

Lab I

TensorFlow Basics

Sung Kim <hunkim+ml@gmail.com>

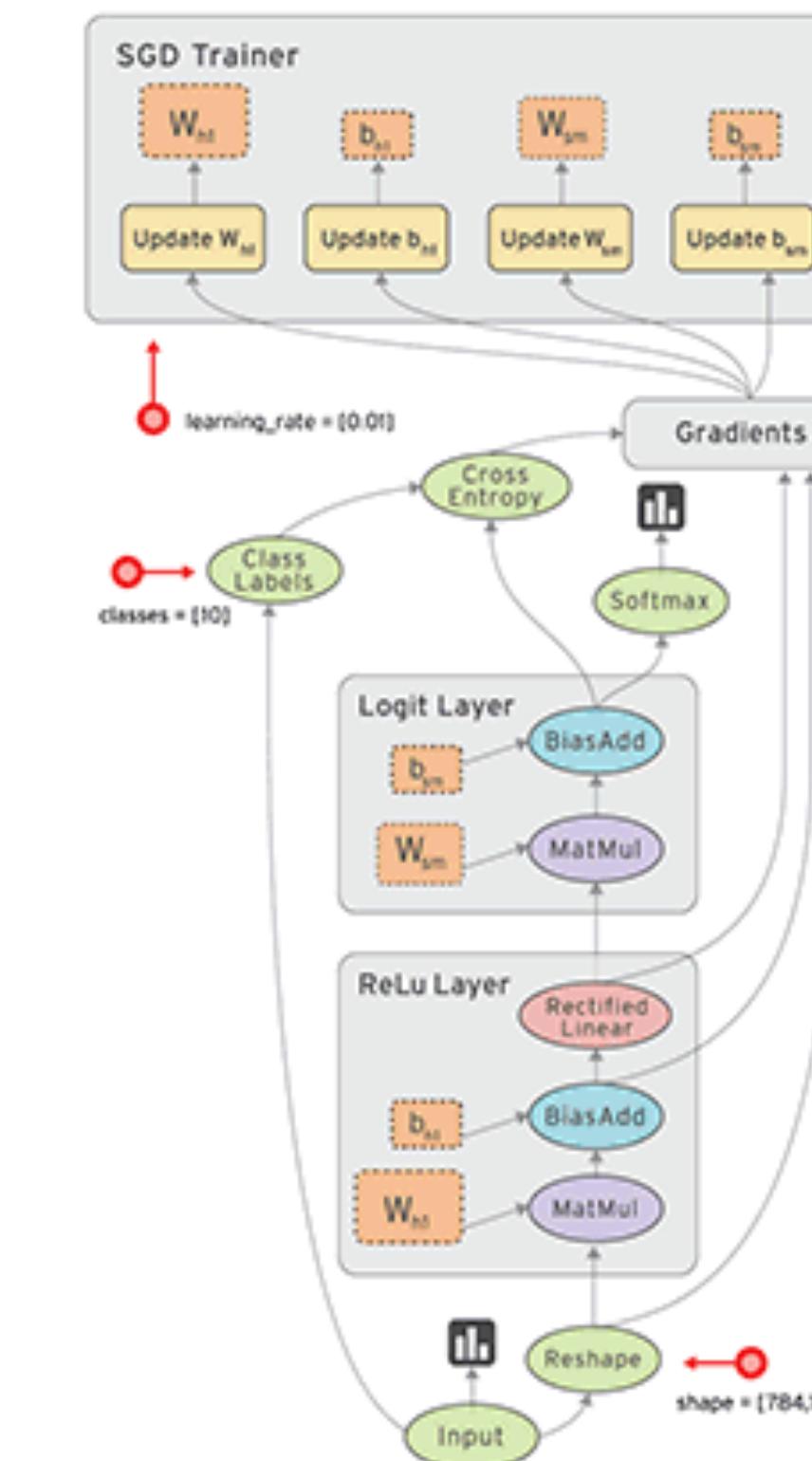
TensorFlow

- TensorFlow™ is an open source software library for numerical computation using data flow graphs.
- Python!



What is a Data Flow Graph?

- Nodes in the graph represent mathematical operations
- Edges represent the multidimensional data arrays (tensors) communicated between them.



