

With TF 1.0!



Lab 6

Softmax 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 6-I

Softmax Classifier

Sung Kim <hunkim+ml@gmail.com>

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



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



zeran4

1 commit / 5 ++ / 4 --

#11



jennykang

19 commits / 940 ++ / 253 --

#2



GzuPark

14 commits / 41 ++ / 31 --

#3



kkweon

5 commits / 372 ++ / 296 --

#4



BlueMelon715

4 commits / 45 ++ / 34 --

#5



jin-chong

2 commits / 4 ++ / 4 --

#6



FuZer

2 commits / 37 ++ / 30 --

#7



cynthia

1 commit / 28 ++ / 28 --

#8



keon

1 commit / 3 ++ / 3 --

#9

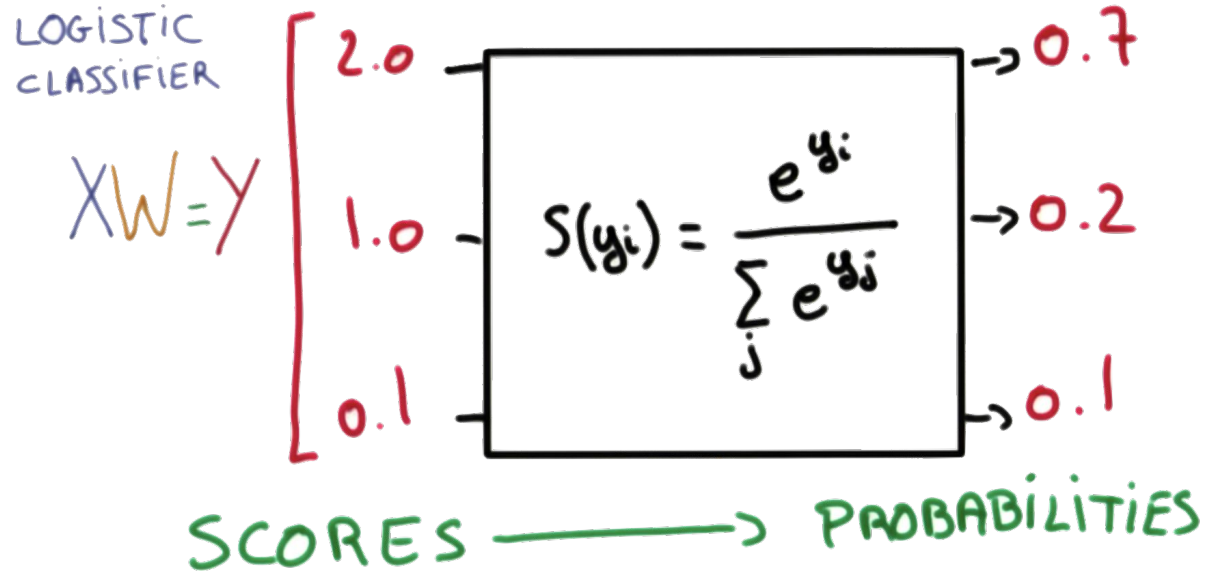


allieus

1 commit / 55 ++ / 59 --

#10

Softmax function

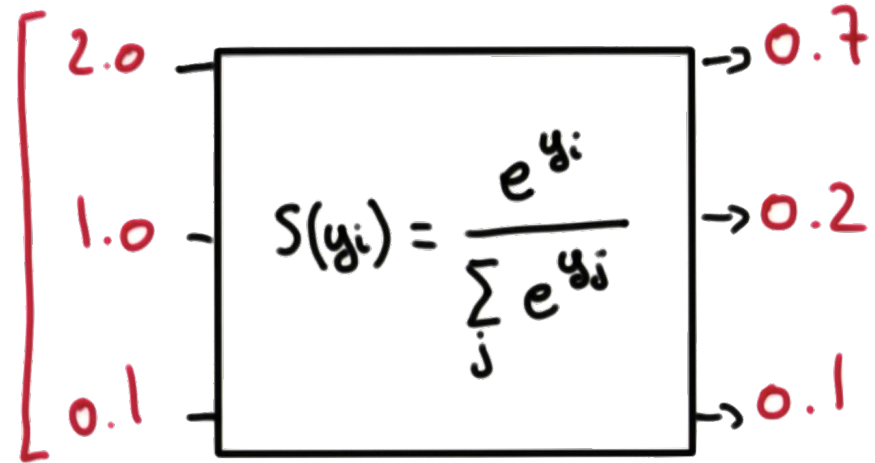


```
hypothesis = tf.nn.softmax(tf.matmul(X,W)+b)
```

```
tf.matmul(X,W)+b
```

