

With TF 1.0!

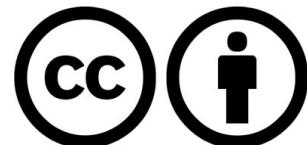


Lab 9

NN for XOR

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/jPtWNt>



With TF 1.0!

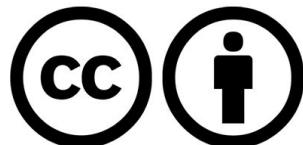


Lab 9-1

NN for XOR

Sung Kim <hunkim+ml@gmail.com>

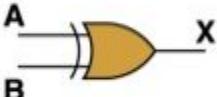
Code: <https://github.com/hunkim/DeepLearningZeroToAll/>



<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

XOR data set

Boolean Expression	Logic Diagram Symbol	Truth Table															
$X = A \oplus B$	 A standard logic symbol for an XOR gate. It consists of two input lines labeled 'A' and 'B' entering from the left, and one output line labeled 'X' exiting to the right. The symbol is yellow with a black outline.	<table border="1"><thead><tr><th>A</th><th>B</th><th>X</th></tr></thead><tbody><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></tbody></table>	A	B	X	0	0	0	0	1	1	1	0	1	1	1	0
A	B	X															
0	0	0															
0	1	1															
1	0	1															
1	1	0															

```
x_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)
y_data = np.array([0, 1, 1, 0], dtype=np.float32)
```

XOR with logistic regression?

```
x_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)
y_data = np.array([[0], [1], [1], [0]], dtype=np.float32)

X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)
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=0.1).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):
        sess.run(train, feed_dict={X: x_data, Y: y_data})
        if step % 100 == 0:
            print(step, sess.run(cost, feed_dict={X: x_data, Y: y_data}), sess.run(W))

    # Accuracy report
h, c, a = sess.run([hypothesis, predicted, accuracy], feed_dict={X: x_data, Y: y_data})
print("\nHypothesis: ", h, "\nCorrect: ", c, "\nAccuracy: ", a)      https://github.com/hunkim/DeepLearningZeroToAll/blob/master/lab-09-1-xor.py
```

```

x_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)
y_data = np.array([[0], [1], [1], [0]], dtype=np.float32)

X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)
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=0.1).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):
        sess.run(train, feed_dict={X: x_data, Y: y_data})
        if step % 100 == 0:
            print(step, sess.run(cost, feed_dict={X: x_data, Y: y_data}), sess.run(W))

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

```

XOR with
logistic regression?
But
it doesn't work!

Hypothesis:

[[0.5]
[0.5]
[0.5]
[0.5]]

Correct:

[[0.]
[0.]
[0.]
[0.]]

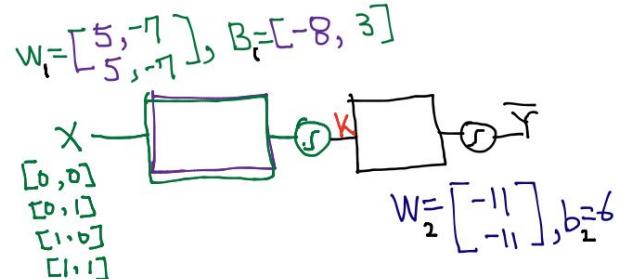
Accuracy: 0.5

<https://github.com/hunkim/DeepLearningZeroToAll/blob/master/lab-09-1-xor.py>

Neural Net

```
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)
```



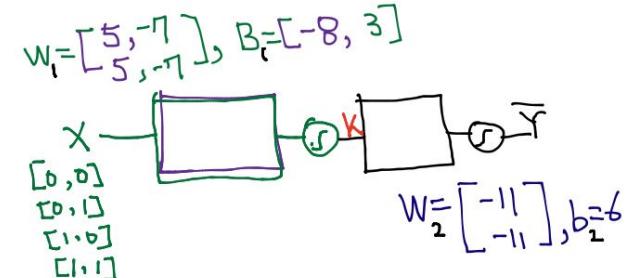
Neural Net

```
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)
```

```
w1 = tf.Variable(tf.random_normal([2, 2]), name='weight1')  
b1 = tf.Variable(tf.random_normal([2]), name='bias1')  
layer1 = tf.sigmoid(tf.matmul(X, w1) + b1)
```

```
w2 = tf.Variable(tf.random_normal([2, 1]), name='weight2')  
b2 = tf.Variable(tf.random_normal([1]), name='bias2')  
hypothesis = tf.sigmoid(tf.matmul(layer1, w2) + b2)
```



```

x_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)
y_data = np.array([[0], [1], [1], [0]], dtype=np.float32)
X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)

W1 = tf.Variable(tf.random_normal([2, 2]), name='weight1')
b1 = tf.Variable(tf.random_normal([2]), name='bias1')
layer1 = tf.sigmoid(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([2, 1]), name='weight2')
b2 = tf.Variable(tf.random_normal([1]), name='bias2')
hypothesis = tf.sigmoid(tf.matmul(layer1, W2) + b2)

# cost/Loss function
cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) * tf.log(1 - hypothesis))
train = tf.train.GradientDescentOptimizer(learning_rate=0.1).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):
        sess.run(train, feed_dict={X: x_data, Y: y_data})
        if step % 100 == 0:
            print(step, sess.run(cost, feed_dict={X: x_data, Y: y_data}), sess.run([W1, W2]))

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

```

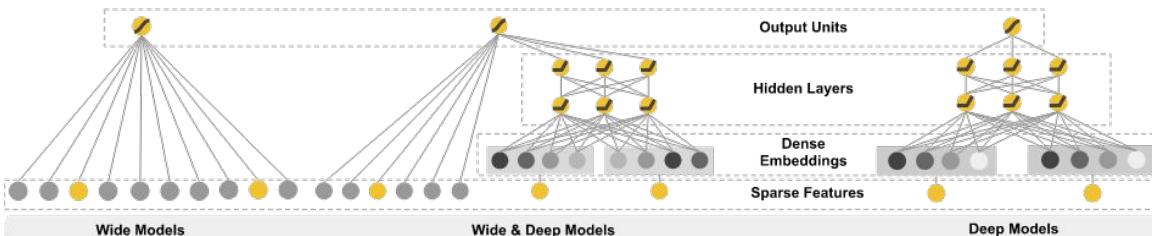
NN for XOR

Hypothesis:
[[0.01338218]
[0.98166394]
[0.98809403]
[0.01135799]]
Correct:
[[0.]
[1.]
[1.]
[0.]]
Accuracy: 1.0

Wide NN for XOR

```
W1 = tf.Variable(tf.random_normal([2, 10]), name='weight1')
b1 = tf.Variable(tf.random_normal([10]), name='bias1')
layer1 = tf.sigmoid(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([10, 1]), name='weight2')
b2 = tf.Variable(tf.random_normal([1]), name='bias2')
hypothesis = tf.sigmoid(tf.matmul(layer1, W2) + b2)
```



[2,10], [10,1]

Hypothesis:
[[0.00358802]
[0.99366933]
[0.99204296]
[0.0095663]]

Correct:

[[0.]
[1.]
[1.]
[0.]]

Accuracy: 1.0

[2,2], [2,1]

Hypothesis:
[[0.01338218]
[0.98166394]
[0.98809403]
[0.01135799]]

Correct:

[[0.]
[1.]
[1.]
[0.]]

Accuracy: 1.0

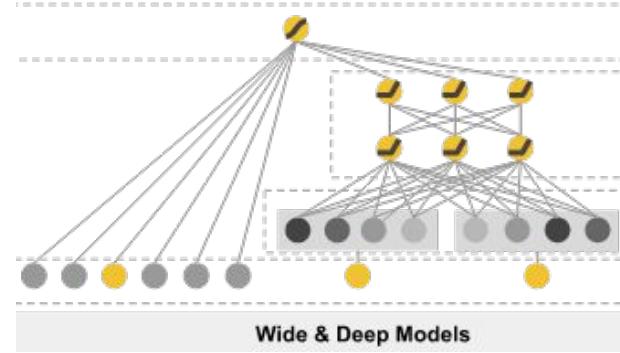
Deep NN for XOR

```
W1 = tf.Variable(tf.random_normal([2, 10]), name='weight1')
b1 = tf.Variable(tf.random_normal([10]), name='bias1')
layer1 = tf.sigmoid(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([10, 10]), name='weight2')
b2 = tf.Variable(tf.random_normal([10]), name='bias2')
layer2 = tf.sigmoid(tf.matmul(layer1, W2) + b2)

W3 = tf.Variable(tf.random_normal([10, 10]), name='weight3')
b3 = tf.Variable(tf.random_normal([10]), name='bias3')
layer3 = tf.sigmoid(tf.matmul(layer2, W3) + b3)

W4 = tf.Variable(tf.random_normal([10, 1]), name='weight4')
b4 = tf.Variable(tf.random_normal([1]), name='bias4')
hypothesis = tf.sigmoid(tf.matmul(layer3, W4) + b4)
```



4 layers

Hypothesis:
[[7.80e-04]
 [9.99e-01]
 [9.98e-01]
 [1.55e-03]]

Correct:

[[0.]
 [1.]
 [1.]
 [0.]]

Accuracy: 1.0

2 layers

Hypothesis:
[[0.01338218]
 [0.98166394]
 [0.98809403]
 [0.01135799]]

Correct:

[[0.]
 [1.]
 [1.]
 [0.]]

Accuracy: 1.0

Exercise

- Wide and Deep NN for MNIST



With TF 1.0!

