|  |  |
| --- | --- |
| 교육 제목 | Tensorflow |
| 교육 일시 | 2021. 10. 21 |
| 교육 장소 | 영우글로벌러닝 |
| **교육 내용** | |
| 오전 | import tensorflow as tf  print(tf.\_\_version\_\_)  # 1. tf.constant로 실수 선언해서 연산 수행하기  x1=tf.constant(5)  x2=tf.constant(3)  x3=tf.constant(2)  # 연산  add\_result=tf.add(x2, x3) # 3+2  mul\_result=tf.multiply(x1, x3)  sub\_result=tf.subtract(mul\_result, add\_result)  print("add\_result = {}".format(add\_result))  print("mul\_result = {}".format(mul\_result))  print("sub\_reuslt = {}".format(sub\_result))  # 2. tf.Varialbe로 상수 선언해서 연산 수행하기  x1=tf.Variable(5)  x2=tf.Variable(3)  x3=tf.Variable(2)  # 연산  add\_result=tf.add(x2, x3) # 3+2  mul\_result=tf.multiply(x1, x3)  sub\_result=tf.subtract(mul\_result, add\_result)  print("add\_result = {}".format(add\_result))  print("mul\_result = {}".format(mul\_result))  print("sub\_reuslt = {}".format(sub\_result))  import numpy as np  x1 = tf.Variable(5)  x2 = np.array(3, dtype= 'int32')  x3 = tf.Variable(2)  # 연산  add\_result = tf.add(x2, x3) # 3+2  mul\_result = tf.multiply(x1, x3)  sub\_result = tf.subtract(mul\_result, add\_result)  print("add\_result = {}".format(add\_result))  print("mul\_result = {}".format(mul\_result))  print("sub\_result = {}".format(sub\_result))  # tf를 선언하고 for문을 이용하여 1부터 10까지 더해지는 코드를 작성  # 변수 선언  x=tf.Variable(0)  # for문  for i in range(1, 11):  x=x+i  print("value : {}".format(x))    print("total sum : {}".format(x))  # 랜덤 변수 선언하기  # 평균이 0이고, 분산이 1인 파라미터의 정규분포로부터 값 구하기  x1 = tf.Variable(tf.random.normal((1,1), mean=0, stddev=1.0))  x2 = tf.Variable(tf.random.normal((1,1), mean=0, stddev=1.0))  x3 = tf.Variable(tf.random.normal((1,1), mean=0, stddev=1.0))  add\_result = tf.add(x2, x3) # 3+2  mul\_result = tf.multiply(x1, x3)  sub\_result = tf.subtract(mul\_result, add\_result)  print("add\_result = {}".format(add\_result))  print("mul\_result = {}".format(mul\_result))  print("sub\_result = {}".format(sub\_result))  # 다차워너 변수 선언하기  x1=tf.Variable([[1,2],[3,4],[5,6],[7,8]])  print("x1 = {}".format(x1))  print("x1 = {}".format(x1.shape))  x1\_reshape=tf.reshape(x1, [2,4])  print("x1 = {}".format(x1\_reshape))  print("x1 = {}".format(x1\_reshape.shape))  # scalar  scalar\_A=tf.constant(1)  scalar\_B=tf.constant(1)  add\_scalar\_AB=scalar\_A+scalar\_B  # 1-dim  vector\_A=tf.constant([1, 2, 3])  vector\_B=tf.constant([3, 5, 7])  add\_vector\_AB=vector\_A+vector\_B  # 2-dim  matrix\_A=tf.constant([[1, 2],  [3, 4]])  matrix\_B=tf.constant([[4, 3],  [2, 1]])  add\_matrix\_AB=matrix\_A+matrix\_B  print('==== scalar shape ====')  print(scalar\_A.shape)  print(scalar\_B.shape)  print(add\_scalar\_AB.shape)  print('==== vector shape ====')  print(vector\_A.shape)  print(vector\_B.shape)  print(add\_vector\_AB.shape)  print('==== matrix shape ====')  print(matrix\_A.shape)  print(matrix\_B.shape)  print(add\_matrix\_AB.shape) |
| 오후 | A = tf.constant([[2, 2, 4],  [1, 1, 6],  [1, 3, 8]])  B = tf.constant([ [4, 3, 3],  [2, 1, 6],  [1, 2, 8] ])  AB = tf.matmul(A, B)  print("result = {}".format(AB))  print(AB.shape)  ## data 선언  x\_data =[[1.],[2.],[3.],[4.]]  y\_data =[[1.],[3.],[5.],[7.]]  test\_data=[[9.]]  print(x\_data)  print(y\_data)  ## 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 ))  b= tf.Variable(tf.random.normal((1,1), mean=0, stddev=1 ))  print("W : ", W)  print("b : ", b)  for j in range(len(x\_data)):  ## data \* weight 작성  WX = tf.matmul([x\_data[j]], W)  ## bias add 작성  y\_hat = tf.add(WX,b)  ## W와 b로 예측 하기  print("y\_data: , ",y\_data[j], "prediction : ", y\_hat)  ## 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 ))  b= tf.Variable(tf.random.normal((1,1), mean=0, stddev=1 ))  lr=tf.constant(0.0001)  history=np.zeros([2000, 3], 'float32')  for i in range(2000): # 에폭  total\_error=0    for j in range(len(x\_data)): # 배치 1  # data\*weight  WX=tf.matmul([x\_data[j]], W) #      # bias add  y\_hat=tf.add(WX, b) # y=WX+b    # 정답인 Y와 출력값의 error 계산  error= tf.subtract(y\_data[j], y\_hat) # true-prediction  # 경사하강법으로 W와 b 업데이트  # 도함수 구하기  diff\_W=tf.matmul(error, [x\_data[j]]) # error\*x의 합  diff\_b=error    # 업데이트할 만큼 러닝레이트 곱  diff\_w=tf.multiply(lr, diff\_W)  diff\_b=tf.multiply(lr, diff\_b) # lr\*error  # w,b 업데이트  W=tf.add(W, diff\_W) # W+delta w+lr\*x\*(error)  b=tf.add(b, diff\_b) # b+delta b+lr\*(error)  # 토탈 에러  visual\_error=tf.square(error)  total\_error=total\_error+visual\_error  #모든 데이터에 따른 error 값  print("epoch: ", i, "error : ", total\_error/len(x\_data))  history[i,:]=[(total\_error/len(x\_data))[0], W[0], b[0]]  # 학습이 끝난 후 W와 b로 예측하기  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))  import tensorflow as tf  x\_data =[[1.],[2.],[3.],[4.]]  y\_data =[[1.],[3.],[5.],[7.]]  test\_data=[[9.]]  model=tf.keras.Sequential()  model.add(tf.keras.layers.Dense(1, input\_dim=1))  model.summary()  optimizer=tf.keras.optimizers.SGD(lr=0.01) # 경사하강법 global min 찾아그는 최적의 방법  loss=tf.keras.losses.mse # 예측한 값과 정답간의 오차를 계산 mse=mean square error (예측값-정답)^2  metrics=tf.keras.metrics.mae # mae=mean absolute error |예측값-정답|  # 모델 컴파일  # model.compile(loss='mse', optimizer='SGD', metrics='mae')  model.compile(loss=loss, optimizer=optimizer, metrics=[metrics])  model.fit(x\_data, y\_data, epochs=2000, batch\_size=1)  # 결과를 출력  print(model.weights)  print("test data [9.] 예측값 : ", model.predict(test\_data)) |