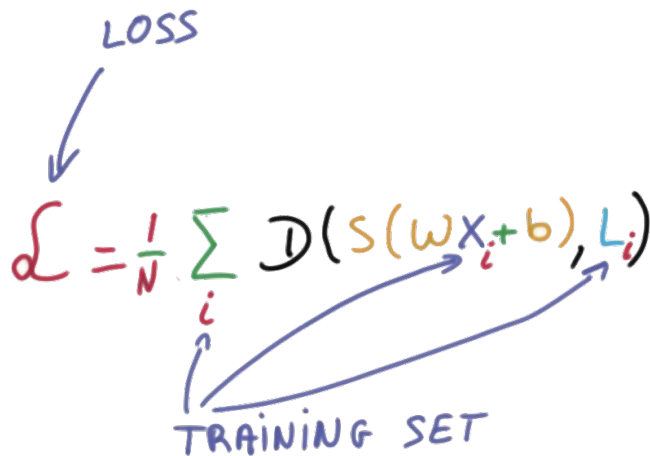
LOGISTIC
CLASSIFIER

$$XW=Y$$



SCORES ———> PROBABILITIES

Cost function: cross entropy



A handwritten diagram of the cross entropy loss formula. The formula is $\mathcal{L} = \frac{1}{N} \sum_i \mathcal{D}(s(w x_i + b), L_i)$. Annotations include: a blue arrow labeled 'LOSS' pointing to the \mathcal{L} ; a blue arrow labeled 'TRAINING SET' pointing to the index i in the summation; and two blue arrows pointing from the 'TRAINING SET' label to the x_i and L_i terms in the divergence function \mathcal{D} .

$$\mathcal{L} = \frac{1}{N} \sum_i \mathcal{D}(s(w x_i + b), L_i)$$

STEP



A handwritten expression for the derivative: $-\alpha \Delta \mathcal{L}(w_1, w_2)$. The term $\Delta \mathcal{L}(w_1, w_2)$ is underlined with a wavy line and labeled 'DERIVATIVE' below it.

$$-\alpha \Delta \mathcal{L}(w_1, w_2)$$

DERIVATIVE

Cross entropy cost/loss

```
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
```

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
```

```
x_data = [[1, 2, 1, 1], [2, 1, 3, 2], [3, 1, 3, 4], [4, 1, 5, 5], [1, 7, 5, 5],
          [1, 2, 5, 6], [1, 6, 6, 6], [1, 7, 7, 7]]
y_data = [[0, 0, 1], [0, 0, 1], [0, 0, 1], [0, 1, 0], [0, 1, 0], [0, 1, 0], [1, 0, 0], [1, 0, 0]]
```

```
X = tf.placeholder("float", [None, 4])
Y = tf.placeholder("float", [None, 3])
nb_classes = 3
```

```
W = tf.Variable(tf.random_normal([4, nb_classes]), name='weight')
b = tf.Variable(tf.random_normal([nb_classes]), name='bias')
```

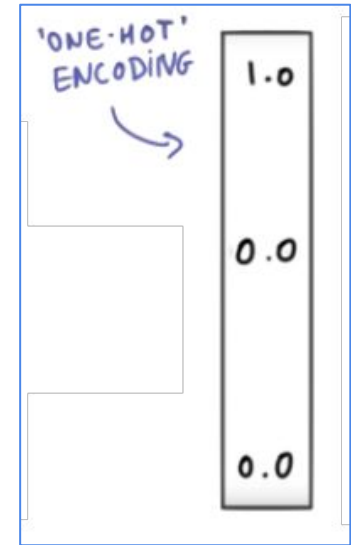
```
# tf.nn.softmax computes softmax activations
# softmax = exp(logits) / reduce_sum(exp(logits), dim)
hypothesis = tf.nn.softmax(tf.matmul(X, W) + b)
```

```
# Cross entropy cost/loss
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
```

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

