|  |  |
| --- | --- |
| 교육 제목 | Tensorflow |
| 교육 일시 | 2021. 10. 25 |
| 교육 장소 | 영우글로벌러닝 |
| **교육 내용** | |
| 오전 | import matplotlib.pyplot as plt  import tensorflow as tf  import numpy as np  # data 선언  x\_data=[[1.],[2.],[3.],[4.]]  y\_data=[[1.],[3.],[5.],[7.]]  plt.plot(x\_data, y\_data, 'o')  ## 평균 0, 분산 1의 파라미터의 정규분포로 부터 값을 가져옴.  # 학습을 통해 업데이트가 되어 변화되는 모델의 파라미터인 w,b를 의미한다.  W = tf.Variable(tf.random.normal((1,1),mean=0, stddev=1.0))  b = tf.Variable(tf.random.normal((1,1),mean=0, stddev=1.0))  lr = 0.001  history = np.zeros([2000,3], 'float32')  w = np.array(W)  w = w.reshape(1)  B = np.array(b)  B = B.reshape(1)  plt.plot(x\_data, y\_data, 'o')  plt.plot([0,4], [b, (w\*4 +B)], 'r-')  w\_trace = []  b\_trace = []  for epoch in range(2000):  total\_error = 0  for x, y in zip(x\_data, y\_data):  with tf.GradientTape() as tape:  y\_hat = W \* x + b  error = (y\_hat - y) \*\*2    gradients = tape.gradient(error, [W, b]) # 별명.gradient(타겟값, [업데이터시킬 파라미터, ...])  W = tf.Variable(W - lr \* gradients[0])  b = tf.Variable(b - lr \* gradients[1])  w\_trace.append(W.numpy())  b\_trace.append(b.numpy())  visual\_error = tf.square(error)  total\_error = total\_error + visual\_error  print("epoch : ", epoch, "error :", total\_error/len(x\_data))  history[epoch,:] = [(total\_error/len(x\_data))[0], W[0], b[0]]  w = np.array(W)  w = w.reshape(1)  B = np.array(b)  B = B.reshape(1)  plt.plot(x\_data, y\_data, 'o')  plt.plot([0, 4], [b, (w\*4 + B)], 'r-')  # 학습이 끝난 후 W와 b로 예측  print(history)  print("W :", W)  print("b:", b)  print("input 3", tf.add(tf.matmul([[3.]], W), b))  print("input 4", tf.add(tf.matmul([[4.]], W), b))  # loss function  plt.plot(history[:,0])  plt.title('loss function')  plt.xlabel('epoch')  plt.ylabel('loss')  a = np.array(w\_trace)  a = a.reshape(8000,1)  plt.plot(a)  import tensorflow as tf  ## data 선언  x\_data = [[2.,0.,7.], [6.,4.,2.], [5.,2.,4.],[8.,4.,1]]  y\_data = [[75], [95], [91], [97]]  test\_data=[[5.,5.,5.]]  print(len(x\_data),len(x\_data[1])) # 행크기 , 열크기  ## tf.keras를 활용한 perceptron 모델 구현.  model = tf.keras.Sequential() ## 모델 만들기 위해 sequential 매서드를 선언. 이를 통해 모델을 만들 수 있다.  model.add(tf.keras.layers.Dense(1, input\_dim=3)) # 선언된 모델에 add를 통해 쌓아감. , 현재는 입력 변수 갯수 3, perceptron 1개.  model.summary() ## 설계한 모델 프린트  loss = tf.keras.losses.mse # mean squre error  optimizer = tf.keras.optimizers.SGD(lr=0.01)  metrics = tf.keras.metrics.mae # mean absolute error ㅣ예측값 - 정답ㅣ  # 모델 컴파일하기  model.compile(loss=loss, optimizer=optimizer, metrics=[metrics])  # 모델 동작하기  model.fit(x\_data, y\_data, epochs=20, batch\_size=2)  # 결과를 출력  print(model.weights)  print("test data [5., 5., 5.] 예측 값 :", model.predict(test\_data))  print(model.weights)  print("test data [0.8, 0.8] 예측 값 :", model.predict(test\_data))  if model.predict(test\_data) >0.5:  print("합격")  else:  print("불합격") |