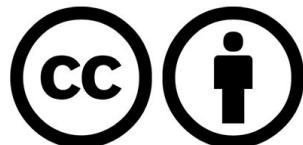


Lab 9-2

Tensorboard for XOR NN

Sung Kim <hunkim+ml@gmail.com>

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



TensorBoard: TF logging/debugging tool

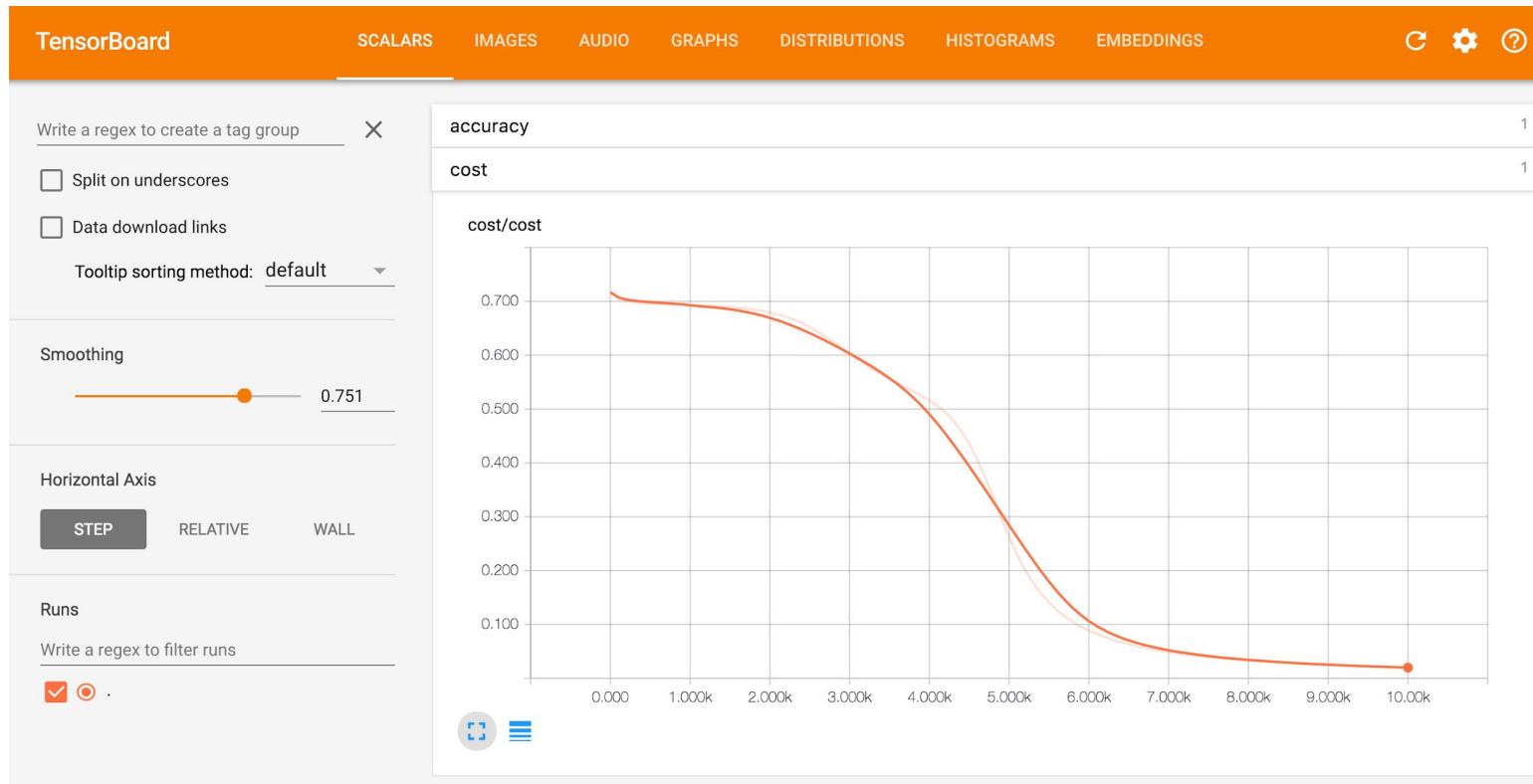
- Visualize your TF graph
- Plot quantitative metrics
- Show additional data



Old fashion: print, print, print

```
9400 0.0151413 [array([[ 6.21692038,  6.05913448],  
[-6.33773184, -5.75189114]], dtype=float32), array([[ 9.93581772],  
[-9.43034935]], dtype=float32)]  
9500 0.014909 [array([[ 6.22498751,  6.07049847],  
[-6.34637976, -5.76352596]], dtype=float32), array([[ 9.96414757],  
[-9.45942593]], dtype=float32)]  
9600 0.0146836 [array([[ 6.23292685,  6.08166742],  
[-6.35489035, -5.77496052]], dtype=float32), array([[ 9.99207973],  
[-9.48807526]], dtype=float32)]  
9700 0.0144647 [array([[ 6.24074268,  6.09264851],  
[-6.36326933, -5.78619957]], dtype=float32), array([[ 10.01962471],  
[-9.51631165]], dtype=float32)]  
9800 0.0142521 [array([[ 6.24843407,  6.10344648],  
[-6.37151814, -5.79724932]], dtype=float32), array([[ 10.04679298],  
[-9.54414845]], dtype=float32)]  
9900 0.0140456 [array([[ 6.25601053,  6.11406422],  
[-6.3796401 , -5.80811596]], dtype=float32), array([[ 10.07359505],  
[-9.57159519]], dtype=float32)]  
10000 0.0138448 [array([[ 6.26347113,  6.12451124],  
[-6.38764334, -5.81880617]], dtype=float32), array([[ 10.10004139],  
[-9.59866238]], dtype=float32)]
```

New way!



- 1 From TF graph, decide which tensors you want to log

```
w2_hist = tf.summary.histogram("weights2", W2)
cost_summ = tf.summary.scalar("cost", cost)
```

- 2 Merge all summaries

```
summary = tf.summary.merge_all()
```

- 3 Create writer and add graph

```
# Create summary writer
writer = tf.summary.FileWriter('./logs')
writer.add_graph(sess.graph)
```

- 4 Run summary merge and add_summary

```
s, _ = sess.run([summary, optimizer], feed_dict=feed_dict)
writer.add_summary(s, global_step=global_step)
```

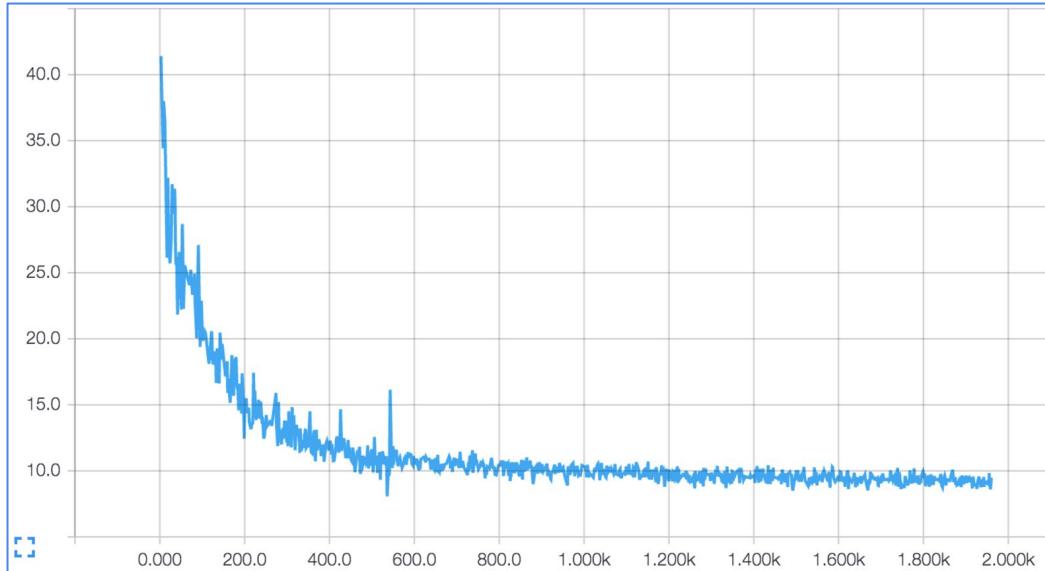
- 5 Launch TensorBoard

```
tensorboard --logdir=./logs
```

5 steps of using TensorBoard

Scalar tensors

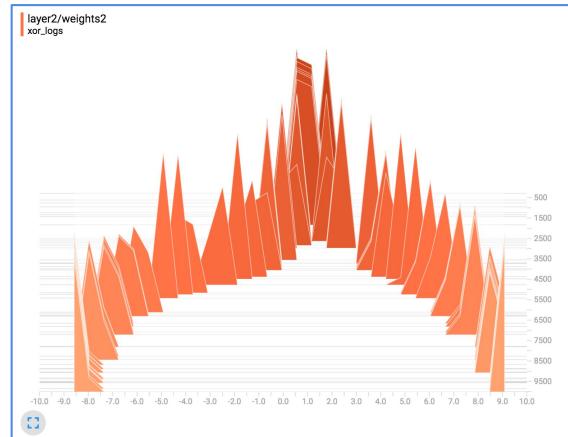
```
cost_summ = tf.summary.scalar("cost", cost)
```



Histogram (multi-dimensional tensors)

```
w2 = tf.Variable(tf.random_normal([2, 1]), name='weight2')  
b2 = tf.Variable(tf.random_normal([1]), name='bias2')  
hypothesis = tf.sigmoid(tf.matmul(layer1, w2) + b2)
```

```
w2_hist = tf.summary.histogram("weights2", w2)  
b2_hist = tf.summary.histogram("biases2", b2)  
hypothesis_hist = tf.summary.histogram("hypothesis", hypothesis)
```

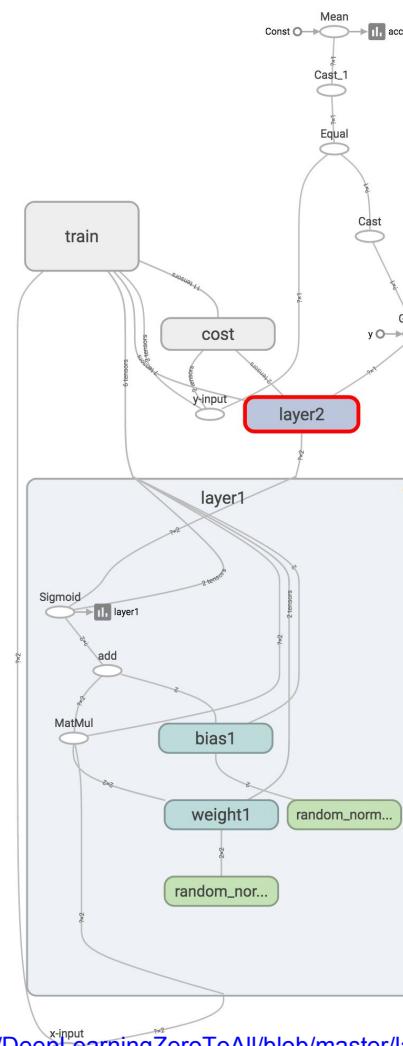


Add scope for better graph hierarchy

```
with tf.name_scope("layer1") as scope:  
    W1 = tf.Variable(tf.random_normal([2, 2]), name='weight1')  
    b1 = tf.Variable(tf.random_normal([2]), name='bias1')  
    layer1 = tf.sigmoid(tf.matmul(X, W1) + b1)
```

```
w1_hist = tf.summary.histogram("weights1", W1)  
b1_hist = tf.summary.histogram("biases1", b1)  
layer1_hist = tf.summary.histogram("layer1", layer1)
```

```
with tf.name_scope("layer2") as scope:  
    W2 = tf.Variable(tf.random_normal([2, 1]), name='weight2')  
    b2 = tf.Variable(tf.random_normal([1]), name='bias2')  
    hypothesis = tf.sigmoid(tf.matmul(layer1, W2) + b2)  
  
w2_hist = tf.summary.histogram("weights2", W2)  
b2_hist = tf.summary.histogram("biases2", b2)  
hypothesis_hist = tf.summary.histogram("hypothesis", hypothesis)
```



2 Merge summaries and create writer 3 after creating session

```
# Summary
summary = tf.summary.merge_all()

# initialize
sess = tf.Session()
sess.run(tf.global_variables_initializer())

# Create summary writer
writer = tf.summary.FileWriter(TB_SUMMARY_DIR)
writer.add_graph(sess.graph) # Add graph in the tensorboard
```

4

Run merged summary and write (add summary)

```
s, _ = sess.run([summary, optimizer], feed_dict=feed_dict)
writer.add_summary(s, global_step=global_step)
global_step += 1
```

5

Launch tensorflow (local)

```
writer = tf.summary.FileWriter("./logs/xor_logs")
```

```
$ tensorboard --logdir=./logs/xor_logs
```

```
Starting TensorBoard b'41' on port 6006  
(You can navigate to http://127.0.0.1:6006)
```

5

Launch tensorboard (remote server)

```
ssh -L local_port:127.0.0.1:remote_port username@server.com
```

```
local> $ ssh -L 7007:121.0.0.0:6006 hunkim@server.com  
server> $ tensorboard --logdir=./logs/xor_logs
```

(You can navigate to <http://127.0.0.1:7007>)

