

With TF 1.0!

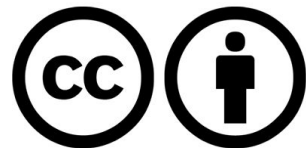


Lab 5

Logistic (regression) classifier

Sung Kim <hunkim+ml@gmail.com>

Code: <https://github.com/hunkim/DeepLearningZeroToAll/>



Call for comments

Please feel free to add comments directly on these slides

Other slides: <https://goo.gl/jPtVNt>



With TF 1.0!

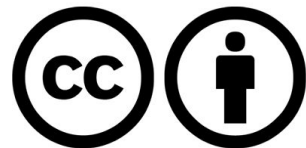


Lab 5

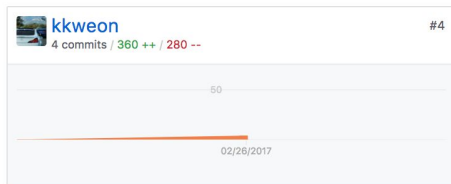
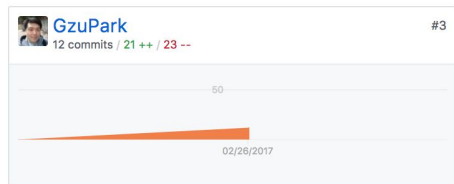
Logistic (regression) classifier

Sung Kim <hunkim+ml@gmail.com>

Code: <https://github.com/hunkim/DeepLearningZeroToAll/>



<https://github.com/hunkim/DeepLearningZeroToAll/>



Logistic Regression

$$H(X) = \frac{1}{1 + e^{-W^T X}}$$

$$cost(W) = -\frac{1}{m} \sum y \log(H(x)) + (1 - y) (\log(1 - H(x)))$$

$$W := W - \alpha \frac{\partial}{\partial W} cost(W)$$

Training Data

```
x_data = [[1, 2], [2, 3], [3, 1], [4, 3], [5, 3], [6, 2]]  
y_data = [[0], [0], [0], [1], [1], [1]]
```

```
# placeholders for a tensor that will be always fed.
```

```
X = tf.placeholder(tf.float32, shape=[None, 2])
```

```
Y = tf.placeholder(tf.float32, shape=[None, 1])
```

```

X = tf.placeholder(tf.float32, shape=[None, 2])
Y = tf.placeholder(tf.float32, shape=[None, 1])
W = tf.Variable(tf.random_normal([2, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

```

```

# Hypothesis using sigmoid: tf.div(1., 1. + tf.exp(tf.matmul(X, W) + b))

```

```

hypothesis = tf.sigmoid(tf.matmul(X, W) + b)

```

$$H(X) = \frac{1}{1 + e^{-W^T X}}$$

```

# cost/loss function

```

```

cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) *
                        tf.log(1 - hypothesis))

```

$$cost(W) = -\frac{1}{m} \sum y \log(H(x)) + (1 - y) (\log(1 - H(x)))$$

```

train = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)

```

$$W := W - \alpha \frac{\partial}{\partial W} cost(W)$$

```

# Accuracy computation

```

```

# True if hypothesis>0.5 else False

```

```

predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)

```

```

accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))

```

Train the model

Launch graph

```
with tf.Session() as sess:
```

Initialize TensorFlow variables

```
sess.run(tf.global_variables_initializer())
```

```
for step in range(10001):
```

```
    cost_val, _ = sess.run([cost, train], feed_dict={X: x_data, Y: y_data})
```

```
    if step % 200 == 0:
```

```
        print(step, cost_val)
```

