교육 제목	데이터 기반 인공지능 시스템 엔지니어 양성 과정
교육 일시	2021년 10월 25일
교육 장소	YGL C-6 학과장 & 자택(디스코드 이용한 온라인)
교육 내용	
오전	추가 정리 자료 1. 텐서플로

1). 텐서플로우(TensorFlow)는 텐서(Tensor)와 "흐르다"의 영어표현인 플로우(Flow)를 합친뜻
결국 "**텐서라는것이 흐른다**" 라는 모습을 브랜딩화 했다고 한다.

- **텐서(tensor)**는 벡터와 행렬을 일반화한 것이고 고차원으로 확장 가능. 내부적으로 텐서플로는 기본적으로 제공되는 자료형을 사용해 n-차원 배열로 나타낸다.

간단하게 생각하면 텐서는 텐서플로우가 연산하기 위해 사용하는 **자료형, 데이터의 형태**이다.
머신러닝에서는 데이터를 N차원의 배열로 구성하는데,
예를 들어 RDB로 비교를 해본다면 각 컬럼의 갯수가 N개이면, 해당 데이터로 1차원 ~ N차원상의 배열까지 만들 수있다.

3). 텐서의 구성요소
텐서 플로우의 자료형인 텐서는 랭크, 형태, 타입 3가지 구성요소가 있다.

랭크	형태	차원	예제
0	[]	0-차원	스칼라인 0-차원 텐서.
1	[D0]	1-차원	형태가 [5]인 1-차원 텐서.
2	[D0, D1]	2-차원	형태가 [3, 4]인 2-차원 텐서.
3	[D0, D1, D2]	3-차원	형태가 [1, 4, 3]인 3-차원 텐서.
n	[D0, D1, ... Dn-1]	n-차원	형태가 [D0, D1, ... Dn-1]인 텐서.

3-1) 랭크 (Rank)

- 텐서를 설명할때 N-차원으로 설명하였는데, 이 차원이라는것이 랭크라고 생각하면된다.
- 1차원 = 1Rank / 3차원 = 3Rank

@데이터를 통해 확인해보자

7 = > 0-rank Tensor = 스칼라
[1,2,3] => 1-rank Tensor = 1차원 백터
[[1,2,3],[4,5,6]] => 2-rank Tensor = 2차원 백터인 행렬
[[[1,2,3]],[[3,4,5]]] = > 3-rank Tensor = 3차원 백터

3-2) 형태(Shape)

- 각 랭크(차원)에 있는 원소개수 == 몇개의 데이터로 이루어져있는지

@데이터를 통해 확인해보자

7 # 0-Rank Tensor & shape : []
[1 ,2, 3] # 1-Rank Tensor a& shape : shape [3]
[[1, 2, 3], [4, 5, 6]] # 2-Rank Tensor & shape : shape [2, 3]
[[[1, 2, 3]], [[7, 8, 9]]] # 3-Rank Tensor & shape : shape [2, 1, 3]

2. 일반적 인공지능 구현 과정

1. 문제 정의 2. 입력과 출력 결정 3. 데이터 수집 4. 데이터 정제 및 전처리 5. 머신러닝/딥러닝 모델 구축 6. 학습 및 테스트	4.데이터 전처리 : onehotencoding → 정규화 → data 나누기 5. 모델구축 : 적용 모델 선정 → Layer 구축 → option 선정

오후

딥러닝 실습

1. 데이터 정규화

```
# data의 min, max, mean, std값 구하기
dataset_stats = data.describe()
dataset_stats = dataset_stats.transpose()
#data.min()
#data.max()
#data.mean()
```

```
## data normalization
def min_max_norm(x):
  return (x - dataset_stats['min']) / (dataset_stats['max'] - dataset_stats['min'])

def standard_norm(x):
  return (x - dataset_stats['mean']) / dataset_stats['std']
```

```
min_max_norm_train_data = min_max_norm(data)
standard_norm_train_data = standard_norm(data)
```

```
normed_train_data = standard_norm(data)
```

2. 데이터 분리(훈련세트, 검증세트, 테스트세트)

```
data = pd.read_csv('./BostonHousing.csv')
```

```
x_data = data.copy()

ori_y = x_data.pop('MEDV')
```

```
from sklearn.model_selection import train_test_split

x_train1, x_test, y_train1, y_test =
train_test_split(x_data, ori_y, test_size=0.3,
shuffle=True)
x_train, x_valid, y_train, y_valid =
train_test_split(x_train1, y_train1, test_size=0.2,
shuffle=True)
```

```
result = model.fit(x_train, y_train, epochs = 200,
batch_size = 10, validation_data=(x_valid, y_valid))
```

3. One hot encoding

```
from sklearn.preprocessing import LabelEncoder
```

```
# case 1) sklearn
# Label Encoder는 독립 변수가 아닌 종속 변수(라벨)에 대해 사용한다.
# 문자열이나 정수로된 라벨 값을  0  ~  K-1 까지의 정수로 변환.

e = LabelEncoder()
e.fit(y_data)
print("Label Class String : {}".format(e.classes_))

Y = e.transform(y_data)
print("Label Class int : {}".format(Y))

y_encoded = tf.keras.utils.to_categorical(Y)
print("case 1 One hot label class : {}".format(y_encoded))

print(np.argmax(y_encoded, axis=1).reshape(-1,1))
print(y_encoded.shape)
```

```
# case 2) pandas
one_hot_label = pd.get_dummies(y_data)
print("case2 one_hot_label : ", one_hot_label)
print(one_hot_label.shape)
```