TensorBoard

SCALARS

IMAGES

AUDIO

GRAPHS

DISTRIBUTIONS

HISTOGRAMS

EMBEDDINGS

Write a regex to create a tag group X Split on underscores Data download links

Tooltip sorting method: default

Smoothing



Horizontal Axis

STEP

RELATIVE

WALL

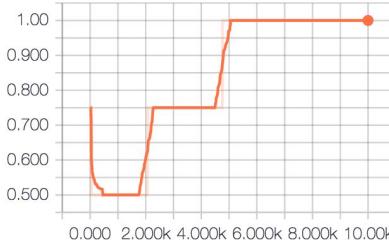
Runs

Write a regex to filter runs

 xor_logs xor_logs_r0_01

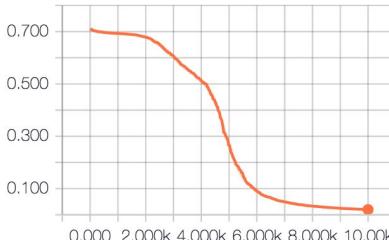
accuracy

accuracy



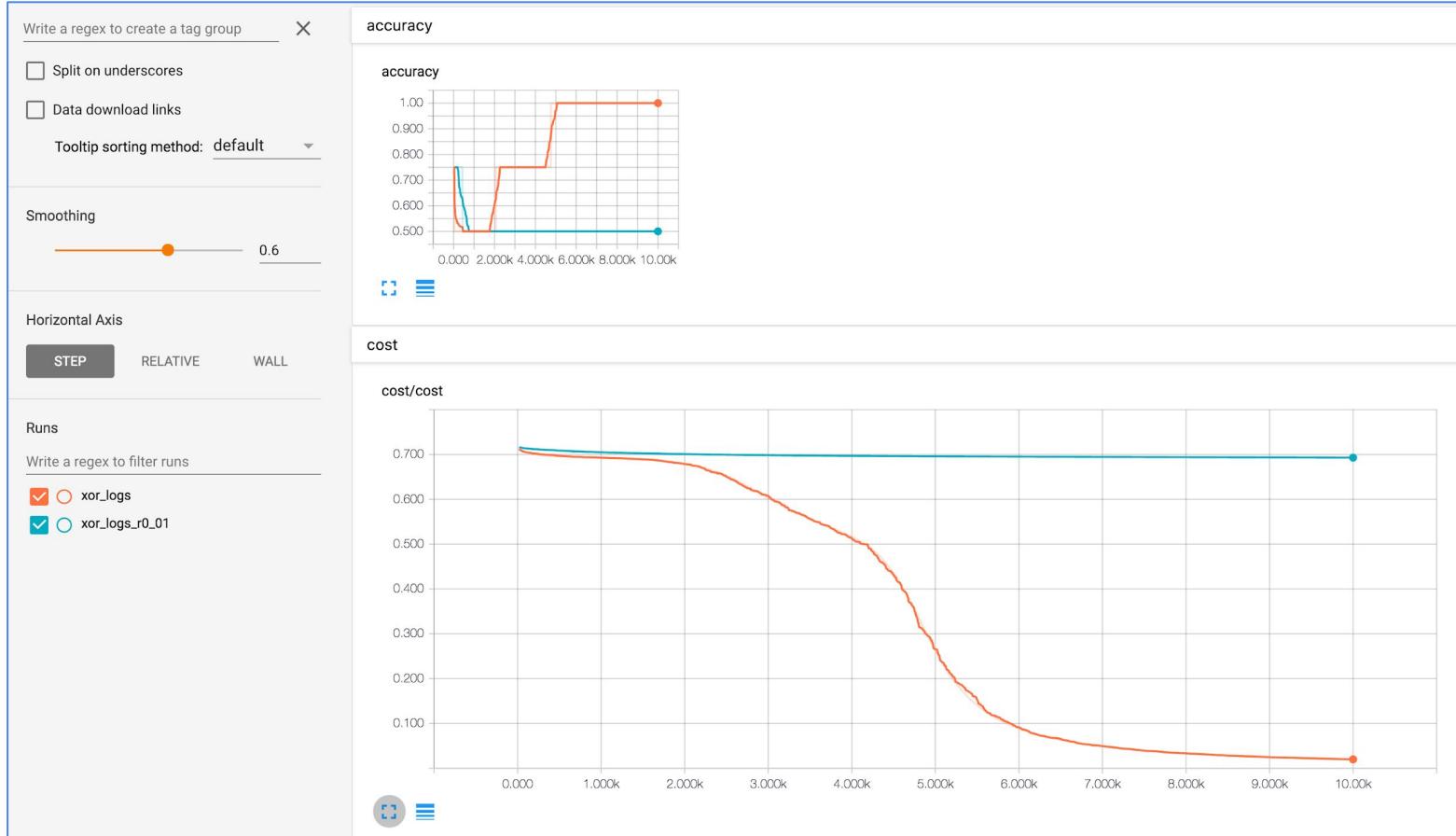
cost

cost/cost



Multiple runs

learning_rate=0.1 vs learning_rate=0.01



Multiple runs

```
tensorboard --logdir=./logs/xor_logs
```

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

```
...
```

```
    writer = tf.summary.FileWriter("./logs/xor_logs")
```

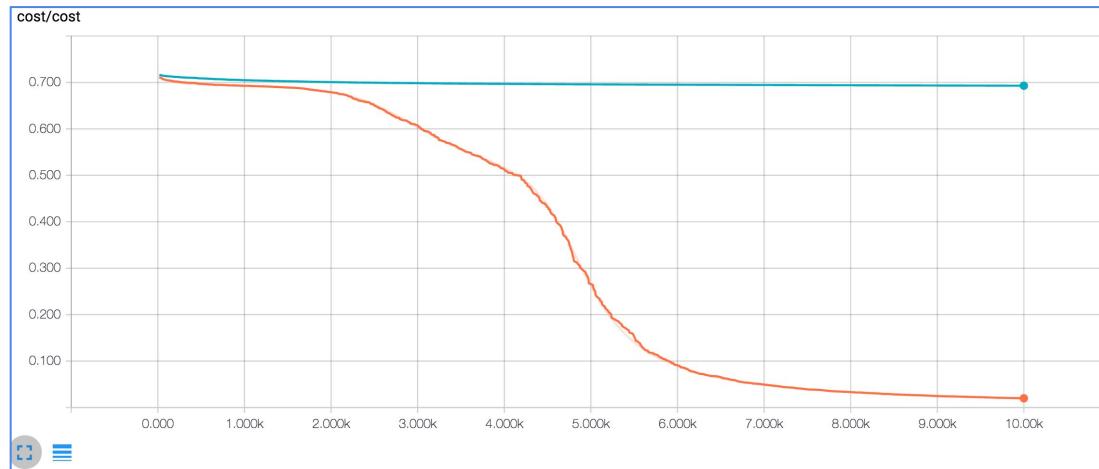
```
tensorboard --logdir=./logs/xor_logs_r0_01
```

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

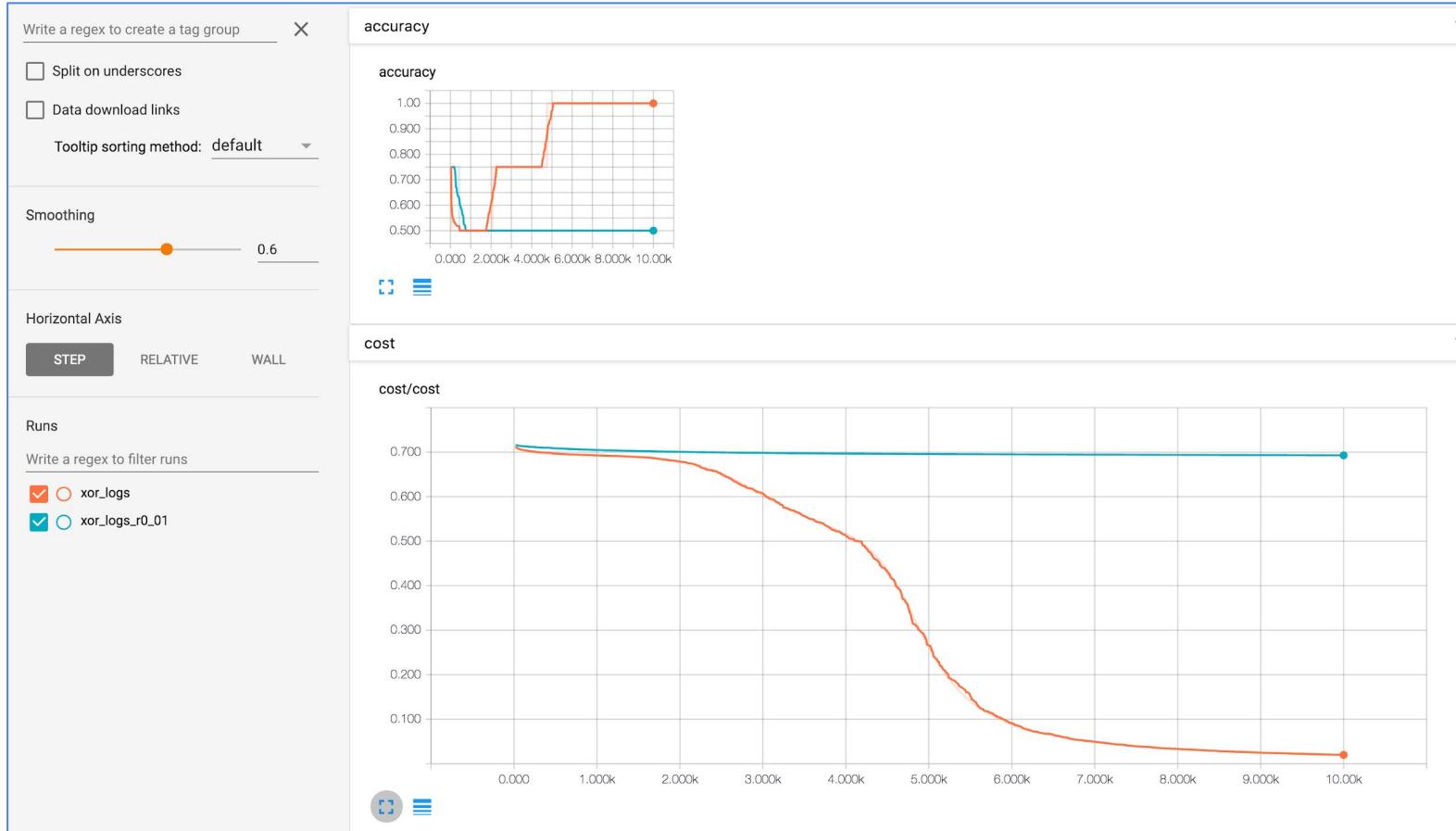
```
...
```

```
    writer = tf.summary.FileWriter("./logs/xor_logs_r0_01")
```

```
tensorboard --logdir=./logs
```



Multiple runs



- 1 From TF graph, decide which tensors you want to log

```
w2_hist = tf.summary.histogram("weights2", W2)  
cost_summ = tf.summary.scalar("cost", cost)
```

- 2 Merge all summaries

```
summary = tf.summary.merge_all()
```

- 3 Create writer and add graph

```
# Create summary writer  
writer = tf.summary.FileWriter('./logs')  
writer.add_graph(sess.graph)
```

- 4 Run summary merge and add_summary

```
s, _ = sess.run([summary, optimizer], feed_dict=feed_dict)  
writer.add_summary(s, global_step=global_step)
```

- 5 Launch TensorBoard

```
tensorboard --logdir=./logs
```

5 steps of using TensorBoard

Exercise

- Wide and Deep NN for MNIST
- Add tensorboard



With TF 1.0!

