

알고리즘 설계 설계프로젝트

보고서 작성 서약서

1. 나는 타학생의 보고서를 베끼거나 여러 보고서의 내용을 짜집기하지 않겠습니다.

2. 나는 보고서의 주요 내용을 인터넷사이트 등을 통해 얻지 않겠습니다.

3. 나는 보고서의 내용을 조작하지 않겠습니다.

4. 나는 보고서 작성에 참고한 문헌의 출처를 밝히겠습니다.

5. 나는 나의 보고서를 제출 전에 타학생에게 보여주지 않겠습니다.

나는 보고서 작성시 윤리에 어긋난 행동을 하지 않고 정보통신공학인으로서 나의 명예를 지킬 것을 맹세합니다.

2017년 06월 18일

학부 정보통신공학과

학년 3

성명 최민석

학번 12121710



1. 개요

주어진 데이터를 기반으로 딥러닝을 구현하는 프로젝트였습니다. 딥러닝을 실행하기 위해서 파이썬, 파이참 그리고 tensorflow를 이용해 구현하였습니다. Tensorflow를 이용하여 세 가지 모델을 구현하였습니다. Multi-variable linear regression, sigmoid 그리고 softmax 구현하였습니다. 이것을 구현하기 전에 엑셀의 데이터의 값을, 한 분반에서 가장 높은 점수를 1, 가장 낮은 점수를 0으로 0에서 1사이의 숫자로 가공하였습니다. 그리고 학점(A+, A0, B+, B0, C+, C0, D+, D0, F 총 9가지)은 각각 one-hot encoding으로 0, 0, 0, 0, 0, 0, 0, 0, 1부터 1, 0, 0, 0, 0, 0, 0, 0, 0까지 총 9가지로 설정하였습니다.

학습시킬 데이터와 테스트할 데이터는 모두 엑셀파일에 있던 데이터를 이용하였습니다. 10개의 임의의 데이터를 선택하여 테스트 테이터로 선택하였습니다. 이 테스트데이터의 학점들은 순서대로 각각 F, A+, A0, A0, C0, B+, A+, C0, A+, C+입니다.

첫 번째로 linear regression은 입력 값이 선형으로 증가할 때 사용할 수 있는 것으로, 점수와 학점은 상관관계가 있어서 사용할 수 있습니다. 하지만 여기선 HW1,2,3,4,midterm 그리고 final까지 총 6가지 input을 고려해야 합니다. 입력으로 들어온 데이터에서 점수들을 배열로 만들어 matmul()함수로 weight와 곱하고 bias를 더하였습니다. 그렇게 나온 값이 hypothesis입니다. 이 값을 Y와 빼준 값을 곱한 값이 cost가 되어서, 이 cost를 최소화 하면 예상한 값이 나옵니다. 이 모델에서는 learning rate를 0.1로 설정하였습니다.

두 번째로 sigmoid를 사용한 모델은 두 개의 레이어를 사용해 sigmoid를 두번 거치게 됩니다. Sigmoid는 data와 weight를 곱한 값에 bias를 더한값인 hypothesis가 0부터 1까지의 값을 가지게 하는 함수 입니다. 이렇게 나온 Hypothesis를 통해서 학습을 시키면 cost를 최소화하는 값을 나오게 할 수 있습니다. 이 모델도 learning rate는 0.1으로 설정하였습니다.

세 번째로 softmax는 hypothesis가 0부터 1까지의 값을 가지는 것은 sigmoid와 같습니다. 하지만 softmax는 hypothesis를 확률로 바꿔 모든 hypothesis를 더하면 1이 되는 모델입니다. Hypothesis의 값인 확률은 학습을 통해 가장 좋은 output의 값에 가장 높은 확률을 주는 것 입니다. 그래서 가장 높은 값이 1 나머지는 0으로 나와서 예측이 가능합니다. 이 모델 또한 learning rate는 0.1입니다.