Installation

- install pip

```
# Ubuntu/Linux 64-bit  
$ sudo apt-get install python-pip python-dev  
  
# Mac OS X  
$ sudo easy_install pip
```

- install TensorFlow

```
# Ubuntu/Linux 64-bit, CPU only:  
$ sudo pip install --upgrade https://storage.googleapis.com/tensorflow/linux/cpu/tensorflow  
  
# Ubuntu/Linux 64-bit, GPU enabled:  
$ sudo pip install --upgrade https://storage.googleapis.com/tensorflow/linux/gpu/tensorflow  
  
# Mac OS X, CPU only:  
$ sudo easy_install --upgrade six  
$ sudo pip install --upgrade https://storage.googleapis.com/tensorflow/mac/tensorflow
```

Hello World!

```
import tensorflow as tf

#Simple hello world using TensorFlow

# Create a Constant op
# The op is added as a node to the default graph.
#
# The value returned by the constructor represents the output
# of the Constant op.
hello = tf.constant('Hello, TensorFlow!')

# Start tf session
sess = tf.Session()

print sess.run(hello)
```

<https://github.com/aymericdamien/TensorFlow-Examples/>

Everything is operation!

```
import tensorflow as tf

# Start tf session
sess = tf.Session()

# Basic constant operations
# The value returned by the constructor represents the output
# of the Constant op.
a = tf.constant(2)
b = tf.constant(3)

c = a+b

# Print out operation
print c

# Print out the result of operation
print sess.run(c)
```

Everything is operation!

```
import tensorflow as tf

# Start tf session
sess = tf.Session()

# Basic constant operations
# The value returned by the constructor represents the output
# of the Constant op.
a = tf.constant(2)
b = tf.constant(3)

c = a+b

# Print out operation
print c

# Print out the result of operation
print sess.run(c)
```

```
Run 02-helloOp
▶ /Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /Users/hunkim/gitWorkspace/tFlow/01-Basics/02-helloOp.py
Tensor("add:0", shape=(), dtype=int32)
5
```

Basic operations

```
import tensorflow as tf

# Basic constant operations
# The value returned by the constructor represents the output
# of the Constant op.
a = tf.constant(2)
b = tf.constant(3)

# Launch the default graph.
with tf.Session() as sess:
    print "a=2, b=3"
    print "Addition with constants: %i" % sess.run(a+b)
    print "Multiplication with constants: %i" % sess.run(a*b)
```

output:

```
a=2, b=3
Addition with constants: 5
Multiplication with constants: 6
..... . . . . -
```

Placeholder

```
# Basic Operations with variable as graph input
# The value returned by the constructor represents the output
# of the Variable op. (define as input when running session)
# tf Graph input
a = tf.placeholder(tf.int16)
b = tf.placeholder(tf.int16)

# Define some operations
add = tf.add(a, b)
mul = tf.mul(a, b)

# Launch the default graph.
with tf.Session() as sess:
    # Run every operation with variable input
    print "Addition with variables: %i" % sess.run(add, feed_dict={a: 2, b: 3})
    print "Multiplication with variables: %i" % sess.run(mul, feed_dict={a: 2, b: 3})
```

output:

```
Addition with variables: 5
Multiplication with variables: 6
```

Lab 2

Linear Regression

Sung Kim <hunkim+ml@gmail.com>

Hypothesis and cost function

$$H(x) = Wx + b$$

$$cost(W, b) = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y^{(i)})^2$$

```

import tensorflow as tf

x_data = [1, 2, 3]
y_data = [1, 2, 3]

# Try to find values for W and b that compute y_data = W * x_data + b
# (We know that W should be 1 and b 0, but Tensorflow will
# figure that out for us.)
W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.random_uniform([1], -1.0, 1.0))

# Our hypothesis
hypothesis = W * x_data + b          
$$H(x) = Wx + b$$


# Simplified cost function
cost = tf.reduce_mean(tf.square(hypothesis - y_data))      
$$cost(W, b) = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y^{(i)})^2$$


# Minimize
a = tf.Variable(0.1) # Learning rate, alpha
optimizer = tf.train.GradientDescentOptimizer(a)
train = optimizer.minimize(cost)

# Before starting, initialize the variables. We will 'run' this first.
init = tf.initialize_all_variables()

# Launch the graph.
sess = tf.Session()
sess.run(init)

# Fit the line.
for step in xrange(2001):
    sess.run(train)
    if step % 20 == 0:
        print step, sess.run(cost), sess.run(W), sess.run(b)

# Learns best fit is W: [0.1], b: [0.3]