```
for step in range(2001):
    sess.run(optimizer, feed_dict={X: x_data, Y: y_data})
    if step % 200 == 0:
        print(step, sess.run(cost, feed_dict={X: x_data, Y: y_data}))
```

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/lab-06-1-softmax_classifier.py



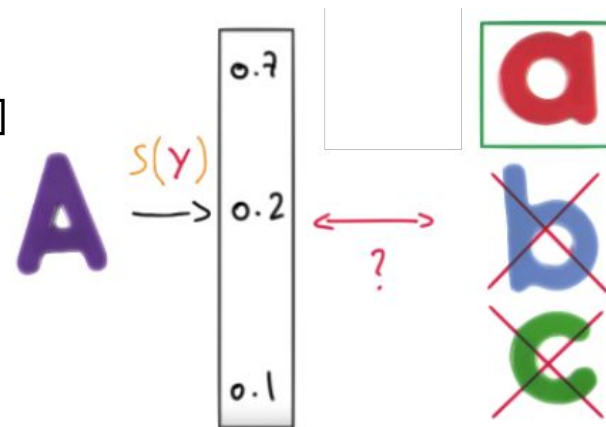
Test & one-hot encoding

```
hypothesis = tf.nn.softmax(tf.matmul(X,W)+b)
```

```
# Testing & One-hot encoding
```

```
a = sess.run(hypothesis, feed_dict={X: [[1, 11, 7, 9]]  
print(a, sess.run(tf.argmax(a, 1)))
```

```
[[ 1.38904958e-03  9.98601854e-01  9.06129117e-06]] [1]
```



Test & one-hot encoding

```
hypothesis = tf.nn.softmax(tf.matmul(X,W)+b)
```

```
all = sess.run(hypothesis, feed_dict={X: [[1, 11, 7, 9],  
                                           [1, 3, 4, 3],  
                                           [1, 1, 0, 1]]})  
print(all, sess.run(tf.argmax(all, 1)))
```

```
[[ 1.38904958e-03  9.98601854e-01  9.06129117e-06]  
 [ 9.31192040e-01  6.29020557e-02  5.90589503e-03]  
 [ 1.27327668e-08  3.34112905e-04  9.99665856e-01]]
```

```
[1 0 2]
```

With TF 1.0!



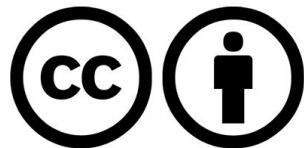
Lab 6-2

Fancy Softmax Classifier

cross_entropy, one_hot, reshape

Sung Kim <hunkim+ml@gmail.com>

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



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



zeran4

1 commit / 5 ++ / 4 --

#11



jennykang

19 commits / 940 ++ / 253 --

#2



GzuPark

14 commits / 41 ++ / 31 --

#3



kkweon

5 commits / 372 ++ / 296 --

#4



BlueMelon715

4 commits / 45 ++ / 34 --

#5



jin-chong

2 commits / 4 ++ / 4 --

#6



FuZer

2 commits / 37 ++ / 30 --

#7



cynthia

1 commit / 28 ++ / 28 --

#8



keon

1 commit / 3 ++ / 3 --

#9



allieus

1 commit / 55 ++ / 59 --

#10

softmax_cross_entropy_with_logits

```
logits = tf.matmul(X, W) + b  
hypothesis = tf.nn.softmax(logits)
```

1

Cross entropy cost/loss

```
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
```

2

Cross entropy cost/loss

```
cost_i = tf.nn.softmax_cross_entropy_with_logits(logits=logits,  
                                                  labels=Y_one_hot)
```

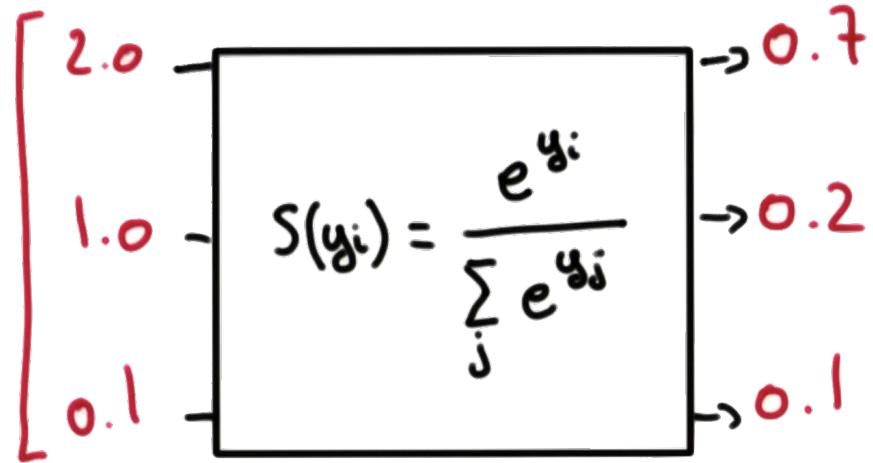
```
cost = tf.reduce_mean(cost_i)
```