| 오후 | import matplotlib.pyplot as plt  import tensorflow as tf  import numpy as np  import tensorflow as tf  ## data 선언  x\_data = [[5.], [30.], [95.], [100.], [265.], [270.], [290.], [300.],[365.]]  y\_data = [[0.], [0.], [0.], [0.], [1.], [1.], [1.], [1.], [1.]]  test\_data= [[7.]]  test\_data2= [[80.]]  test\_data3= [[110.]]  test\_data4= [[180.]]  test\_data5= [[320.]]  ## tf.keras를 활용한 perceptron 모델 구현.  model = tf.keras.Sequential() ## 모델 선언  model.add(tf.keras.layers.Dense(1, input\_dim=1, activation='sigmoid'))  model.summary()  loss = tf.keras.losses.mse # mean squre error ### 경사 하강법으로 global min 에 찾아가는 최적화 방법 선언.  optimizer = tf.keras.optimizers.SGD(lr=0.01) ## 예측값 과 정답의 오차값 정의. mse는 mean squre error로 (예측값 - 정답)^2 를 의미  metrics = tf.keras.metrics.binary\_accuracy # mean absolute error ㅣ예측값 - 정답ㅣ  # 모델 컴파일하기  model.compile(loss=loss, optimizer=optimizer, metrics=[metrics])  # 결과를 출력  # 결과를 출력합니다.  print(" test data [7.] 예측 값 : ", model.predict(test\_data))  print(" test data [80.] 예측 값 : ", model.predict(test\_data2))  print(" test data [110.] 예측 값 : ", model.predict(test\_data3))  print(" test data [180.] 예측 값 : ", model.predict(test\_data4))  print(" test data [320.] 예측 값 : ", model.predict(test\_data5))  x\_data = [[0.,0.], [0.,1.], [1., 0.], [1.,1.]]  y\_data = [[0.], [1.], [1.],[0.]]  test\_data = [[0.5, 0.5]]  model = tf.keras.Sequential()  model.add(tf.keras.layers.Dense(4, input\_dim=2, activation='sigmoid'))  model.add(tf.keras.layers.Dense(5, activation='sigmoid'))  model.add(tf.keras.layers.Dense(1, activation='sigmoid'))  model.summary()  optimizer=tf.keras.optimizers.SGD(learning\_rate=0.5) ### 경사 하강법으로 global min 에 찾아가는 최적화 방법 선언.  loss=tf.keras.losses.binary\_crossentropy ## 예측값 과 정답의 오차값 정의.  metrics=tf.keras.metrics.binary\_accuracy ### 학습하면서 평가할 메트릭스 선언언  model.compile(loss =loss, optimizer= optimizer, metrics=[metrics])  plt.plot(hist.history['loss'])  print(" test data [0.5 0.5] 예측 값 : ", model.predict(test\_data))  --------------------------------------------------------------------------  import pandas as pd  import tensorflow as tf  df = pd.read\_csv("./house\_price\_of\_unit\_area.csv")  df.head()  x\_data = df.copy()  tf.random.set\_seed(777) # random seed를 고정해줘서 돌릴때마다 랜덤값을 고정  y\_data = x\_data.pop("house price of unit area")  print(y\_data)  print(x\_data)  model = tf.keras.Sequential()  model.add(tf.keras.layers.Dense(64, input\_dim=5, activation='sigmoid'))  model.add(tf.keras.layers.Dense(128, activation='sigmoid'))  model.add(tf.keras.layers.Dense(256, activation='sigmoid'))  model.add(tf.keras.layers.Dense(1))  model.summary()  optimizer=tf.keras.optimizers.SGD(learning\_rate=0.04) ### 경사 하강법으로 global min 에 찾아가는 최적화 방법 선언.  loss=tf.keras.losses.mean\_squared\_error ## 예측값 과 정답의 오차값 정의.  metrics=tf.keras.metrics.RootMeanSquaredError() ### 학습하면서 평가할 메트릭스 선언언  model.compile(loss =loss, optimizer= optimizer, metrics=[metrics])  result = model.fit(x\_data, y\_data, epochs=100, batch\_size=100)  print(result.history.keys())  loss = result.history['loss']  # loss 그래프  epochs = range(1, len(loss)+1)  plt.subplot(211)  plt.plot(epochs, loss, 'b-', label='Training loss')  plt.title('Training loss')  plt.xlabel('Epochs')  plt.ylabel('Loss')  plt.legend()  mae = result.history['root\_mean\_squared\_error']  epochs = range(1, len(mae)+1)  # mean absolute error 그래프화  plt.subplot(212)  plt.plot(epochs, mae, 'r-', label='Training mae')  plt.title('Training rmse')  plt.xlabel('Epochs')  plt.ylabel('rmse')  plt.legend()  # x\_data -> model -> predict값 - y\_data값을 추출  print(model.evaluate(x\_data, y\_data))  print("\n Test rmse : %.4f" % (model.evaluate(x\_data, y\_data)[1])) |