```

Placeholder

```
# Basic Operations with variable as graph input
# The value returned by the constructor represents the output
# of the Variable op. (define as input when running session)
# tf Graph input
a = tf.placeholder(tf.int16)
b = tf.placeholder(tf.int16)

# Define some operations
add = tf.add(a, b)
mul = tf.mul(a, b)

# Launch the default graph.
with tf.Session() as sess:
    # Run every operation with variable input
    print "Addition with variables: %i" % sess.run(add, feed_dict={a: 2, b: 3})
    print "Multiplication with variables: %i" % sess.run(mul, feed_dict={a: 2, b: 3})
```

output:

```
Addition with variables: 5
Multiplication with variables: 6
```

```
x_data = [1., 2., 3.]
y_data = [1., 2., 3.]

# Try to find values for W and b that compute y_data = W * x_data + b
# (We know that W should be 1 and b 0, but Tensorflow will
# figure that out for us.)
W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.random_uniform([1], -1.0, 1.0))

X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)

# Our hypothesis
hypothesis = W * X + b

# Simplified cost function
cost = tf.reduce_mean(tf.square(hypothesis - Y))

# Minimize
a = tf.Variable(0.1)
optimizer = tf.train.GradientDescentOptimizer(a)
train = optimizer.minimize(cost)

# Before starting, initialize the variables. We will 'run' this first.
init = tf.initialize_all_variables()

# Launch the graph.
sess = tf.Session()
sess.run(init)

# Fit the line.
for step in xrange(2001):
    sess.run(train, feed_dict={X:x_data, Y:y_data})
    if step % 20 == 0:
        print step, sess.run(cost, feed_dict={X:x_data, Y:y_data}), sess.run(W), sess.run(b)
```

Placeholder

```
# Our hypothesis
hypothesis = W * X + b
...
# Launch the graph.
sess = tf.Session()
sess.run(init)

# Fit the line.
for step in xrange(2001):
    sess.run(train, feed_dict={X:x_data, Y:y_data})
    if step % 20 == 0:
        print step, sess.run(cost, feed_dict={X:x_data, Y:y_data}), sess.run(W), sess.run(b)

# Learns best fit is W: [1], b: [0]
print sess.run(hypothesis, feed_dict={X:5})
print sess.run(hypothesis, feed_dict={X:2.5})
```

Lab 3

Minimizing Cost

Sung Kim <hunkim+ml@gmail.com>

Simplified hypothesis

$$H(x) = Wx$$

$$cost(W) = \frac{1}{m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)})^2$$

```

# tf Graph Input
X = [1., 2., 3.]
Y = [1., 2., 3.]
m = n_samples = len(X)

# Set model weights
W = tf.placeholder(tf.float32)

# Construct a linear model
hypothesis = tf.mul(X, W)

# Cost function
cost = tf.reduce_sum(tf.pow(hypothesis-Y, 2))/(m)

# Initializing the variables
init = tf.initialize_all_variables()

# For graphs
W_val = []
cost_val = []

# Launch the graph
sess = tf.Session()
sess.run(init)
for i in range(-30, 50):
    print i*0.1, sess.run(cost, feed_dict={W: i*0.1})
    W_val.append(i*0.1)
    cost_val.append(sess.run(cost, feed_dict={W: i*0.1}))

# Graphic display
plt.plot(W_val, cost_val, 'ro')
plt.ylabel('Cost')
plt.xlabel('W')
plt.show()