4. Iris multi classification 연습

```
import tensorflow as tf

import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

import pandas as pd
```

```
data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/영우_4기_딥러닝/dataset/iris.csv', names = ["sepal_length", "sepal_width", "petal_length", "petal_width", "species"])
```

```
x_data = data.copy()
y_data = x_data.pop("species")
X = x_data.copy()
```

```
# 문자열을 숫자로 변환
e = LabelEncoder()
e.fit(y_data)
y = e.transform(y_data)
y_encoded = tf.keras.utils.to_categorical(y)
```

```
# train과 test를 분리
```

```
x_train1, x_test, y_train1, y_test =
train_test_split(X, y_encoded, test_size=0.1,
shuffle=True)
# train set에서 train과 validation 분리
x_train, x_valid, y_train, y_valid =
train_test_split(x_train1, y_train1, test_size=0.2,
shuffle=True)
```

```
input_layer = tf.keras.layers.Input(shape=(4,))
x = tf.keras.layers.Dense(16,
activation='sigmoid')(input_layer)
x = tf.keras.layers.Dense(32,
activation='sigmoid')(x)
output_layer = tf.keras.layers.Dense(3,
activation='softmax')(x)

model = tf.keras.models.Model(inputs=[input_layer],
outputs = [output_layer])
model.summary()
```

```
loss = tf.keras.losses.categorical_crossentropy
optimizer = tf.keras.optimizers.SGD(learning_rate=0.04)
metrics = tf.keras.metrics.categorical_accuracy
model.compile(loss=loss, optimizer=optimizer, metrics=[metrics])
```

```
history = model.fit(x_train, y_train, epochs=200,
batch_size=50, validation_data=(x_valid, y_valid))
```

```
loss = history.history['loss']
val_loss = history.history['val_loss']

# loss와 val_loss를 그래프
epochs = range(1, len(loss)+1)
plt.subplot(211)
plt.plot(epochs, loss, 'b-', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

```
acc = history.history['categorical_accuracy']
val_acc = history.history['val_categorical_accuracy']

plt.subplot(212)  ## 2x1 개의 그래프 중에 2번째
plt.plot(epochs, acc, 'b-', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```

```
print("\n Test Accuracy : %.4f"
%(model.evaluate(x_test, y_test)[1]))
```

```
model.save('iris_multi_model.h5')
```

5. 모델 저장(바로 위) 불러오기 (아래)

```
model_path = "./{epoch:02d}-{val_loss:.4f}.h5"

model = tf.keras.models.load_model('iris_multi_model.h5')
```

6. 모델 중단 시, 재시동(?)

```
from datetime import datetime
# 개인 컴퓨터에서 텐서보드 로드하기
logdir = "log_dir/scalars/" + datetime.now().strftime("%Y%m%d-%H%M%S")
file_writer = tf.summary.create_file_writer(logdir + "/metrics")
file_writer.set_as_default()
```

```
import math
def step_decay(epoch):
 initial_lrate = 0.001
 drop = 0.98
 epochs_drop = 50.0
 lrate = initial_lrate * math.pow(drop,
math.floor((1+epoch)/epochs_drop))
 return lrate
```

```
modelpath = "./{epoch:02d}-{val_loss:.4f}.h5"
callback_list =
[tf.keras.callbacks.EarlyStopping(monitor='val_loss'
, patience=20),

tf.keras.callbacks.ModelCheckpoint(filepath=modelpat
h, monitor='val_loss', verbose=1,
save_best_only=True),
```

```
tf.keras.callbacks.LearningRateScheduler(step_decay,
verbose=1),

tf.keras.callbacks.TensorBoard(log_dir=logdir,
histogram_freq=1)]
                  # ModelCheckpoint
                  # LearningRateScheduler
```

```
result = model.fit(x_train, y_train, epochs=200,
batch_size= 50, validation_data = (x_valid,
y_valid), callbacks=callback_list)
#model.save('iris_multi_model2.h5')
```

```
# EarlyStopping ( 성능 향상이 멈추면 훈련 중지
#                 monitor : 모델의 검증 정확도를 모니터링
#                 patience : 에포크보다 더 길게 정확도가
향상되지 않으면 훈련 중지 (ex: patience=1이면, 에포크 2가
넘어갔을때)

# ModelCheckpoint ( 에포크 마다 현재 가중치 저장
#                 filepath : 모델 파일 경로
#                 save_best_only : val_loss가 줄었을때만
저장.
#                 verbose : 값 화면 표시)

# TensorBoard(log_dir='log_dir', : log 저장 폴더
#             histogram_freq=1, : 1 에포크마다 활성화
히스토그램 출력)

##  logdir가 있는 폴더에가서  tensorboard
--logdir=./log_dir/   http://localhost:6006
```