```
hypothesis = tf.nn.softmax(tf.matmul(X,W))
```

```
tf.matmul(X,W)
```

LOGISTIC
CLASSIFIER

$$XW=Y$$



SCORES \longrightarrow PROBABILITIES

softmax_cross_entropy_with_logits

```
logits = tf.matmul(X, W) + b  
hypothesis = tf.nn.softmax(logits)
```

1

Cross entropy cost/loss

```
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
```

2


































Cross entropy cost/loss

```
cost_i = tf.nn.softmax_cross_entropy_with_logits(logits=logits,  
                                                  labels=Y_one_hot)
```

```
cost = tf.reduce_mean(cost_i)
```

Animal classification

with *softmax_cross_entropy_with_logits*

Birds	Insect	Fishes	Amphibians	Reptiles	Mammals
					
					
					
					
					
					
					

1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	0
0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	3
1	0	0	1	0	0	1	1	1	1	0	0	4	0	0	1	0
1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	0
1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	0
1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	0
1	0	0	1	0	0	0	1	1	1	0	0	4	1	1	1	0
0	0	1	0	0	1	0	1	1	0	0	1	0	1	1	0	3
0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	3
1	0	0	1	0	0	0	1	1	1	0	0	4	0	1	0	0
1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	0
0	1	1	0	1	0	0	0	1	1	0	0	2	1	1	0	1
0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	3
0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	6
0	0	1	0	0	1	1	0	0	0	0	0	4	0	0	0	6
0	0	1	0	0	1	1	0	0	0	0	0	6	0	0	0	6
0	1	1	0	1	0	1	0	1	1	0	0	2	1	0	0	1
1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	0

Predicting animal type based on various features

```
xy = np.loadtxt('data-04-zoo.csv', delimiter=',', dtype=np.float32)
```

```
x_data = xy[:, 0:-1]
```

```
y_data = xy[:, [-1]]
```


tf.one_hot and reshape

1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	0
0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	3
1	0	0	1	0	0	1	1	1	1	0	0	4	0	0	1	0
1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	0
1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	0
1	0	0	1	0	0	0	1	1	1	0	0	4	1	1	1	0
0	0	1	0	0	1	0	1	1	0	0	1	0	1	1	0	3
0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	3
1	0	0	1	0	0	0	1	1	1	0	0	4	0	1	0	0
1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	0
0	1	1	0	1	0	0	0	1	1	0	0	2	1	1	0	1
0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	3
0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	6
0	0	1	0	0	1	1	0	0	0	0	0	4	0	0	0	6
0	0	1	0	0	1	1	0	0	0	0	0	6	0	0	0	6
0	1	1	0	1	0	1	0	1	1	0	0	2	1	0	0	1
1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	0

```
Y = tf.placeholder(tf.int32, [None, 1]) # 0 ~ 6, shape=(?, 1)
Y_one_hot = tf.one_hot(Y, nb_classes) # one hot shape=(?, 1, 7)
Y_one_hot = tf.reshape(Y_one_hot, [-1, nb_classes]) # shape=(?, 7)
```

If the input indices is rank N, the output will have rank N+1. The new axis is created at dimension axis (default: the new axis is appended at the end).

https://www.tensorflow.org/api_docs/python/tf/one_hot

```
# Predicting animal type based on various features
xy = np.loadtxt('data-04-zoo.csv', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]
```

```
nb_classes = 7 # 0 ~ 6
```

```
X = tf.placeholder(tf.float32, [None, 16])
Y = tf.placeholder(tf.int32, [None, 1]) # 0 ~ 6
```

```
Y_one_hot = tf.one_hot(Y, nb_classes) # one hot
Y_one_hot = tf.reshape(Y_one_hot, [-1, nb_classes])
```

```
W = tf.Variable(tf.random_normal([16, nb_classes]), name='weight')
b = tf.Variable(tf.random_normal([nb_classes]), name='bias')
```