```

$$H(x) = Wx$$

$$cost(W) = \frac{1}{m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)})^2$$

```

# tf Graph Input
X = [1., 2., 3.]
Y = [1., 2., 3.]
m = n_samples = len(X)

# Set model weights
W = tf.placeholder(tf.float32)

# Construct a linear model
hypothesis = tf.mul(X, W)

# Cost function
cost = tf.reduce_sum(tf.pow(hypothesis-Y, 2))/(m)

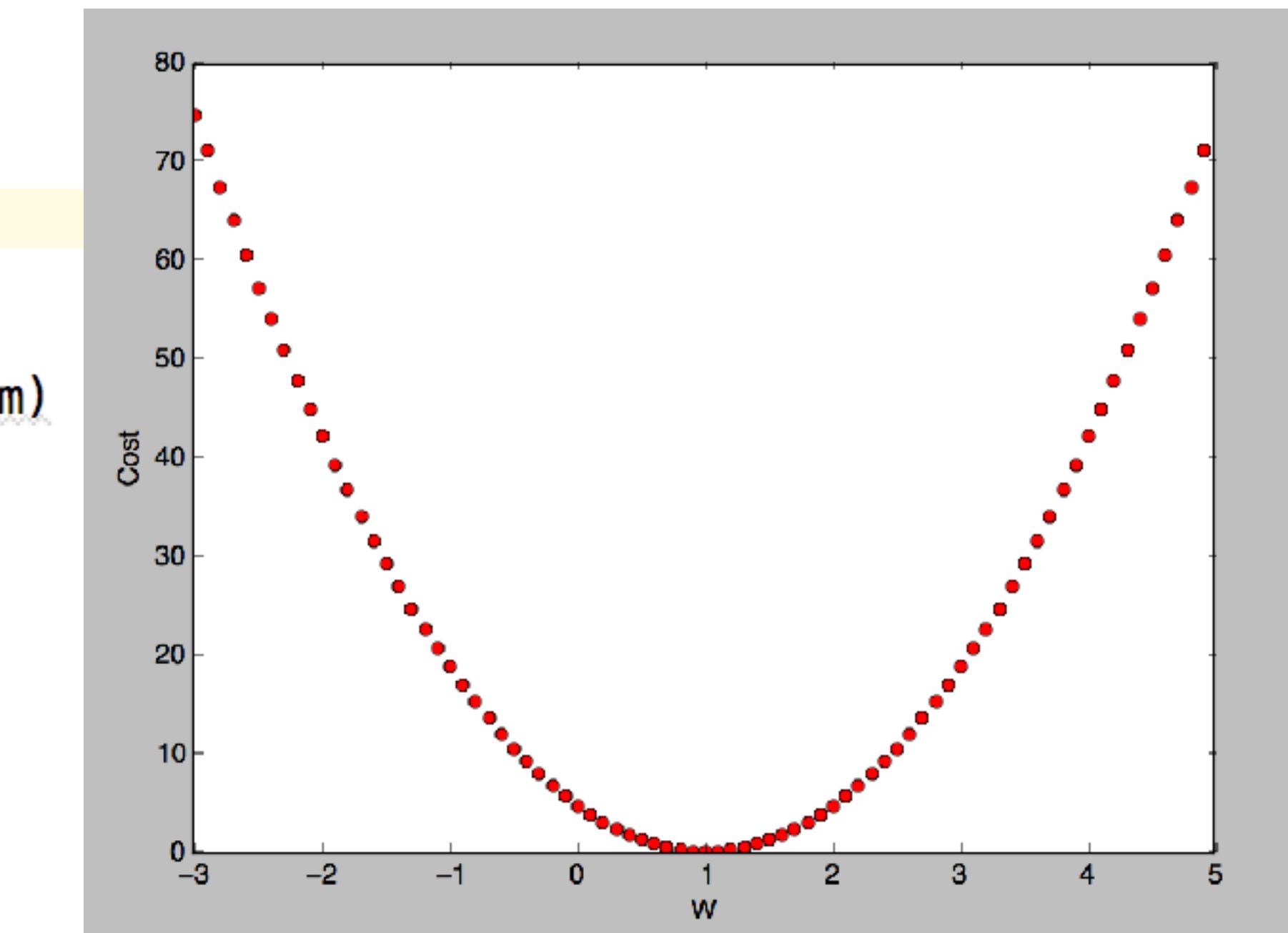
# Initializing the variables
init = tf.initialize_all_variables()

# For graphs
W_val = []
cost_val = []

# Launch the graph
sess = tf.Session()
sess.run(init)
for i in range(-30, 50):
    print i*0.1, sess.run(cost, feed_dict={W: i*0.1})
    W_val.append(i*0.1)
    cost_val.append(sess.run(cost, feed_dict={W: i*0.1}))

# Graphic display
plt.plot(W_val, cost_val, 'ro')
plt.ylabel('Cost')
plt.xlabel('W')
plt.show()