Accuracy report

```
h, c, a = sess.run([hypothesis, predicted, accuracy],
```

```
                    feed_dict={X: x_data, Y: y_data})
```

```
print("\nHypothesis: ", h, "\nCorrect (Y): ", c, "\nAccuracy: ", a)
```



```

x_data = [[1, 2], [1, 2], [3, 1], [3, 1], [4, 3], [5, 3], [6, 2]]
y_data = [[0], [0], [0], [1], [1], [1]]
# placeholders for a tensor that will be always fed.
X = tf.placeholder(tf.float32, shape=[None, 2])
Y = tf.placeholder(tf.float32, shape=[None, 1])

W = tf.Variable(tf.random_normal([2, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')
# Hypothesis using sigmoid: tf.div(1., 1. + tf.exp(tf.matmul(X, W)))
hypothesis = tf.sigmoid(tf.matmul(X, W) + b)
# cost/loss function
cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) * tf.log(1 - hypothesis))
train = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)

# Accuracy computation
# True if hypothesis>0.5 else False
predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))

# Launch graph
with tf.Session() as sess:
    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())

    for step in range(10001):
        cost_val, _ = sess.run([cost, train], feed_dict={X: x_data, Y: y_data})
        if step % 200 == 0:
            print(step, cost_val)

    # Accuracy report
    h, c, a = sess.run([hypothesis, predicted, accuracy],
                        feed_dict={X: x_data, Y: y_data})
    print("\nHypothesis: ", h, "\nCorrect (Y): ", c, "\nAccuracy: ", a)

```

```

# step, cost
0 1.73078
200 0.571512
400 0.507414
...
9600 0.154132
9800 0.151778
10000 0.149496

```

```

Hypothesis:
[[ 0.03074029]
 [ 0.15884677]
 [ 0.30486736]
 [ 0.78138196]
 [ 0.93957496]
 [ 0.98016882]]

```

```

Correct (Y):
[[ 0.]
 [ 0.]
 [ 0.]
 [ 1.]
 [ 1.]
 [ 1.]]

```

Accuracy: 1.0

Classifying diabetes



| | | | | | | | | |
|------------|----------|-----------|------------|-----------|------------|-----------|-----------|---|
| -0.411765 | 0.165829 | 0.213115 | 0 | 0 | -0.23696 | -0.894962 | -0.7 | 1 |
| -0.647059 | -0.21608 | -0.180328 | -0.353535 | -0.791962 | -0.0760059 | -0.854825 | -0.833333 | 0 |
| 0.176471 | 0.155779 | 0 | 0 | 0 | 0.052161 | -0.952178 | -0.733333 | 1 |
| -0.764706 | 0.979899 | 0.147541 | -0.0909091 | 0.283688 | -0.0909091 | -0.931682 | 0.0666667 | 0 |
| -0.0588235 | 0.256281 | 0.57377 | 0 | 0 | 0 | -0.868488 | 0.1 | 0 |
| -0.529412 | 0.105528 | 0.508197 | 0 | 0 | 0.120715 | -0.903501 | -0.7 | 1 |
| 0.176471 | 0.688442 | 0.213115 | 0 | 0 | 0.132638 | -0.608027 | -0.566667 | 0 |
| 0.176471 | 0.396985 | 0.311475 | 0 | 0 | -0.19225 | 0.163962 | 0.2 | 1 |

```
xy = np.loadtxt('data-03-diabetes.csv', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]
```

```

xy = np.loadtxt('data-03-diabetes.csv', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]

# placeholders for a tensor that will be always fed.
X = tf.placeholder(tf.float32, shape=[None, 8])
Y = tf.placeholder(tf.float32, shape=[None, 1])

W = tf.Variable(tf.random_normal([8, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

# Hypothesis using sigmoid: tf.div(1., 1. + tf.exp(tf.matmul(X, W)))
hypothesis = tf.sigmoid(tf.matmul(X, W) + b)
# cost/loss function
cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) * tf.log(1 - hypothesis))
train = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)

# Accuracy computation
# True if hypothesis>0.5 else False
predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))

# Launch graph
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    feed = {X: x_data, Y: y_data}
    for step in range(10001):
        sess.run(train, feed_dict=feed)
        if step % 200 == 0:
            print(step, sess.run(cost, feed_dict=feed))

# Accuracy report
h, c, a = sess.run([hypothesis, predicted, accuracy], feed_dict=feed)
print("\nHypothesis: ", h, "\nCorrect (Y): ", c, "\nAccuracy: ", a)

```

```

0 0.82794
200 0.755181
400 0.726355
600 0.705179
800 0.686631
...
9600 0.492056
9800 0.491396
10000 0.490767

[ 0.7461012 ]
[ 0.79919308]
[ 0.72995949]
[ 0.88297188]]

[ 1.]
[ 1.]
[ 1.]
Accuracy:
0.762846

```

Exercise

- CSV reading using *tf.decode_csv*
- Try other classification data from Kaggle
 - <https://www.kaggle.com>

Lab 6

Softmax classifier

Sung Kim <hunkim+ml@gmail.com>