1. 상세 설계내용

Multi-variable linear regression

*# Lab 4 Multi-variable linear regression***import** tensorflow **as** tf  
**import** numpy **as** np  
tf.set\_random\_seed(777) *# for reproducibility*xy = np.genfromtxt(**'data1.csv'**, delimiter=**','**)  
x\_data = xy[:442, :6]  
y\_data = xy[:442, 6:]  
  
x\_test = xy[442:, :6]  
y\_test = xy[442:, 6:]  
  
  
  
*# placeholders for a tensor that will be always fed.*X = tf.placeholder(tf.float32, shape=[**None**, 6])  
Y = tf.placeholder(tf.float32, shape=[**None**, 9])  
  
W = tf.Variable(tf.random\_normal([6, 9]), name=**'weight'**)  
b = tf.Variable(tf.random\_normal([9]), name=**'bias'**)  
  
*# Hypothesis*hypothesis = tf.matmul(X, W) + b  
  
*# Simplified cost/loss function*cost = tf.reduce\_mean(tf.square(hypothesis - Y))  
  
*# Minimize*optimizer = tf.train.GradientDescentOptimizer(learning\_rate=0.1)  
train = optimizer.minimize(cost)  
  
prediction = tf.arg\_max(hypothesis, 1)  
is\_correct = tf.equal(prediction, tf.arg\_max(Y, 1))  
accuracy = tf.reduce\_mean(tf.cast(is\_correct, tf.float32))  
  
*# Launch the graph in a session.*sess = tf.Session()  
*# Initializes global variables in the graph.*sess.run(tf.global\_variables\_initializer())  
  
**for** step **in** range(40001):  
 cost\_val, hy\_val, \_ = sess.run(  
 [cost, hypothesis, train], feed\_dict={X: x\_data, Y: y\_data})  
 **if** step % 100 == 0:  
 print(step, **"Cost: "**, cost\_val)  
  
print(**"-----------------"**)  
print(**"12121710최민석"**)  
h, p, a = sess.run([hypothesis, prediction, accuracy], feed\_dict={X: x\_test, Y: y\_test})  
print(**"\nHypothesis: "**, h)  
p = sess.run(prediction, feed\_dict={X: x\_test})  
**for** i **in** p:  
 **if** i == 8:  
 print(**"Predicion: A+"**)  
 **elif** i == 7:  
 print(**"Predicion: A0"**)  
 **elif** i == 6:  
 print(**"Predicion: B+"**)  
 **elif** i == 5:  
 print(**"Predicion: B0"**)  
 **elif** i == 4:  
 print(**"Predicion: C+"**)  
 **elif** i == 3:  
 print(**"Predicion: C0"**)  
 **elif** i == 2:  
 print(**"Predicion: D+"**)  
 **elif** i == 1:  
 print(**"Predicion: D0"**)  
 **elif** i == 0:  
 print(**"Predicion: F"**)  
print(**"Accuracy:"**, sess.run(accuracy, feed\_dict={X: x\_test, Y: y\_test}))

Sigmoid

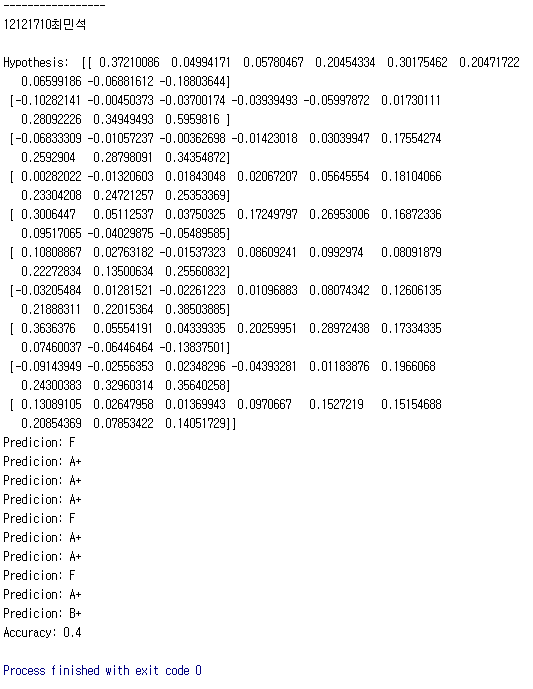
*# Lab 9 XOR-back\_prop***import** tensorflow **as** tf  
**import** numpy **as** np  
  
tf.set\_random\_seed(777)  
learning\_rate = 0.1  
nb\_classes = 9 *#output number*xy = np.genfromtxt(**'data1.csv'**, delimiter=**','**) *#read data  
  