```



```

x_data = [1., 2., 3.]
y_data = [1., 2., 3.]

# Try to find values for W and b that compute y_data = W * x_data + b
# (We know that W should be 1 and b 0, but Tensorflow will
# figure that out for us.)
W = tf.Variable(tf.random_uniform([1], -10.0, 10.0))

X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)

# Our hypothesis
hypothesis = W * X

# Simplified cost function
cost = tf.reduce_mean(tf.square(hypothesis - Y))

# Minimize
descent = W - tf.mul(0.1, tf.reduce_mean(tf.mul((tf.mul(W, X) - Y), X)))
update = W.assign(descent)

# Before starting, initialize the variables. We will 'run' this first.
init = tf.initialize_all_variables()

# Launch the graph.
sess = tf.Session()
sess.run(init)

# Fit the line.
for step in xrange(20):
    sess.run(update, feed_dict={X:x_data, Y:y_data})
    print step, sess.run(cost, feed_dict={X:x_data, Y:y_data}), sess.run(W)

```

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

```
# tf Graph Input
X = [1., 2., 3.]
Y = [1., 2., 3.]
m = n_samples = len(X)

# Set model weights
W = tf.Variable(0.)
# Construct a linear model
hypothesis = tf.mul(X, W)

# Cost function
cost = tf.reduce_sum(tf.pow(hypothesis-Y, 2))/(m)
optimizer = tf.train.GradientDescentOptimizer(0.1)
train = optimizer.minimize(cost)

# Initializing the variables
init = tf.initialize_all_variables()

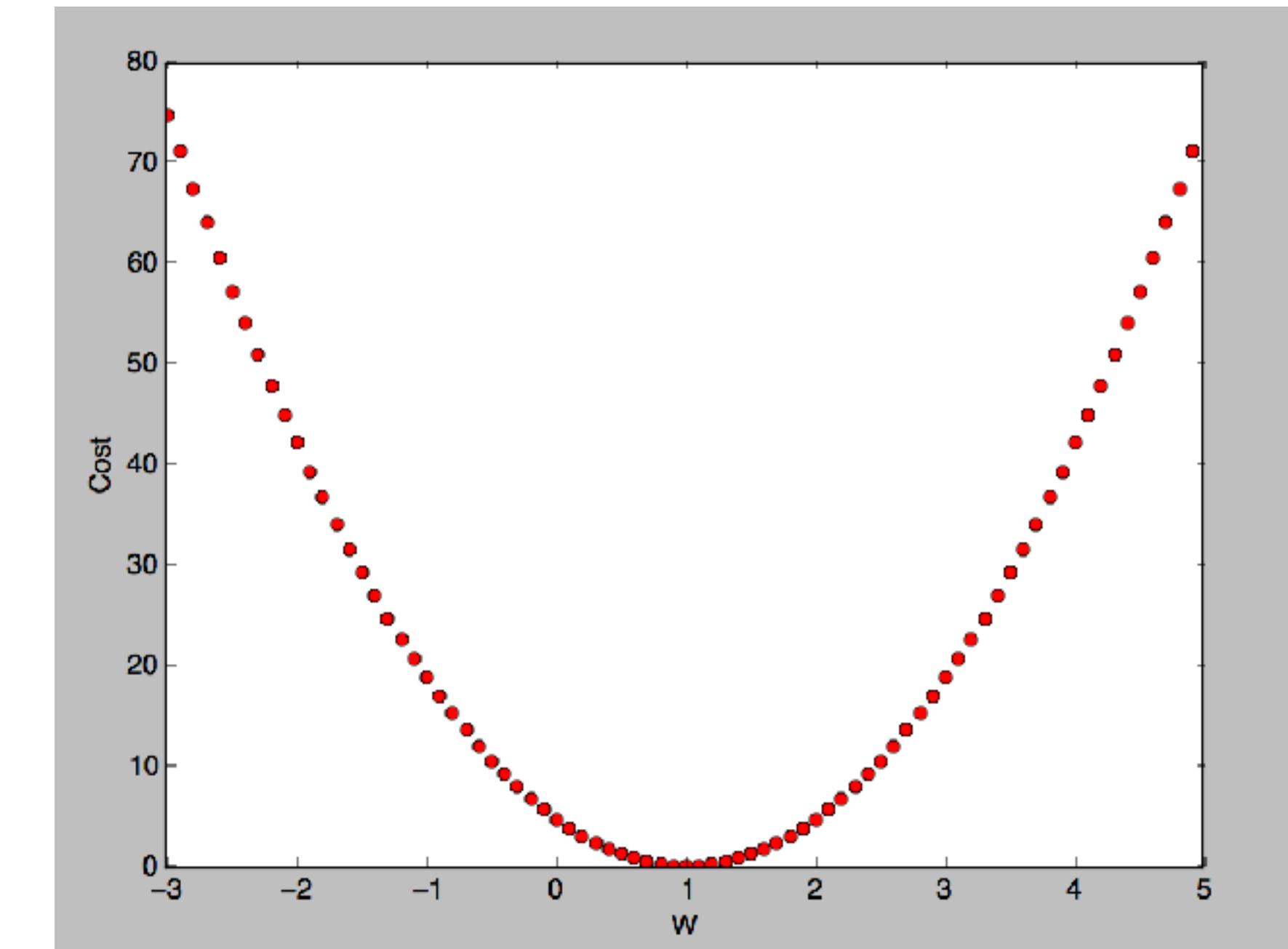
# Launch the graph
sess = tf.Session()
sess.run(init)

# Set model weights
w0p = W.assign(5.)
sess.run(w0p)
for step in xrange(10):
    print step, sess.run(W)
    sess.run(train)
```