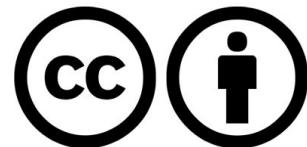


Lab 9-2-E

Tensorboard for MNIST

Sung Kim <hunkim+ml@gmail.com>

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

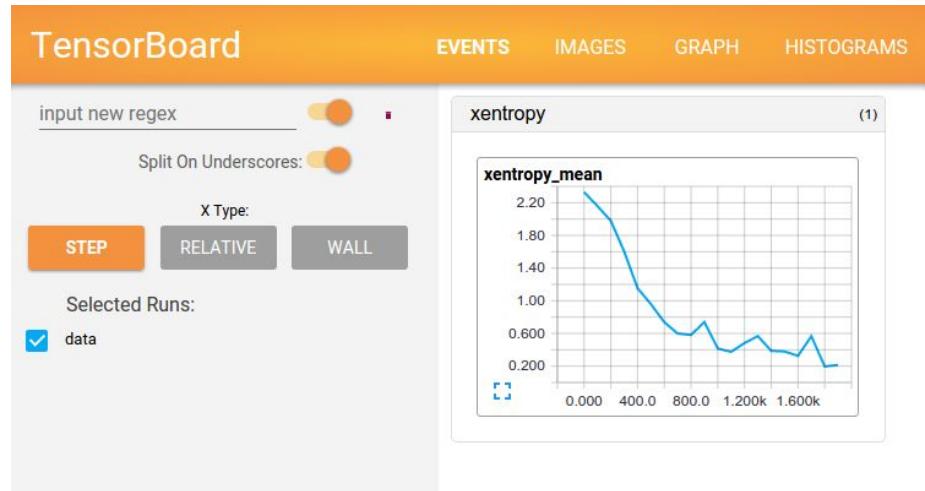


Visualizing your Deep learning using TensorBoard (TensorFlow)

Sung Kim <hunkim+ml@gmail.com>

TensorBoard: TF logging/debugging tool

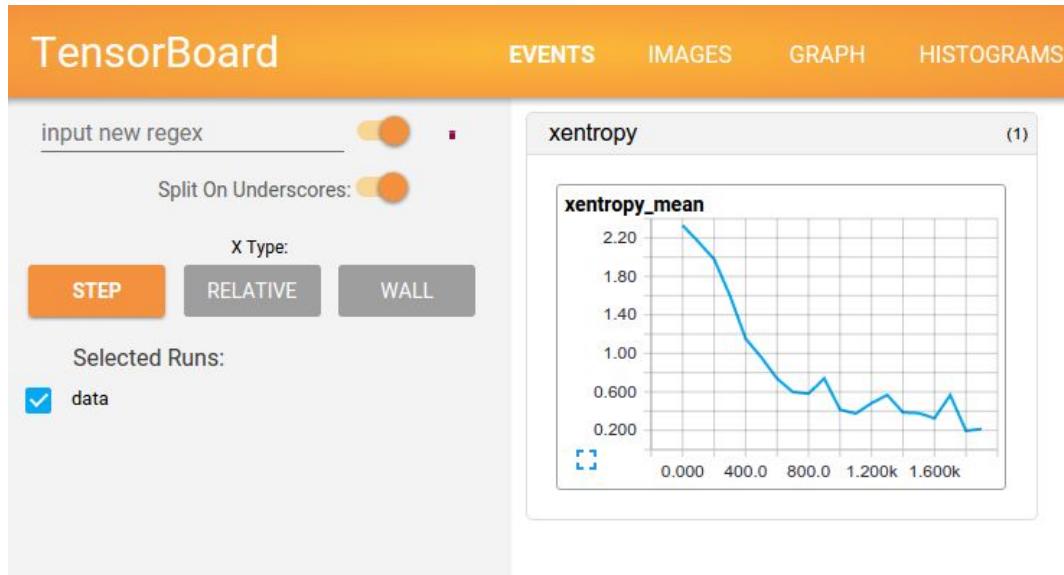
- Visualize your TF graph
- Plot quantitative metrics
- Show additional data



Old fashion: print, print, print

```
2000 [0.69364417, array([[ 0.50981331,  0.50592244],  
[ 0.37054271,  0.37088916],  
[ 0.6810087 ,  0.38607275],  
[ 0.54717511,  0.26581794]], dtype=float32), array([[ 0.50861073],  
[ 0.51602864],  
[ 0.4826754 ],  
[ 0.49036184]], dtype=float32), array([[ 0.71915275, -0.48754135],  
[-0.56914777, -0.55209494]], dtype=float32), array([[-0.44138899],  
[ 0.23536676]], dtype=float32), array([ 0.03925836,  0.02369077], dtype=float32), array([ 0.14039496], dtype=float32)]  
4000 [0.69332385, array([[ 0.52235132,  0.50927138],  
[ 0.38598102,  0.37814924],  
[ 0.69650716,  0.39592981],  
[ 0.56881481,  0.27748841]], dtype=float32), array([[ 0.50748861],  
[ 0.51554251],  
[ 0.48338425],  
[ 0.49113813]], dtype=float32), array([[ 0.74125487, -0.45954311],  
[-0.55370271, -0.53450096]], dtype=float32), array([[-0.42565805],  
[ 0.19686614]], dtype=float32), array([ 0.08946501,  0.03708982], dtype=float32), array([ 0.15204136], dtype=float32)]  
6000 [0.69306737, array([[ 0.53439337,  0.51197231],  
[ 0.39961013,  0.38383543],  
[ 0.71191686,  0.40380618],  
[ 0.58899951,  0.2868301 ]], dtype=float32), array([[ 0.50660294],  
[ 0.51538038],
```

New way!



- 1 From TF graph, decide which tensors you want to log

```
with tf.variable_scope('layer1') as scope:  
    tf.summary.image('input', x_image, 3)  
    tf.summary.histogram("layer", L1)  
    tf.summary.scalar("loss", cost)
```

- 2 Merge all summaries

```
summary = tf.summary.merge_all()
```

- 3 Create writer and add graph

```
# Create summary writer  
writer = tf.summary.FileWriter(TB_SUMMARY_DIR)  
writer.add_graph(sess.graph)
```

- 4 Run summary merge and add_summary

```
s, _ = sess.run([summary, optimizer], feed_dict=feed_dict)  
writer.add_summary(s, global_step=global_step)
```

- 5 Launch TensorBoard

```
tensorboard --logdir=/tmp/mnist_logs
```

5 steps of using TensorBoard

Image Input

```
# Image input  
x_image = tf.reshape(X, [-1, 28, 28, 1])  
tf.summary.image('input', x_image, 3)
```

TensorBoard

SCALARS IMAGES AUDIO GRAPHS DISTRIBUTIONS HISTOGRAMS EMBEDDINGS

Write a regex to create a tag group X

Split on underscores

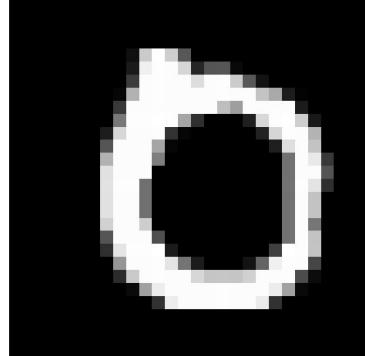
Runs

Write a regex to filter runs

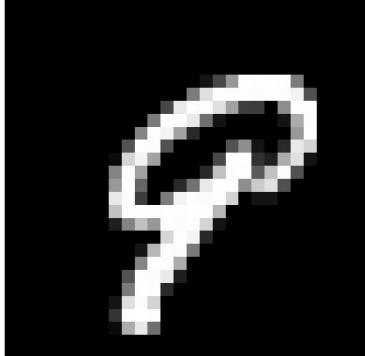
mnist

input

input/image/0
mnist
step 106 (Tue Feb 28 2017 08:30:35 GMT+0800 (HKT))



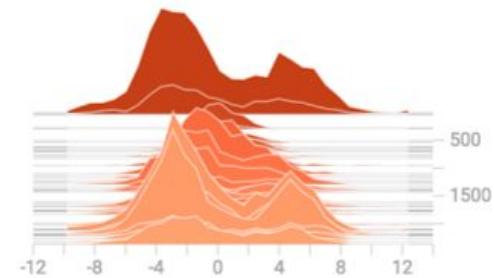
input/image/1
mnist
step 106 (Tue Feb 28 2017 08:30:35 GMT+0800 (HKT))



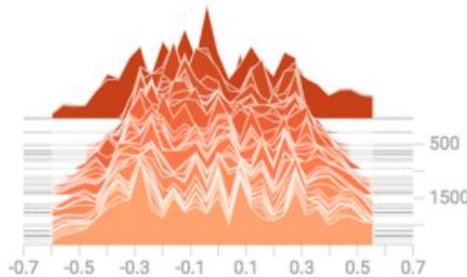
Histogram (multi-dimensional tensors)

```
with tf.variable_scope('layer1') as scope:  
    W1 = tf.get_variable("W", shape=[784, 512])  
    b1 = tf.Variable(tf.random_normal([512]))  
    L1 = tf.nn.relu(tf.matmul(X, W1) + b1)  
    L1 = tf.nn.dropout(L1, keep_prob=keep_prob)  
  
tf.summary.histogram("X", X)  
tf.summary.histogram("weights", W1)  
tf.summary.histogram("bias", b1)  
tf.summary.histogram("layer", L1)
```

layer2/layer
.

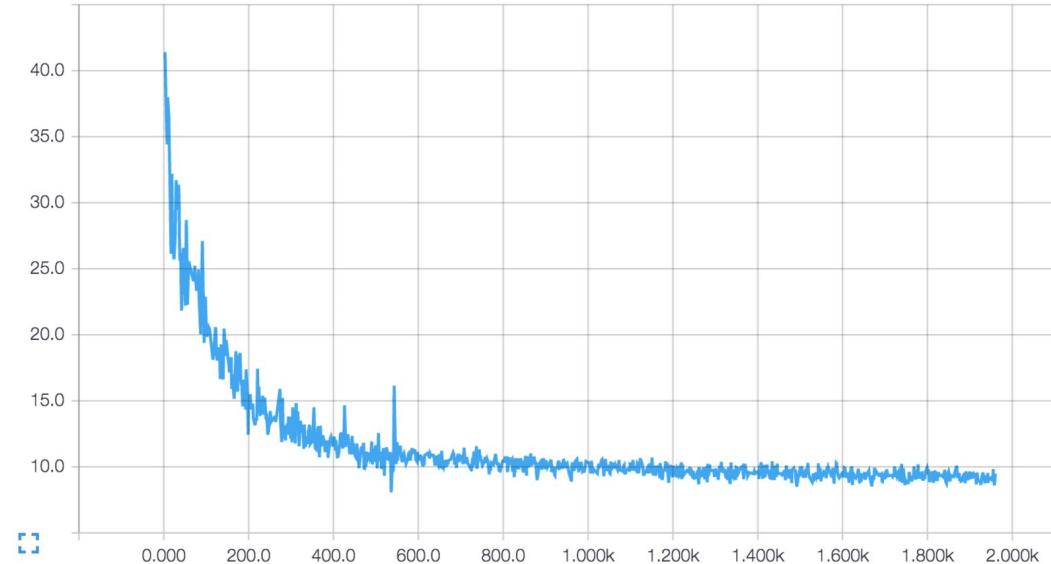


layer2/weights
.



