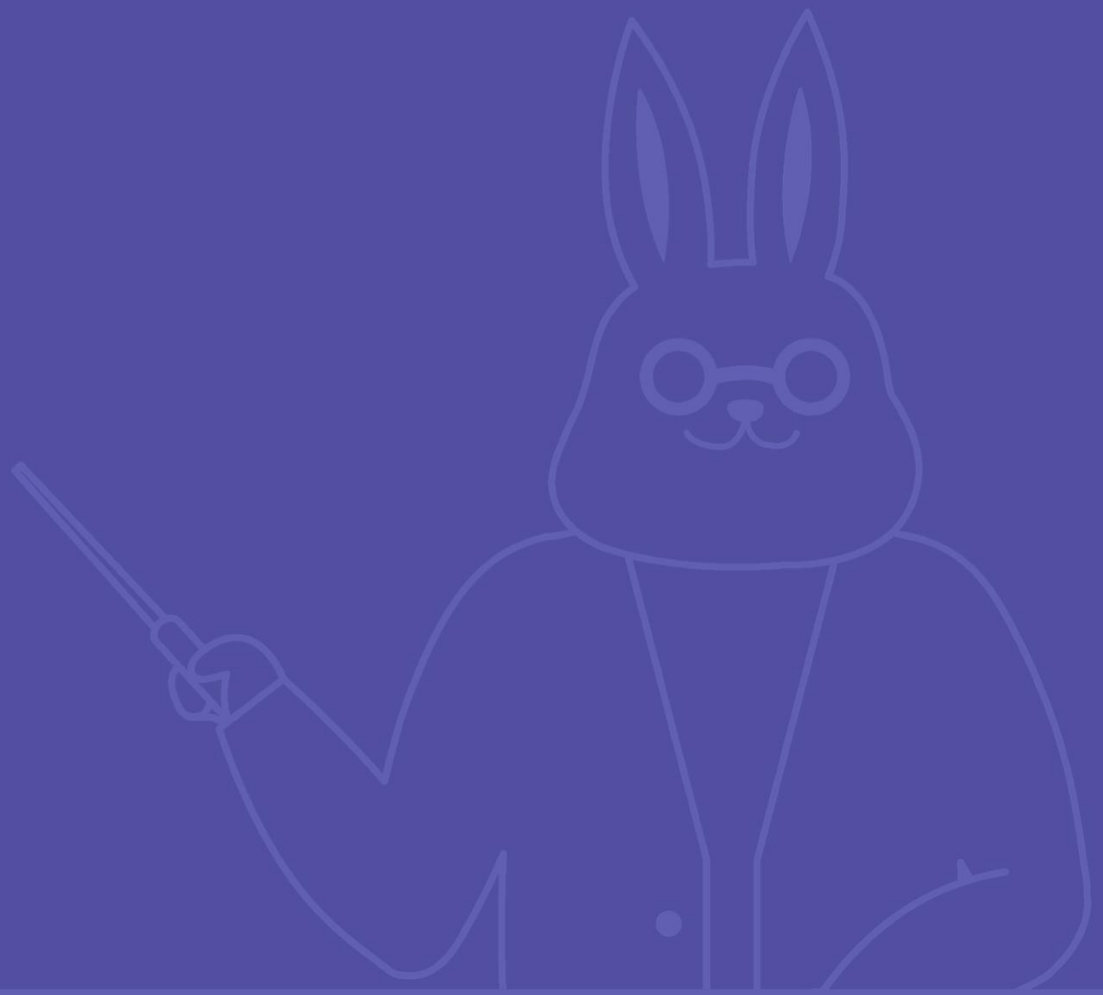
```
filepath = 'checkpoints/cp-{epoch:04d}.ckpt'
```

- 저장된 디렉토리 구조의 예시

```
checkpoints
├── cp-0001.ckpt
│   ├── assets
│   ├── keas_metadata.pb
│   ├── saved_model.pb
│   └── variables
│       ├── variables.data-00000-of-00001
│       └── variables.index
├── cp-0002.ckpt
...
```