Output when $W=5$

Run 02-costMin

	/Library/Frameworks
0	5.0
1	1.26667
2	1.01778
3	1.00119
4	1.00008
5	1.00001
6	1.0
7	1.0
8	1.0
9	1.0

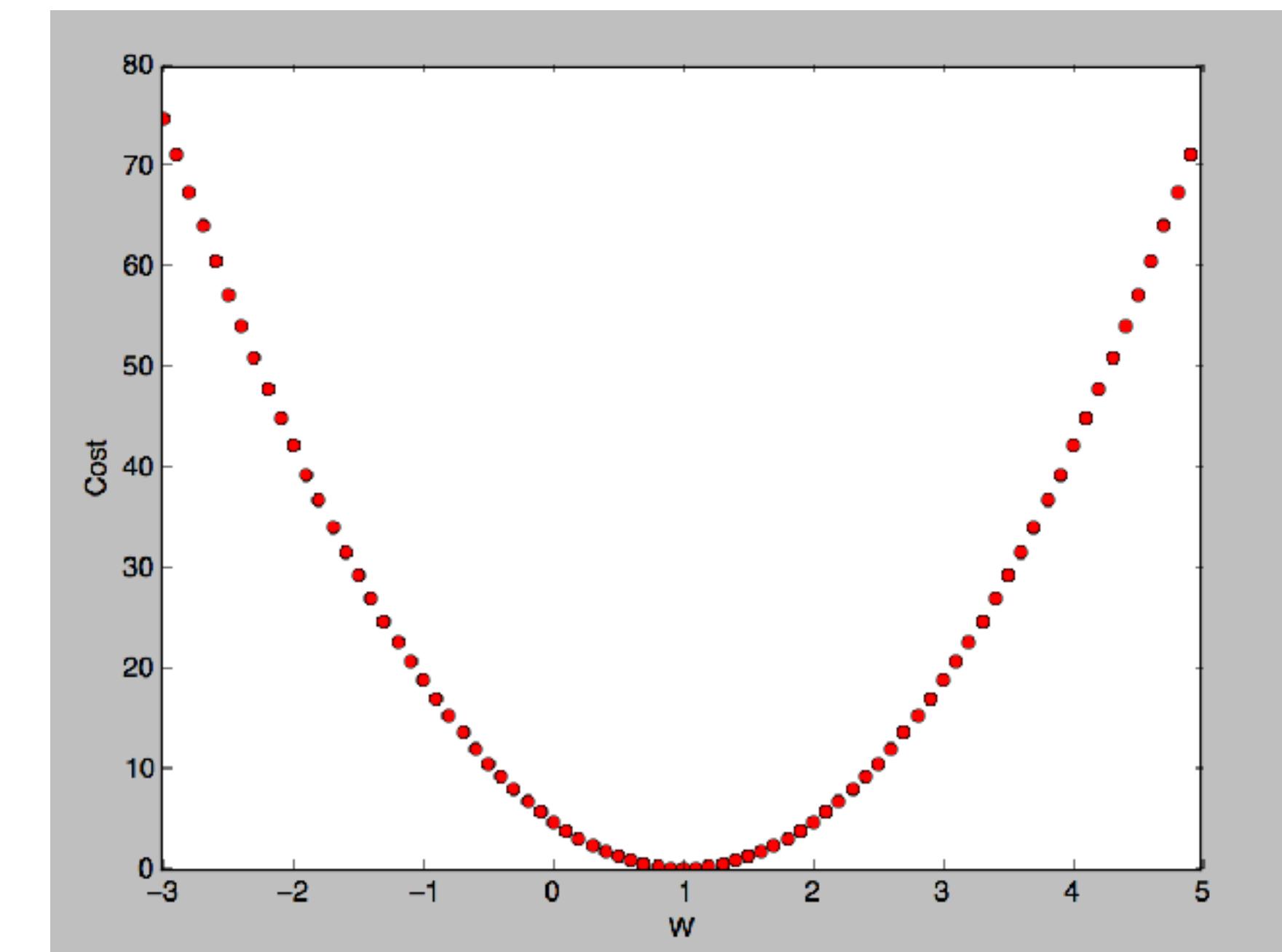


Output when $W=-3$

```
w0p = w.assign(-3.)  
sess.run(init)  
sess.run(w0p)  
for step in xrange(10):  
    print step, sess.run(w)  
    sess.run(train)
```

Run 02-costMin

Step	Cost
0	-3.0
1	0.733334
2	0.982222
3	0.998815
4	0.999921
5	0.999995
6	1.0
7	1.0
8	1.0
9	1.0



Lab 4

Multi-variable linear regression

Sung Kim <hunkim+ml@gmail.com>

Multivariable example

$$H(x_1, x_2) = w_1x_1 + w_2x_2 + b$$

x1	x2	y
1	0	1
0	2	2
3	0	3
0	4	4
5	0	5

Multivariable example

$$H(x_1, x_2) = w_1 x_1 + w_2 x_2 + b$$

x1	x2	y
1	0	1
0	2	2
3	0	3
0	4	4
5	0	5

```
W1 = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
W2 = tf.Variable(tf.random_uniform([1], -1.0, 1.0))

b = tf.Variable(tf.random_uniform([1], -1.0, 1.0))

# Our hypothesis
hypothesis = W1 * x1_data + W2 * x2_data + b
```

```
x1_data = [1, 0, 3, 0, 5]
x2_data = [0, 2, 0, 4, 0]
y_data = [1, 2, 3, 4, 5]
```

```
x1_data = [1, 0, 3, 0, 5]
x2_data = [0, 2, 0, 4, 0]
y_data = [1, 2, 3, 4, 5]