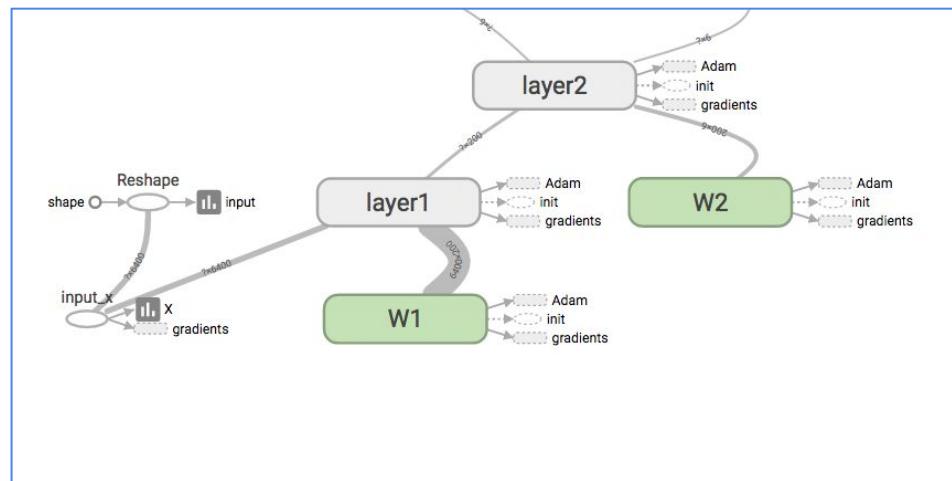
Scalar tensors

```
tf.summary.scalar("loss", cost)
```



Add scope for better hierarchy

```
with tf.variable_scope('layer1') as scope:  
    W1 = tf.get_variable("W", shape=[784, 512], ...  
    b1 = tf.Variable(tf.random_normal([512]))  
    L1 = tf.nn.relu(tf.matmul(X, W1) + b1)  
    L1 = tf.nn.dropout(L1, keep_prob=keep_prob)  
  
tf.summary.histogram("X", X)  
tf.summary.histogram("weights", W1)  
tf.summary.histogram("bias", b1)  
tf.summary.histogram("layer", L1)  
  
with tf.variable_scope('layer2') as scope:  
    ...  
  
with tf.variable_scope('layer3') as scope:  
    ...  
  
with tf.variable_scope('layer4') as scope:  
    ...  
  
with tf.variable_scope('layer5') as scope:  
    ...
```



2

Merge summaries and create writer after creating session

```
# Summary
summary = tf.summary.merge_all()

# initialize
sess = tf.Session()
sess.run(tf.global_variables_initializer())

# Create summary writer
writer = tf.summary.FileWriter(TB_SUMMARY_DIR)
writer.add_graph(sess.graph)
```

Run merged summary and write (add summary)

```
s, _ = sess.run([summary, optimizer], feed_dict=feed_dict)
writer.add_summary(s, global_step=global_step)
global_step += 1
```

Launch tensorflow (local)

```
writer = tf.summary.FileWriter("/tmp/mnist_logs")
```

```
$ tensorboard --logdir=/tmp/mnist_logs
```

Starting TensorBoard b'41' on port 6006
(You can navigate to <http://127.0.0.1:6006>)

Launch tensorboard (remote server)

```
ssh -L local_port:127.0.0.1:remote_port username@server.com
```

```
local> $ ssh -L 7007:127.0.0.1:6006 hunkim@server.com  
server> $ tensorboard --logdir=/tmp/mnist_logs
```

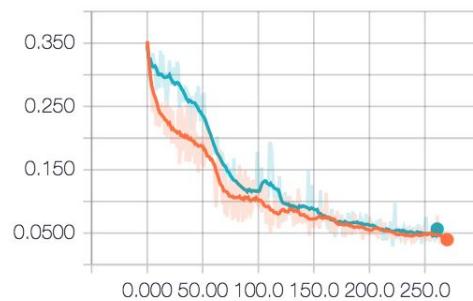
(You can navigate to <http://127.0.0.1:7007>)

Multiple runs

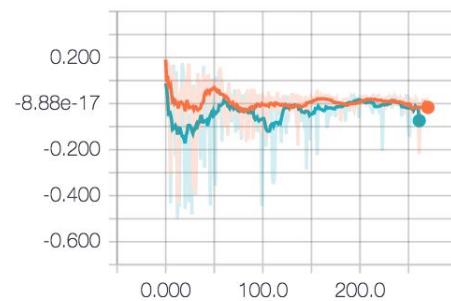
```
tensorboard --logdir=/tmp/mnist_logs/run1  
writer = tf.summary.FileWriter("/tmp/mnist_logs/run1")  
tensorboard --logdir=/tmp/mnist_logs/run2  
writer = tf.summary.FileWriter("/tmp/mnist_logs/run1")
```

tensorboard --logdir=/tmp/mnist_logs

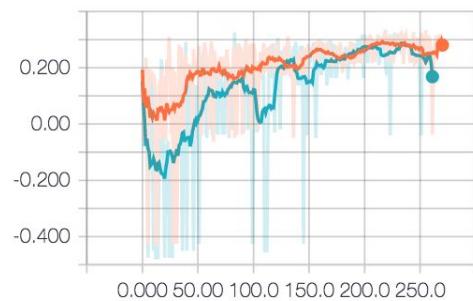
log_likelihood



loss



reward_1



- 1 From TF graph, decide which tensors you want to log

```
with tf.variable_scope('layer1') as scope:  
    tf.summary.image('input', x_image, 3)  
    tf.summary.histogram("layer", L1)  
    tf.summary.scalar("loss", cost)
```

- 2 Merge all summaries

```
summary = tf.summary.merge_all()
```

- 3 Create writer and add graph

```
# Create summary writer  
writer = tf.summary.FileWriter(TB_SUMMARY_DIR)  
writer.add_graph(sess.graph)
```

- 4 Run summary merge and add_summary

```
s, _ = sess.run([summary, optimizer], feed_dict=feed_dict)  
writer.add_summary(s, global_step=global_step)
```

- 5 Launch TensorBoard

```
tensorboard --logdir=/tmp/mnist_logs
```

5 steps of using TensorBoard

With TF 1.0!

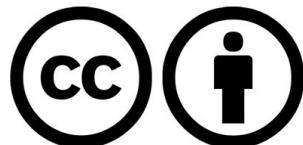


Lab 9-3 (optional)

NN Backpropagation

Sung Kim <hunkim+ml@gmail.com>

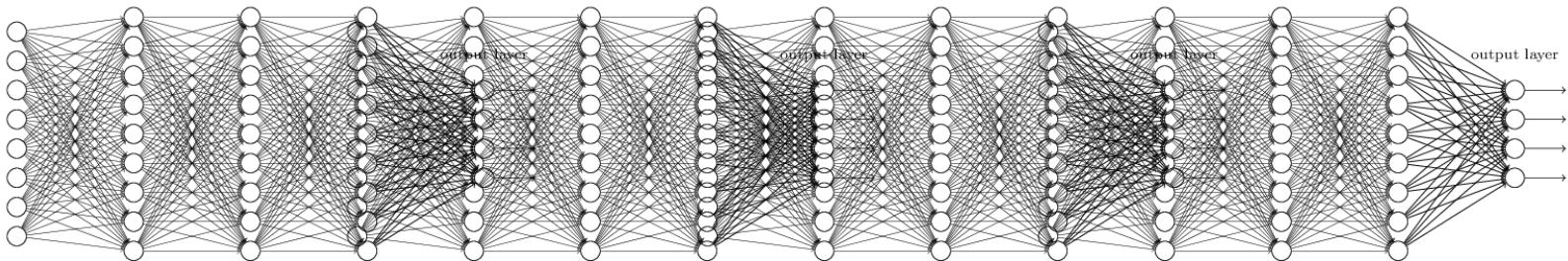
Code: <https://github.com/hunkim/DeepLearningZeroToAll/>



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

 sxjscience 1 commit / 350 ++ / 0 --	#12	 jennykang 19 commits / 940 ++ / 253 --	#2	 GzuPark 15 commits / 49 ++ / 39 --	#3	 kkweon 12 commits / 1,087 ++ / 340 --	#4
 BlueMelon715 5 commits / 56 ++ / 44 --	#5	 jihobak 3 commits / 244 ++ / 1,289 --	#6	 FuZer 2 commits / 37 ++ / 30 --	#7	 jin-chong 2 commits / 4 ++ / 4 --	#8
 zeran4 1 commit / 5 ++ / 4 --	#9	 cynthia 1 commit / 28 ++ / 28 --	#10	 kaka120011 1 commit / 1 ++ / 1 --	#11	 keon 1 commit / 3 ++ / 3 --	#12
 allieus 1 commit / 55 ++ / 59 --	#13	 togheppi 1 commit / 1,280 ++ / 0 --	#14	 davinovation 1 commit / 64 ++ / 0 --	#15	 skyer9 1 commit / 94 ++ / 69 --	#16
 redongjun 1 commit / 78 ++ / 0 --	#17	 blkRyusim 1 commit / 1 ++ / 1 --	#18	 maestrojeong 1 commit / 50 ++ / 0 --	#19	 mjc92 1 commit / 1,215 ++ / 0 --	#13

How to train?



Gradient descent algorithm

$$W := W - \alpha \frac{1}{m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)})x^{(i)}$$

$$w = w - \alpha \frac{\partial E}{\partial w}$$

Tensorflow Magic!

$$w = w - \alpha \frac{\partial E}{\partial w}$$



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

“Yes you should understand backprop”

- “If you try to ignore how it works under the hood because *TensorFlow automagically makes my networks learn*”
 - “You will not be ready to wrestle with the dangers it presents”
 - “You will be much less effective at building and debugging neural networks.”
- “The good news is that backpropagation is not that difficult to understand”
 - “if presented properly.”

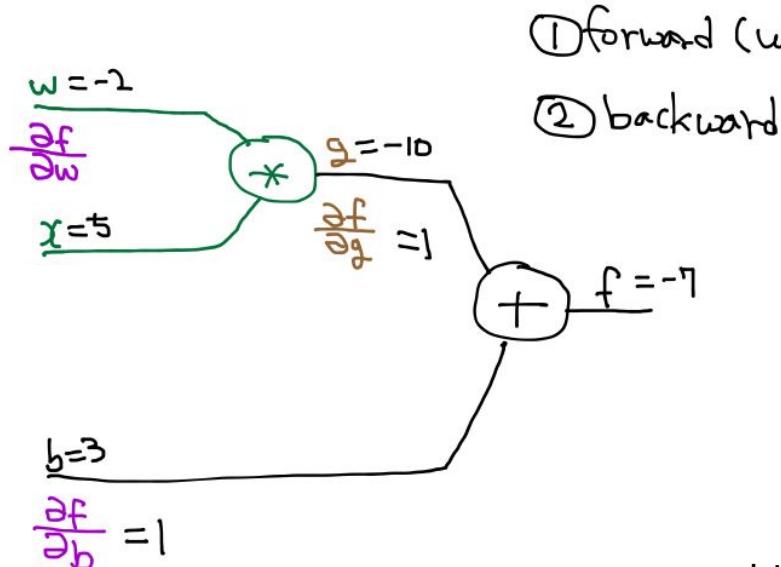
Back propagation (chain rule)

$$f = wx + b, \quad g = wx, \quad f = g + b$$

$\frac{\partial f}{\partial g} = 1, \quad \frac{\partial f}{\partial b} = 1$

$$\frac{\partial g}{\partial w} = x$$

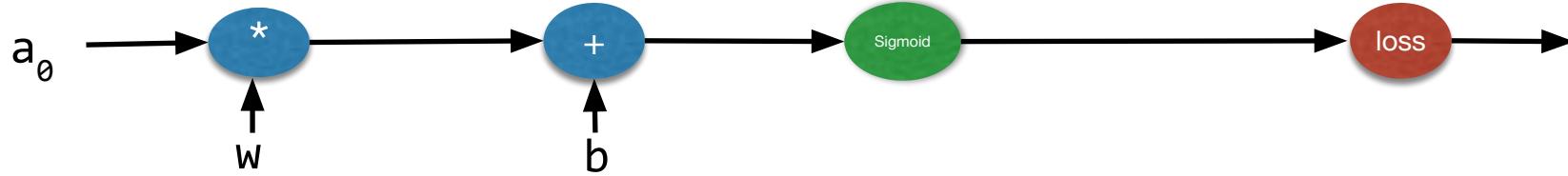
$$\frac{\partial f}{\partial w} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial w} = 1 * x = 5$$