```
# tf.nn.softmax computes softmax activations
# softmax = exp(logits) / reduce_sum(exp(logits), dim)
logits = tf.matmul(X, W) + b
hypothesis = tf.nn.softmax(logits)
```

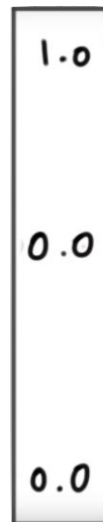
```
# Cross entropy cost/loss
```

```
cost_i = tf.nn.softmax_cross_entropy_with_logits(logits=logits,
                                                  labels=Y_one_hot)
```

```
cost = tf.reduce_mean(cost_i)
```

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
```

'ONE-HOT'
ENCODING



```
cost = tf.reduce_mean(cost_i)
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
```

```
prediction = tf.argmax(hypothesis, 1)
correct_prediction = tf.equal(prediction, tf.argmax(Y_one_hot, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

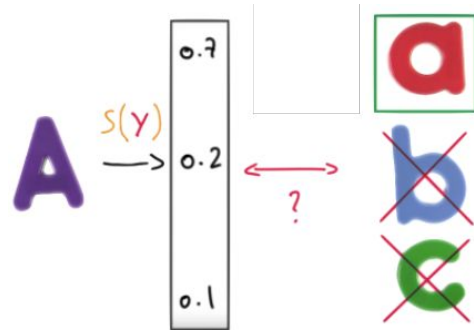
Launch graph

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

```
for step in range(2000):
    sess.run(optimizer, feed_dict={X: x_data, Y: y_data})
    if step % 100 == 0:
        loss, acc = sess.run([cost, accuracy], feed_dict={
            X: x_data, Y: y_data})
        print("Step: {:5}\tLoss: {:.3f}\tAcc: {:.2%}".format(
            step, loss, acc))
```

Let's see if we can predict

```
pred = sess.run(prediction, feed_dict={X: x_data})
# y_data: (N,1) = flatten => (N, ) matches pred.shape
for p, y in zip(pred, y_data.flatten()):
    print("[{}] Prediction: {} True Y: {}".format(p == int(y), p, int(y)))
```



```
cost = tf.reduce_mean(cost_i)
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
```

```
prediction = tf.argmax(hypothesis, 1)
correct_prediction = tf.equal(prediction, tf.argmax(Y_one_hot, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

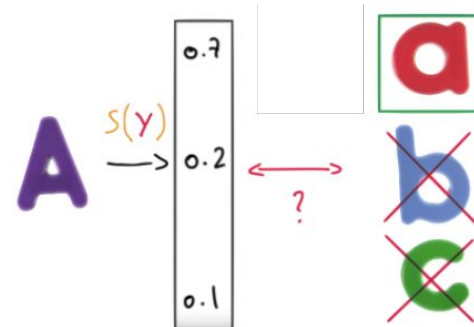
```
# Launch graph
```

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

```
for step in range(2000):
    sess.run(optimizer, feed_dict={X: x_data, Y: y_data})
    if step % 100 == 0:
        loss, acc = sess.run([cost, accuracy], feed_dict={
            X: x_data, Y: y_data})
        print("Step: {:5}\tLoss: {:.3f}\tAcc: {:.2%}".format(
            step, loss, acc))
```

```
# Let's see if we can predict
```

```
pred = sess.run(prediction, feed_dict={X: x_data})
# y_data: (N,1) = flatten => (N, ) matches pred.shape
for p, y in zip(pred, y_data.flatten()):
    print("[{}] Prediction: {} True Y: {}".format(
        p == int(y), p, int(y)))
```



```
Step: 1100 Loss: 0.101 Acc: 99.01%
Step: 1200 Loss: 0.092 Acc: 100.00%
Step: 1300 Loss: 0.084 Acc: 100.00%
```

```
...
```

```
[True] Prediction: 0 True Y: 0
[True] Prediction: 0 True Y: 0
[True] Prediction: 3 True Y: 3
[True] Prediction: 0 True Y: 0
[True] Prediction: 0 True Y: 0
[True] Prediction: 0 True Y: 0
[True] Prediction: 3 True Y: 3
[True] Prediction: 3 True Y: 3
[True] Prediction: 0 True Y: 0
```

Lab 7

Learning rate, Evaluation

Sung Kim <hunkim+ml@gmail.com>