# Try to find values for W and b that compute y_data = W * x_data + b
# (We know that W should be 1 and b 0, but Tensorflow will
# figure that out for us.)
W1 = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
W2 = tf.Variable(tf.random_uniform([1], -1.0, 1.0))

b = tf.Variable(tf.random_uniform([1], -1.0, 1.0))

# Our hypothesis
hypothesis = W1 * x1_data + W2 * x2_data + b

# Simplified cost function
cost = tf.reduce_mean(tf.square(hypothesis - y_data))

# Minimize
a = tf.Variable(0.1) # Learning rate, alpha
optimizer = tf.train.GradientDescentOptimizer(a)
train = optimizer.minimize(cost)

# Before starting, initialize the variables. We will 'run' this first.
init = tf.initialize_all_variables()

# Launch the graph.
sess = tf.Session()
sess.run(init)

# Fit the line.
for step in xrange(2001):
    sess.run(train)
    if step % 20 == 0:
        print step, sess.run(cost), sess.run(W1), sess.run(W2), sess.run(b)
```

Matrix

$$w_1x_1 + w_2x_2 + w_3x_3 + \dots + w_nx_n$$

$$\begin{bmatrix} w1 & w2 & w3 \end{bmatrix} \times \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} = [w1 \times x1 + w2 \times x2 + w3 \times x3]$$