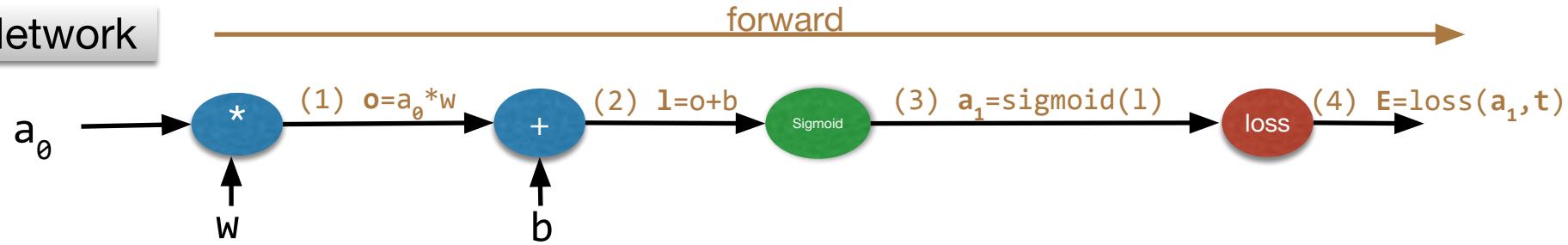
Back propagation (chain rule)

$$\frac{\partial f}{\partial w} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial w} = 1 * 5$$

Logistic Regression Network



Network



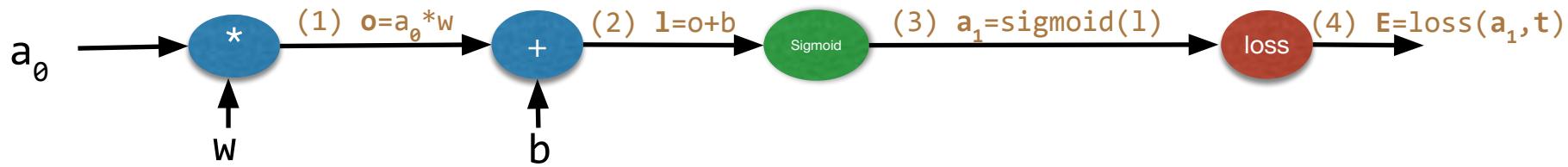
$$a_1 = sigmoid(a_0 * w + b)$$

$$E = loss(a_1, t) = - \sum t * log(a_1) + (1 - t) * log(1 - a_1)$$

Forward pass, OK? Just follow (1), (2), (3) and (4)

Network

forward



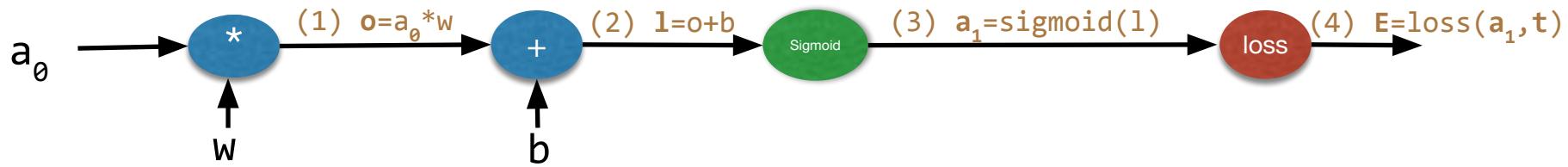
$$a_1 = \text{sigmoid}(a_0 * w + b)$$

$$E = \text{loss}(a_1, t) = - \sum t * \log(a_1) + (1 - t) * \log(1 - a_1)$$

$$a_0 = 0.1, t = 0, w = 0.1, b = 0.3$$

Network

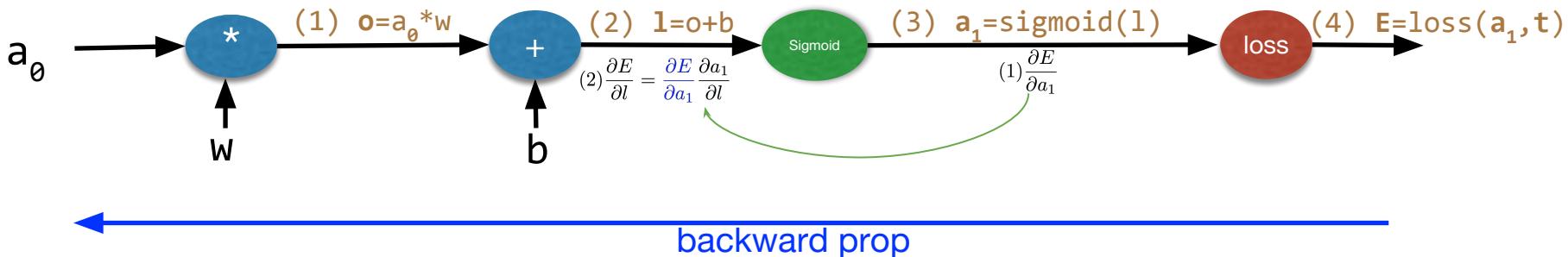
forward



$$w = w - \alpha \frac{\partial E}{\partial w}$$

Network

forward



$$w = w - \alpha \frac{\partial E}{\partial w}$$

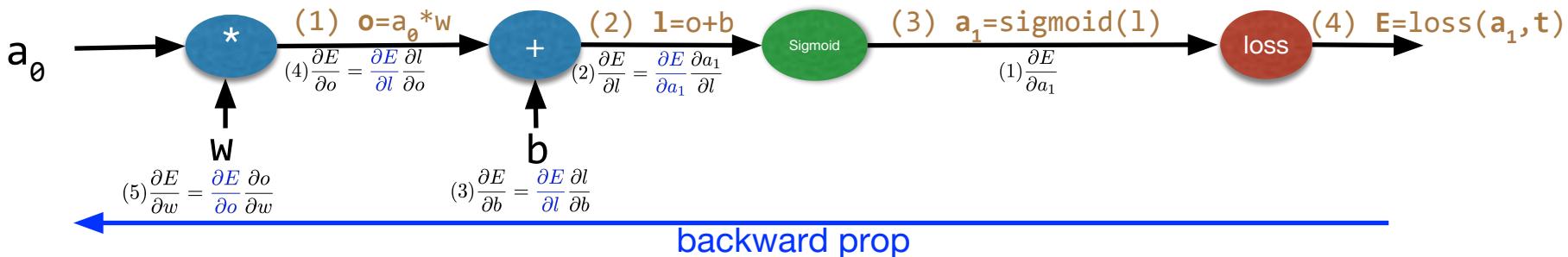
Let's do back propagation!

$(1) \frac{\partial E}{\partial a_1}$ will be given. What would be $(2) \frac{\partial E}{\partial l}$?

We can use the chain rule.

Network

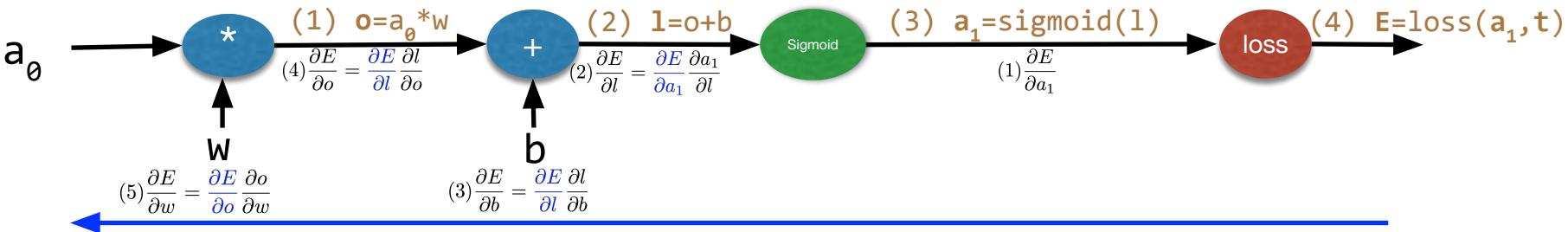
forward



In the same manner, we can get back prop (4), (3), (1) and (1)!

Network

forward



Gate derivatives

$$o = aw, \quad \frac{\partial o}{\partial w} = a, \quad \frac{\partial o}{\partial a} = w$$

$$l = o + b, \quad \frac{\partial l}{\partial o} = 1, \quad \frac{\partial l}{\partial b} = 1$$

$$E = - \sum t \log(o) + (1-t) \log(1-o), \quad \frac{\partial E}{\partial o} = \frac{a-t}{a(1-a)}$$

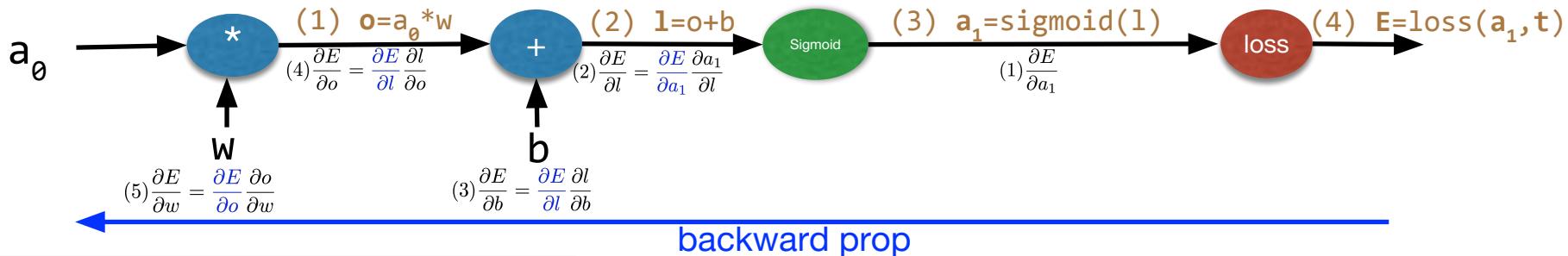
$$a = \text{sigmoid}(l) = \frac{1}{1+e^{-l}}, \quad \frac{\partial a}{\partial l} = a(1-a)$$

These derivatives for each gate will be given.

We can just use them in the chain rule.