#학습시킬 데이터*x\_data = xy[:442, :6]  
y\_data = xy[:442, 6:]  
  
*#테스트 할 데이터*x\_test = xy[442:, :6]  
y\_test = xy[442:, 6:]  
  
  
x\_data = np.array(x\_data, dtype=np.float32)  
y\_data = np.array(y\_data, dtype=np.float32)  
  
  
X = tf.placeholder(tf.float32, [**None**, 6])  
Y = tf.placeholder(tf.float32, [**None**, nb\_classes])  
  
  
W1 = tf.Variable(tf.random\_normal([6, nb\_classes]), name=**'weight1'**)  
b1 = tf.Variable(tf.random\_normal([nb\_classes]), name=**'bias1'**)  
l1 = tf.sigmoid(tf.matmul(X, W1) + b1)  
  
W2 = tf.Variable(tf.random\_normal([nb\_classes, nb\_classes]), name=**'weight2'**)  
b2 = tf.Variable(tf.random\_normal([nb\_classes]), name=**'bias2'**)  
hypothesis = tf.sigmoid(tf.matmul(l1, W2) + b2)  
  
*# cost/loss function*cost = -tf.reduce\_mean(Y \* tf.log(hypothesis) + (1 - Y) \* tf.log(1 - hypothesis))  
  
*# Loss derivative*d\_hypothesis = (hypothesis - Y) / (hypothesis \* (1.0 - hypothesis) + 1e-7)  
  
*# Layer 2*d\_sigma2 = hypothesis \* (1 - hypothesis)  
d\_a2 = d\_hypothesis \* d\_sigma2  
d\_p2 = d\_a2  
d\_b2 = d\_a2  
d\_W2 = tf.matmul(tf.transpose(l1), d\_p2)  
  
*# Mean*d\_b2\_mean = tf.reduce\_mean(d\_b2, axis=[0])  
d\_W2\_mean = d\_W2 / tf.cast(tf.shape(l1)[0], dtype=tf.float32)  
  
*# Layer 1*d\_l1 = tf.matmul(d\_p2, tf.transpose(W2))  
d\_sigma1 = l1 \* (1 - l1)  
d\_a1 = d\_l1 \* d\_sigma1  
d\_b1 = d\_a1  
d\_p1 = d\_a1  
d\_W1 = tf.matmul(tf.transpose(X), d\_a1)  
  
*# Mean*d\_W1\_mean = d\_W1 / tf.cast(tf.shape(X)[0], dtype=tf.float32)  
d\_b1\_mean = tf.reduce\_mean(d\_b1, axis=[0])  
  
*# Weight update*step = [  
 tf.assign(W2, W2 - learning\_rate \* d\_W2\_mean),  
 tf.assign(b2, b2 - learning\_rate \* d\_b2\_mean),  
 tf.assign(W1, W1 - learning\_rate \* d\_W1\_mean),  
 tf.assign(b1, b1 - learning\_rate \* d\_b1\_mean)  
]  
  
*# Accuracy computation*predicted = tf.arg\_max(hypothesis, 1)  
is\_correct = tf.equal(predicted, tf.arg\_max(Y, 1))  
accuracy = tf.reduce\_mean(tf.cast(is\_correct, tf.float32))  
  
*# Launch graph***with** tf.Session() **as** sess:  
 *# Initialize TensorFlow variables* sess.run(tf.global\_variables\_initializer())  
  
 print(**"shape"**, sess.run(tf.shape(X)[0], feed\_dict={X: x\_data}))  
  
  
 **for** i **in** range(40001):  
 sess.run([step, cost], feed\_dict={X: x\_data, Y: y\_data})  
 **if** i % 100 == 0:  
 print(i, sess.run([cost, d\_W1], feed\_dict={X: x\_data, Y: y\_data}), sess.run([W1, W2]))  
  