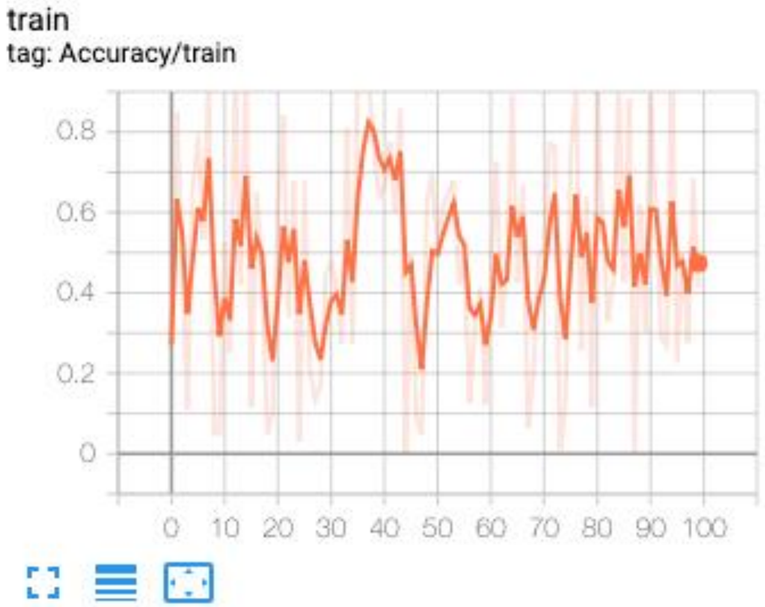
04

TensorBoard

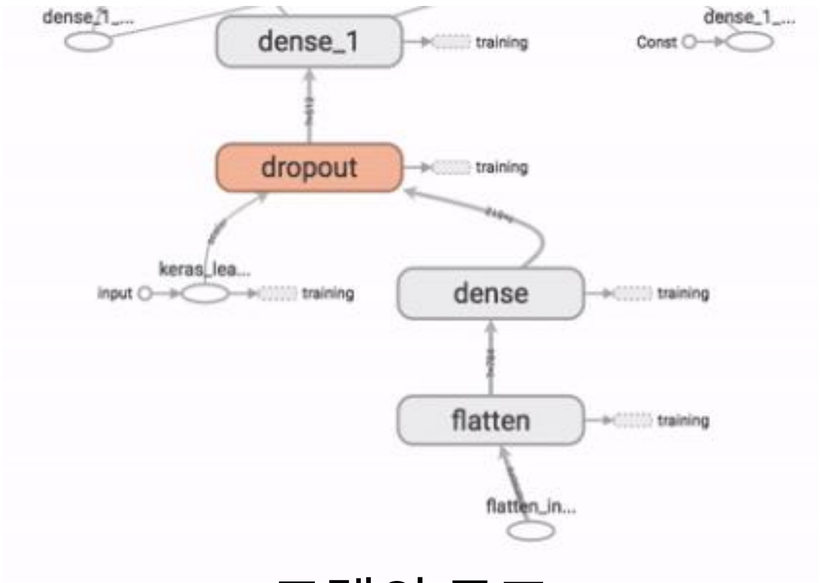


TensorBoard 란?

모델의 정보를 시각화하는 TensorBoard를 사용할 수 있도록 로그를 기록



실시간 학습상황



모델의 구조

TensorBoard을 사용하는 이유

장점

- 원격에서 실시간으로 모델의 학습정보를 포함한 다양한 정보를 확인
- 간단하게 Callback을 추가하고 시각화된 웹페이지를 제공
- 포트와 IP를 설정하면 서버가 아닌 원격에서 이 결과물을 활용 가능

사용과정

- Callback에 TensorBoard 콜백함수를 추가하여 로그 디렉토리에 기록을 저장
- TensorBoard 모듈을 실행하여 로그 디렉토리의 로그를 웹페이지로 호스트
- 웹페이지에 출력된 정보들을 보고 학습 과정을 모니터링

TensorBoard 콜백함수의 인자

log_dir: 로그를 저장할 경로

update_freq: 저장하는 주기

- ‘epoch’ : 한 epoch마다 기록을 저장
- ‘batch’: batch마다 기록을 저장 (학습 속도가 느려짐)

tf.keras.callbacks.TensorBoard()

```
tf.keras.callbacks.TensorBoard(
    log_dir='logs', histogram_freq=0, write_graph=True,
    write_images=False, write_steps_per_second=False, update_freq='epoch'    ,
    profile_batch=0, embeddings_freq=0, embeddings_metadata=None, **kwargs
)
```

TensorBoard 사용 방법

Fit 함수에 콜백함수 추가하기

```
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir="./logs")
model.fit(x_train, y_train, epochs=2, callbacks=[tensorboard_callback])
```

명령줄에서 tensorboard를 실행

- '--logdir'에 저장한 디렉토리 경로를 전달

```
tensorboard --logdir=path_to_your_logs
```

크레딧

/* elice */

코스 매니저

-

콘텐츠 제작자

김승환

강사

김승환

감수자

-

디자이너

강혜정

연락처

TEL

070-4633-2015

WEB

https://elice.io

E-MAIL

contact@elice.io