Network

forward



Derivatives (chain rule)

$$(1) \frac{\partial E}{\partial a_1} = \frac{a_1 - t}{a_1(1 - a_1)}$$

$$(2) \frac{\partial E}{\partial l} = \frac{\partial E}{\partial a_1} \frac{\partial a_1}{\partial l}$$

Given from the pre computed derivative

Gate derivatives

$$o = aw, \quad \frac{\partial o}{\partial w} = a, \quad \frac{\partial o}{\partial a} = w$$

$$l = o + b, \quad \frac{\partial l}{\partial o} = 1, \quad \frac{\partial l}{\partial b} = 1$$

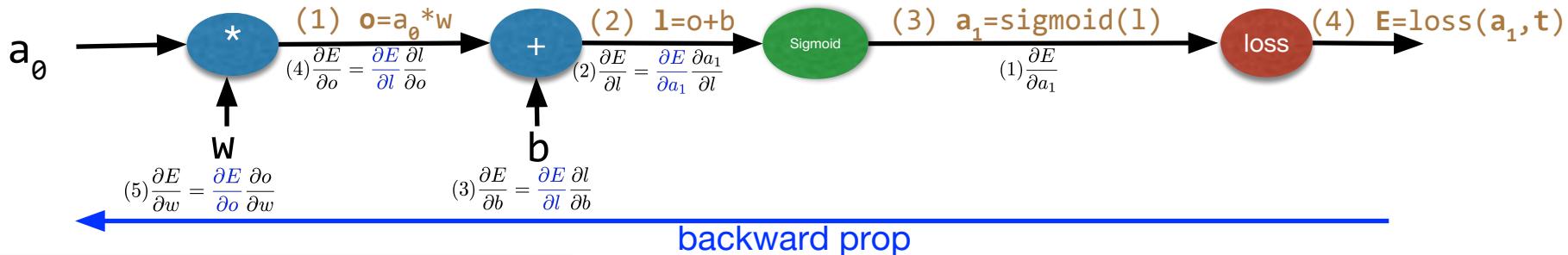
$$E = - \sum t \log(o) + (1-t) \log(1-o), \quad \frac{\partial E}{\partial a} = \frac{a-t}{a(1-a)}$$

$$a = \text{sigmoid}(l) = \frac{1}{1+e^{-l}}, \quad \frac{\partial a}{\partial l} = a(1-a)$$

Just apply them one by one and solve each derivative one by one!

Network

forward



Derivatives (chain rule)

$$(1) \frac{\partial E}{\partial a_1} = \frac{a_1 - t}{a_1(1 - a_1)}$$

$$(2) \frac{\partial E}{\partial l} = \frac{\partial E}{\partial a_1} \frac{\partial a_1}{\partial l} = \frac{a_1 - t}{a_1(1 - a_1)} * a_1(1 - a_1) = a_1 - t$$

Given from the pre computed derivative

Given

Gate derivatives

$$o = aw, \quad \frac{\partial o}{\partial w} = a, \quad \frac{\partial o}{\partial a} = w$$

$$l = o + b, \quad \frac{\partial l}{\partial o} = 1, \quad \frac{\partial l}{\partial b} = 1$$

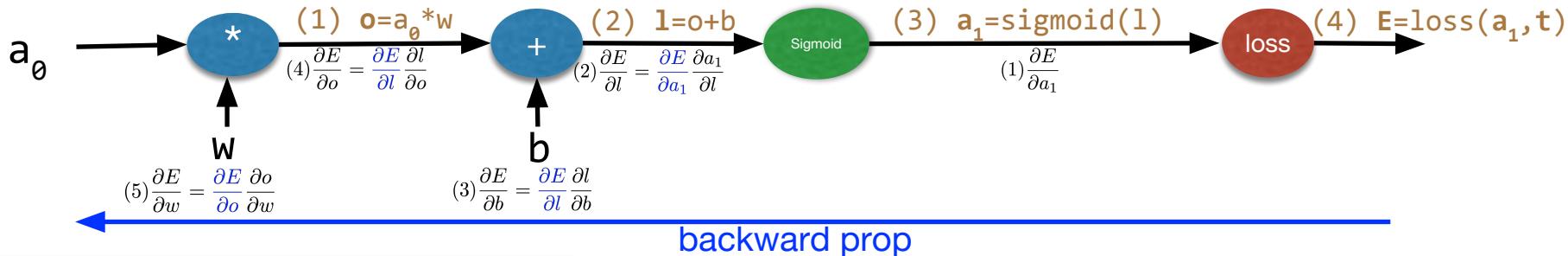
$$E = - \sum t \log(a) + (1-t) \log(1-a), \quad \frac{\partial E}{\partial a} = \frac{a-t}{a(1-a)}$$

$$a = \text{sigmoid}(l) = \frac{1}{1+e^{-l}}, \quad \frac{\partial a}{\partial l} = a(1-a)$$

Just apply them one by one and solve each derivative one by one!

Network

forward



Derivatives (chain rule)

$$(1) \frac{\partial E}{\partial a_1} = \frac{a_1 - t}{a_1(1 - a_1)}$$

$$(2) \frac{\partial E}{\partial l} = \frac{\partial E}{\partial a_1} \frac{\partial a_1}{\partial l} = \frac{a_1 - t}{a_1(1 - a_1)} * a_1(1 - a_1) = a_1 - t$$

$$(3) \frac{\partial E}{\partial b} = \frac{\partial E}{\partial l} \frac{\partial l}{\partial b} = \frac{\partial E}{\partial l} * 1 = a_1 - t$$

$$(4) \frac{\partial E}{\partial o} = \frac{\partial E}{\partial l} \frac{\partial l}{\partial o} = \frac{\partial E}{\partial l} * 1 = a_1 - t$$

$$(5) \frac{\partial E}{\partial w} = \frac{\partial E}{\partial o} \frac{\partial o}{\partial w} = \frac{\partial E}{\partial o} a_0 = (a_1 - t)a_0$$

Given from the pre computed derivative

Given

Gate derivatives

$$o = aw, \quad \frac{\partial o}{\partial w} = a, \quad \frac{\partial o}{\partial a} = w$$

$$l = o + b, \quad \frac{\partial l}{\partial o} = 1, \quad \frac{\partial l}{\partial b} = 1$$

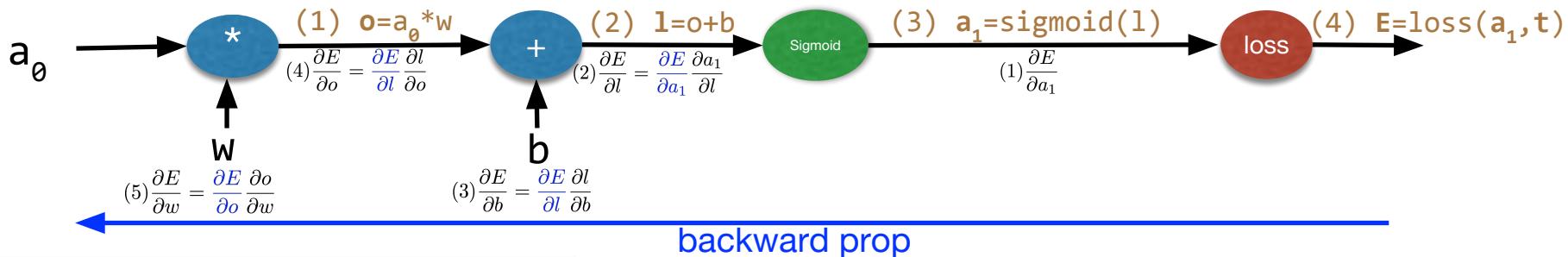
$$E = - \sum t \log(a) + (1-t) \log(1-a), \quad \frac{\partial E}{\partial a} = \frac{a-t}{a(1-a)}$$

$$a = \text{sigmoid}(l) = \frac{1}{1+e^{-l}}, \quad \frac{\partial a}{\partial l} = a(1-a)$$

Just apply them one by one and solve each derivative one by one!

Network

forward



Derivatives (chain rule)

$$(1) \frac{\partial E}{\partial a_1} = \frac{a_1 - t}{a_1(1 - a_1)}$$

$$(2) \frac{\partial E}{\partial l} = \frac{\partial E}{\partial a_1} \frac{\partial a_1}{\partial l} = \frac{a_1 - t}{a_1(1 - a_1)} * a_1(1 - a_1) = a_1 - t$$

$$(3) \frac{\partial E}{\partial b} = \frac{\partial E}{\partial l} \frac{\partial l}{\partial b} = \frac{\partial E}{\partial l} * 1 = a_1 - t$$

$$(4) \frac{\partial E}{\partial o} = \frac{\partial E}{\partial l} \frac{\partial l}{\partial o} = \frac{\partial E}{\partial l} * 1 = a_1 - t$$

$$(5) \frac{\partial E}{\partial w} = \frac{\partial E}{\partial o} \frac{\partial o}{\partial w} = a_0^T \frac{\partial E}{\partial o} = a_0^T (a_1 - t)$$

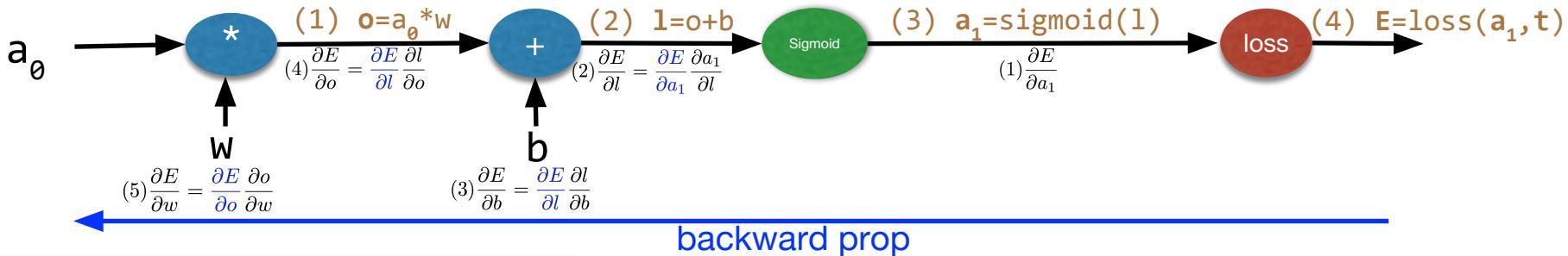
Matrix

$$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial O} \frac{\partial O}{\partial W} = A^T \frac{\partial E}{\partial O}, \text{ where } O = AW$$

For Matrix: <http://cs231n.github.io/optimization-2/#staged>

Network

forward



Derivatives (chain rule)

$$(1) \frac{\partial E}{\partial a_1} = \frac{a_1 - t}{a_1(1 - a_1)}$$

$$(2) \frac{\partial E}{\partial l} = \frac{\partial E}{\partial a_1} \frac{\partial a_1}{\partial l} = \frac{a_1 - t}{a_1(1 - a_1)} * a_1(1 - a_1) = a_1 - t$$

$$(3) \frac{\partial E}{\partial b} = \frac{\partial E}{\partial l} \frac{\partial l}{\partial b} = \frac{\partial E}{\partial l} * 1 = a_1 - t$$

$$(4) \frac{\partial E}{\partial o} = \frac{\partial E}{\partial l} \frac{\partial l}{\partial o} = \frac{\partial E}{\partial l} * 1 = a_1 - t$$

$$(5) \frac{\partial E}{\partial w} = \frac{\partial E}{\partial o} \frac{\partial o}{\partial w} = a_0^T \frac{\partial E}{\partial o} = a_0^T (a_1 - t)$$

Matrix

$$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial O} \frac{\partial O}{\partial W} = A^T \frac{\partial E}{\partial O}, \text{ where } O = AW$$

