|  |  |
| --- | --- |
| 교육제목 | 데이터 기반 인공지능 시스템 엔지니어 양성 과정 |
| 교육일시 | 211026 |
| 교육장소 | 자택(디스코드 사용 온라인 학습) |
| **교육내용** | |
| 1. Normalization(정규화)   데이터가 가진 feature의 스케일이 심하게 차이가 나는 경우 문제가 되기 때문에 모든 데이터 포인트가 동일한 정도의 스케일(중요도)로 반영되도록 해주는 게 정규화의 목표임.  캡처.JPG  import seaborn as sns  import pandas as pd  import matplotlib.pyplot as plt  ## 데이터 읽어오기.## 데이터 읽어오기.  raw\_df = pd.read\_csv("test.csv")  print("data infor")  print(raw\_df.info())  print("data head")  print(raw\_df.head(10))  dataset=raw\_df.copy() # x\_data  ## 데이터 분포도 확인하기.  sns.pairplot(dataset[["A", "B", "C","D"]], diag\_kind='hist')  plt.show()  ## 정답 데이터 추출  train\_labels = dataset.pop("D") # y\_data  ## 데이터의 min , max, mean, std 값 구하기.  dataset\_stats = dataset.describe()  dataset\_stats = dataset\_stats.transpose()  ## 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(dataset)  standard\_norm\_train\_data = standard\_norm(dataset)  print("min max : ")  print(min\_max\_norm\_train\_data)  print("standard : ")  print(standard\_norm\_train\_data)   1. Data division(데이터 분리)   캡처.JPG  캡처.JPG  캡처.JPG   1. Multi classification   하나의 Perceptron ( Node )가 여러 개의 Label/Class를 분류를 할 수 있으려면 Logistic regression을 활용한  Binary classification perceptron을 여러 개 두면 됨(multi classification)  예) 강아지인지, 고양이인지, 자동차인지   * 출력 perceptron(node)의 하나하나가 물체 1개씩 binary classification 해주도록 학습, 이렇게 하려면 원 핫 인코딩(글자 데이터를 숫자로 변환해주는 작업)이 필요함.   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('iris.csv',names= ['length1','width1','length2','width2','species'])  print(data.head())  x\_data = data.copy()  y\_data = x\_data.pop('species')  X = x\_data.copy()  print(y\_data)  # 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)  print(x\_train.shape)  print(y\_train.shape)  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]))  #모델 저장  model.save('iris\_multi\_model.h5')   1. Callback 함수   모델이 학습을 시작하면 학습이 완료될 때 까지 사람이 할 수 있는게 없음. 따라서 이를 해결하고자 존재하는 것이 콜백함수. 예를 들어, 학습되는 과정 사이에 학습률을 변화시키거나 val\_loss가 개선되지 않으면 학습을 멈추게 하는 등의 작업을 할 수 있음.  -위 코드와 동일-  +)  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=modelpath, 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  # Tensorboard Magic Command  %load\_ext tensorboard  %tensorboard --logdir {logdir}  # 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 | |