 *# Accuracy report* print(**'------------------'**)  
 print(**"12121710최민석"**)  
 h, c, a = sess.run([hypothesis, predicted, accuracy],feed\_dict={X: x\_test, Y: y\_test})  
 print(**"\nHypothesis: "**, h)  
 prediction = sess.run(tf.arg\_max(h, 1))  
 **for** i **in** prediction:  
 **if** i == 8:  
 print(**"Predicion: A+"**)  
 **elif** i == 7:  
 print(**"Predicion: A0"**)  
 **elif** i == 6:  
 print(**"Predicion: B+"**)  
 **elif** i == 5:  
 print(**"Predicion: B0"**)  
 **elif** i == 4:  
 print(**"Predicion: C+"**)  
 **elif** i == 3:  
 print(**"Predicion: C0"**)  
 **elif** i == 2:  
 print(**"Predicion: D+"**)  
 **elif** i == 1:  
 print(**"Predicion: D0"**)  
 **elif** i == 0:  
 print(**"Predicion: F"**)  
 print(**"\nAccuracy: "**, a)

softmax

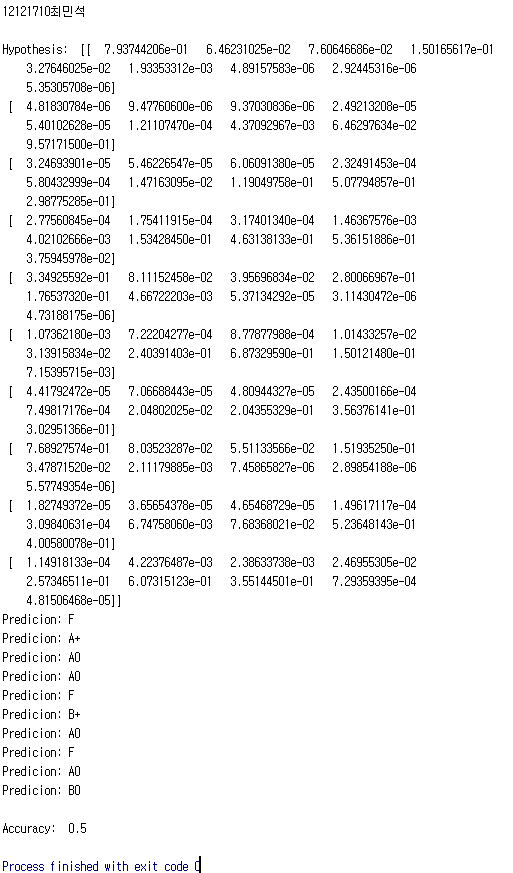
**import** tensorflow **as** tf  
**import** numpy **as** np  
tf.set\_random\_seed(777)  
  
xy = np.genfromtxt(**'data1.csv'**, delimiter=**','**)  
x\_data = xy[:442, :6]  
y\_data = xy[:442, 6:]  
  
x\_test = xy[442:, :6]  
y\_test = xy[442:, 6:]  
  
X = tf.placeholder(tf.float32, shape=[**None**, 6])  
Y = tf.placeholder(tf.float32, shape=[**None**, 9]) *# label의 갯수 : 9개*nb\_classes = 9  
  
W = tf.Variable(tf.random\_normal([6, nb\_classes]), name=**'weight'**)  
b = tf.Variable(tf.random\_normal([nb\_classes]), name=**'bias'**)  
  
hypothesis = tf.nn.softmax(tf.matmul(X, W) + b)  
cost = tf.reduce\_mean(-tf.reduce\_sum(Y \* tf.log(hypothesis), axis=1))  
optimizer = tf.train.GradientDescentOptimizer(learning\_rate=0.1).minimize(cost)  
  
prediction = tf.arg\_max(hypothesis, 1)  
is\_correct = tf.equal(prediction, tf.arg\_max(Y, 1))  
accuracy = tf.reduce\_mean(tf.cast(is\_correct, tf.float32))  
  