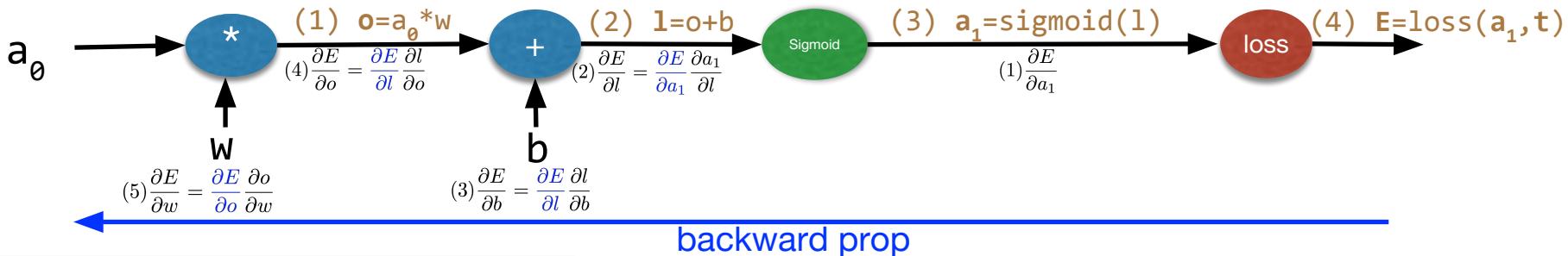
Network update (learning rate, alpha)

$$w = w - \alpha \frac{\partial E}{\partial w}$$

$$b = b - \alpha \frac{\partial E}{\partial b}$$

Network

forward



Derivatives (chain rule)

$$(1) \frac{\partial E}{\partial a_1} = \frac{a_1 - t}{a_1(1 - a_1)}$$

$$(2) \frac{\partial E}{\partial l} = \frac{\partial E}{\partial a_1} \frac{\partial a_1}{\partial l} = \frac{a_1 - t}{a_1(1 - a_1)} * a_1(1 - a_1) = a_1 - t$$

$$(3) \frac{\partial E}{\partial b} = \frac{\partial E}{\partial l} \frac{\partial l}{\partial b} = \frac{\partial E}{\partial l} * 1 = a_1 - t$$

$$(4) \frac{\partial E}{\partial o} = \frac{\partial E}{\partial l} \frac{\partial l}{\partial o} = \frac{\partial E}{\partial l} * 1 = a_1 - t$$

$$(5) \frac{\partial E}{\partial w} = \frac{\partial E}{\partial o} \frac{\partial o}{\partial w} = a_0^T \frac{\partial E}{\partial o} = a_0^T (a_1 - t)$$

Done! Let's update our network using derivatives!

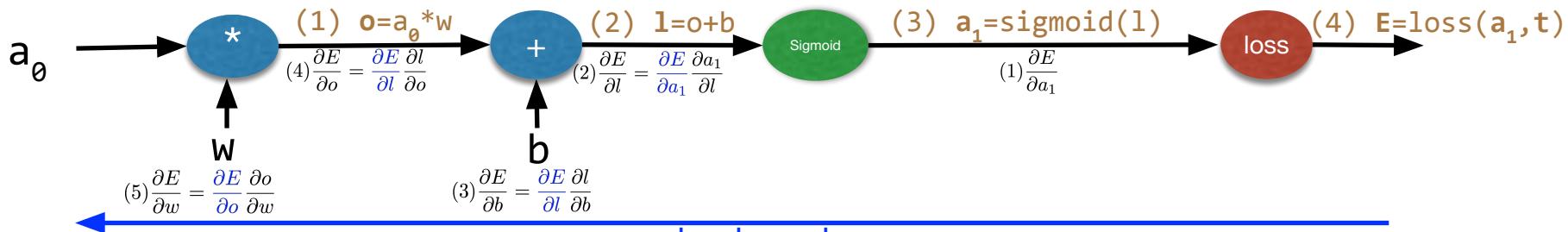
Network update (learning rate, alpha)

$$w = w - \alpha \frac{\partial E}{\partial w}$$

$$b = b - \alpha \frac{\partial E}{\partial b}$$

Network

forward



backward prop

$$E = - \sum t \log(a) + (1-t) \log(1-a), \quad \frac{\partial E}{\partial a} = \frac{a-t}{a(1-a)}$$

$$a = \text{sigmoid}(l) = \frac{1}{1 + e^{-l}}, \quad \frac{\partial a}{\partial l} = a(1-a)$$

```

d_a1 = (a1 - t) / (a1 * (1. - t) + 1e-7)
d_sigma = a1 * (1 - a1) # sigma prime
d_l = d_a1 * d_sigma # (a1 - t)
d_b = d_l * 1
d_o = d_l * 1
d_w = tf.matmul(tf.transpose(a0), d_o)

```

$$(1) \frac{\partial E}{\partial a_1} = \frac{a_1 - t}{a_1(1 - a_1)}$$

$$(2) \frac{\partial E}{\partial l} = \frac{\partial E}{\partial a_1} \frac{\partial a_1}{\partial l} = \frac{a_1 - t}{a_1(1 - a_1)} * a_1(1 - a_1) = a_1 - t$$

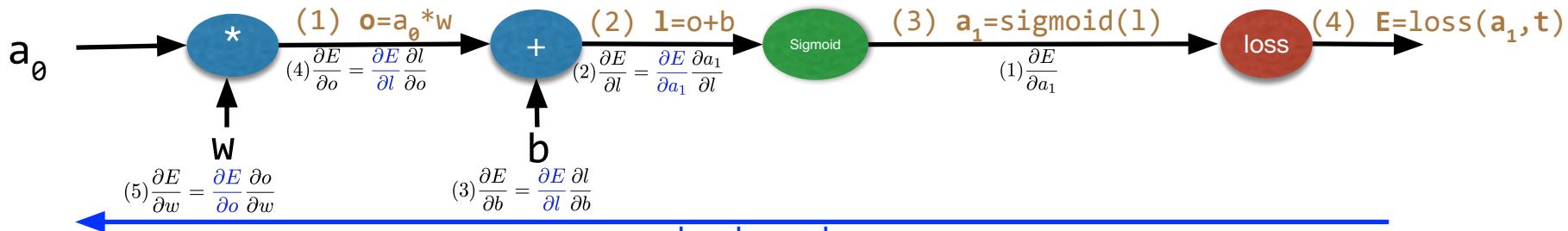
$$(3) \frac{\partial E}{\partial b} = \frac{\partial E}{\partial l} \frac{\partial l}{\partial b} = \frac{\partial E}{\partial l} * 1 = a_1 - t$$

$$(4) \frac{\partial E}{\partial o} = \frac{\partial E}{\partial l} \frac{\partial l}{\partial o} = \frac{\partial E}{\partial l} * 1 = a_1 - t$$

$$(5) \frac{\partial E}{\partial w} = \frac{\partial E}{\partial o} \frac{\partial o}{\partial w} = a_0^T \frac{\partial E}{\partial o} = a_0^T (a_1 - t)$$

Network

forward



Derivatives (chain rule)

$$(1) \frac{\partial E}{\partial a_1} = \frac{a_1 - t}{a_1(1 - a_1)}$$

$$(2) \frac{\partial E}{\partial l} = \frac{\partial E}{\partial a_1} \frac{\partial a_1}{\partial l} = \frac{a_1 - t}{a_1(1 - a_1)} * a_1(1 - a_1) = a_1 - t$$

$$(3) \frac{\partial E}{\partial b} = \frac{\partial E}{\partial l} \frac{\partial l}{\partial b} = \frac{\partial E}{\partial l} * 1 = a_1 - t$$

$$(4) \frac{\partial E}{\partial o} = \frac{\partial E}{\partial l} \frac{\partial l}{\partial o} = \frac{\partial E}{\partial l} * 1 = a_1 - t$$

$$(5) \frac{\partial E}{\partial w} = \frac{\partial E}{\partial o} \frac{\partial o}{\partial w} = a_0^T \frac{\partial E}{\partial o} = a_0^T (a_1 - t)$$

backward prop

$$E = - \sum t \log(a) + (1 - t) \log(1 - a), \quad \frac{\partial E}{\partial a} = \frac{a - t}{a(1 - a)}$$

$$a = \text{sigmoid}(l) = \frac{1}{1 + e^{-l}}, \quad \frac{\partial a}{\partial l} = a(1 - a)$$

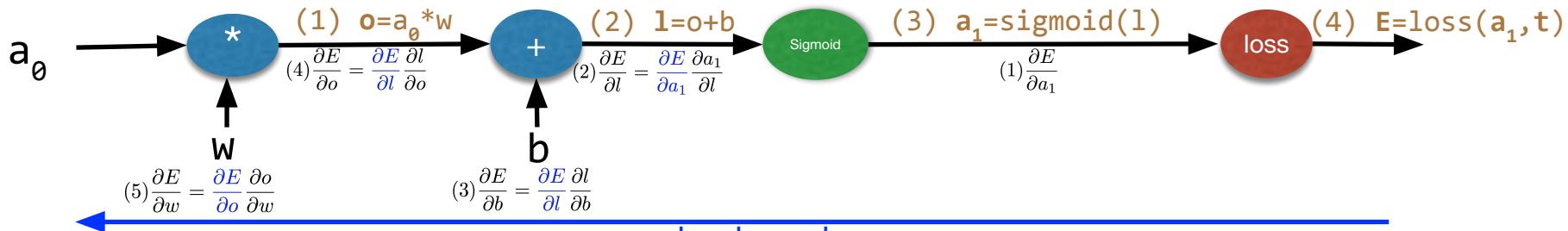
```

d_a1 = (a1 - t) / (a1 * (1. - a1) + 1e-7)
d_sigma = a1 * (1 - a1) # sigma prime
d_l = d_a1 * d_sigma # (a1 - t)
d_b = d_l * 1
d_o = d_l * 1
d_w = tf.matmul(tf.transpose(a0), d_o)
# Updating network using gradients
learning_rate = 0.01
train_step = [
    tf.assign(w, w - learning_rate * d_w),
    tf.assign(b, b - learning_rate * d_b)]

```

Network

forward



Derivatives (chain rule)

$$(1) \frac{\partial E}{\partial a_1} = \frac{a_1 - t}{a_1(1 - a_1)}$$

$$(2) \frac{\partial E}{\partial l} = \frac{\partial E}{\partial a_1} \frac{\partial a_1}{\partial l} = \frac{a_1 - t}{a_1(1 - a_1)} * a_1(1 - a_1) = a_1 - t$$

$$(3) \frac{\partial E}{\partial b} = \frac{\partial E}{\partial l} \frac{\partial l}{\partial b} = \frac{\partial E}{\partial l} * 1 = a_1 - t$$

$$(4) \frac{\partial E}{\partial o} = \frac{\partial E}{\partial l} \frac{\partial l}{\partial o} = \frac{\partial E}{\partial l} * 1 = a_1 - t$$

$$(5) \frac{\partial E}{\partial w} = \frac{\partial E}{\partial o} \frac{\partial o}{\partial w} = a_0^T \frac{\partial E}{\partial o} = a_0^T (a_1 - t)$$

backward prop

$$E = - \sum t \log(a) + (1 - t) \log(1 - a), \quad \frac{\partial E}{\partial a} = \frac{a - t}{a(1 - a)}$$

$$a = \text{sigmoid}(l) = \frac{1}{1 + e^{-l}}, \quad \frac{\partial a}{\partial l} = a(1 - a)$$

```

d_a1 = (a1 - t) / (a1 * (1. - a1) + 1e-7)
d_sigma = a1 * (1 - a1) # sigma prime
d_l = d_a1 * d_sigma # (a1 - t)
d_b = d_l * 1
d_o = d_l * 1
d_W = tf.matmul(tf.transpose(a0), d_o)
# Updating network using gradients
learning_rate = 0.01
train_step = [
    tf.assign(W, W - learning_rate * d_W / N), # sample size
    tf.assign(b, b - learning_rate * tf.reduce_mean(d_b))]

```

Exercise

- See more backprop code samples at
<https://github.com/hunkim/DeepLearningZeroToAll>
- [`https://github.com/hunkim/DeepLearningZeroToAll/blob/master/lab-09-7-sigmoid_back_prop.py`](https://github.com/hunkim/DeepLearningZeroToAll/blob/master/lab-09-7-sigmoid_back_prop.py)
- Solve XOR using NN backprop