**with** tf.Session() **as** sess:  
 sess.run(tf.global\_variables\_initializer())  
  
 **for** step **in** range(40001):  
 sess.run(optimizer, feed\_dict={X: x\_data, Y: y\_data})  
 **if** step % 100 == 0:  
 cost\_val, W\_val, \_ = sess.run([cost, W, optimizer], feed\_dict={X: x\_data, Y: y\_data})  
 print(**'step:'**, step, **'cost:'**, cost\_val)  
  
 print(**'------------------'**)  
 print(**"12121710최민석"**)  
 h, p, a = sess.run([hypothesis, prediction, accuracy], feed\_dict={X: x\_test, Y: y\_test})  
 print(**"\nHypothesis: "**, h)  
 p = sess.run(prediction, feed\_dict={X: x\_test})  
 **for** i **in** p:  
 **if** i == 8:  
 print(**"Predicion: A+"**)  
 **elif** i == 7:  
 print(**"Predicion: A0"**)  
 **elif** i == 6:  
 print(**"Predicion: B+"**)  
 **elif** i == 5:  
 print(**"Predicion: B0"**)  
 **elif** i == 4:  
 print(**"Predicion: C+"**)  
 **elif** i == 3:  
 print(**"Predicion: C0"**)  
 **elif** i == 2:  
 print(**"Predicion: D+"**)  
 **elif** i == 1:  
 print(**"Predicion: D0"**)  
 **elif** i == 0:  
 print(**"Predicion: F"**)  
 print(**"Accuracy:"**, sess.run(accuracy, feed\_dict={X: x\_test, Y: y\_test}))

1. 실행 화면

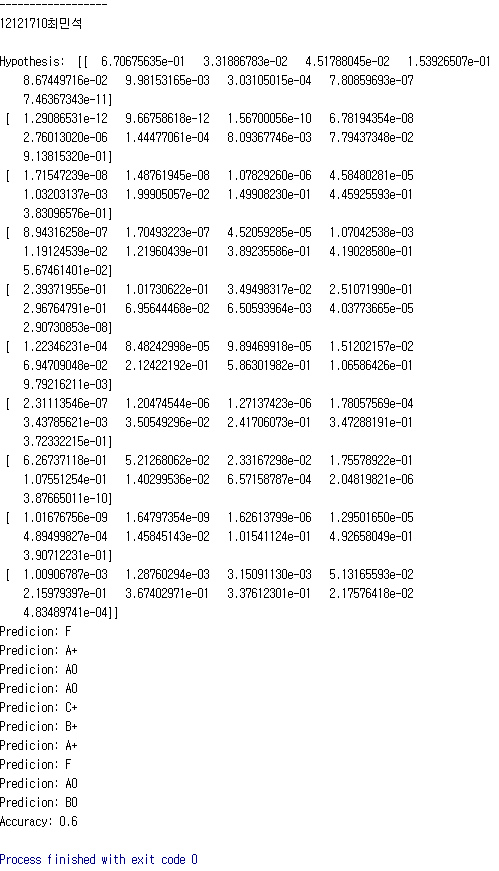
Multi-variable linear regression



sigmoid



softmax



1. 분석 및 결론

파이참을 이용해 텐서플로우를 사용하였습니다. 총 3가지 모델로 Multi-variable linear regression, sigmoid 그리고 softmax을 구현하였는데 각각 정확도가 0.4, 0.5, 0.6으로 나왔습니다. 정확도만 보면 매우 좋지 않지만, linear regression을 제외한 나머지 두 모델은 A+을 A0로 예측한 것, C+을 B0로 예측한 것같이 약간의 차이만 있을 뿐 거의 비슷하게 예측하였습니다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| test | F | A+ | A0 | A0 | C0 | B+ | A+ | C0 | A+ | C+ |
| Linear | F | A+ | A+ | A+ | F | A+ | A+ | F | A+ | B+ |
| sigmoid | F | A+ | A0 | A0 | F | B+ | A0 | F | A0 | B0 |
| softmax | F | A+ | A0 | A0 | C+ | B+ | A+ | F | A0 | B0 |