Matrix

$$w_1x_1 + w_2x_2 + w_3x_3 + \dots + w_nx_n$$

$$\begin{bmatrix} w1 & w2 & w3 \end{bmatrix} \times \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} = [w1 \times x1 + w2 \times x2 + w3 \times x3] :$$

```
x_data = [[0., 2., 0., 4., 0.],  
          [1., 0., 3., 0., 5.]]  
y_data = [1, 2, 3, 4, 5]  
  
# Try to find values for W  
W = tf.Variable(tf.random_uniform([1,2], -1.0, 1.0))  
b = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
```

```
import tensorflow as tf
x_data = [[0., 2., 0., 4., 0.],
           [1., 0., 3., 0., 5.]]
y_data = [1, 2, 3, 4, 5]

# Try to find values for W
W = tf.Variable(tf.random_uniform([1,2], -1.0, 1.0))
b = tf.Variable(tf.random_uniform([1], -1.0, 1.0))

#💡 Our hypothesis
hypothesis = tf.matmul(W, x_data) + b
# Simplified cost function
cost = tf.reduce_mean(tf.square(hypothesis - y_data))

# Minimize
a = tf.Variable(0.1) # Learning rate, alpha
optimizer = tf.train.GradientDescentOptimizer(a)
train = optimizer.minimize(cost)

# Before starting, initialize the variables. We will 'run' this first.
init = tf.initialize_all_variables()

# Launch the graph.
sess = tf.Session()
sess.run(init)

# Fit the line.
for step in xrange(2001):
    sess.run(train)
    if step % 20 == 0:
        print step, sess.run(cost), sess.run(W), sess.run(b)
```

Hypothesis

$$\begin{bmatrix} b & w_1 & w_2 & w_3 \end{bmatrix} \times \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = [b \times 1 + w_1 \times x_1 + w_2 \times x_2 + w_3 \times x_3]$$

$$H(X) = WX$$

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

# Try to find values for W
W = tf.Variable(tf.random_uniform([1,3], -1.0, 1.0))

# Our hypothesis
# previous hypothesis with b, hypothesis = tf.matmul(W, x_data) + b
hypothesis = tf.matmul(W, x_data)
# Simplified cost function
cost = tf.reduce_mean(tf.square(hypothesis - y_data))

# Minimize
a = tf.Variable(0.1) # Learning rate, alpha
optimizer = tf.train.GradientDescentOptimizer(a)
train = optimizer.minimize(cost)

# Before starting, initialize the variables. We will 'run' this first.
init = tf.initialize_all_variables()

# Launch the graph.
sess = tf.Session()
sess.run(init)

# Fit the line.
for step in xrange(2001):
    sess.run(train)
    if step % 20 == 0:
        print step, sess.run(cost), sess.run(W)

# Learns best fit is W: [1 1 0]
```

Loading data from file

train.txt

#	x0	x1	x2	y
1	1	0	0	1
1	0	2	0	2
1	3	0	0	3
1	0	4	0	4
1	5	0	0	5

```
import tensorflow as tf
import numpy as np
# http://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.loadtxt.html
xy = np.loadtxt('train.txt', unpack=True, dtype='float32')
x_data = xy[0:-1]
y_data = xy[-1];
```

```
import tensorflow as tf
import numpy as np
# http://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.loadtxt.html
xy = np.loadtxt('train.txt', unpack=True, dtype='float32')
x_data = xy[0:-1]
y_data = xy[-1];

print 'x', x_data
print 'y', y_data

W = tf.Variable(tf.random_uniform([1, len(x_data)], -5.0, 5.0))

# Our hypothesis
hypothesis = tf.matmul(W, x_data)
# Simplified cost function
cost = tf.reduce_mean(tf.square(hypothesis - y_data))

# Minimize
a = tf.Variable(0.1) # Learning rate, alpha
optimizer = tf.train.GradientDescentOptimizer(a)
train = optimizer.minimize(cost)

# Before starting, initialize the variables. We will 'run' this first.
init = tf.initialize_all_variables()

# Launch the graph.
sess = tf.Session()
sess.run(init)

# Fit the line.
for step in xrange(2001):
    sess.run(train)
    if step % 20 == 0:
        print step, sess.run(cost), sess.run(W)

# Learns best fit is W: [0.1], b: [0.3]
```

Lab 5

Logistic (regression) classifier

Sung Kim <hunkim+ml@gmail.com>

Logistic Regression

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

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

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

```

xy = np.loadtxt('train.txt', unpack=True, dtype='float32')
x_data = xy[0:-1]
y_data = xy[-1];

X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)

W = tf.Variable(tf.random_uniform([1, len(x_data)], -1.0, 1.0))

# Our hypothesis
h = tf.matmul(W, X)
hypothesis = tf.div(1., 1.+tf.exp(-h))
# cost function
cost = -tf.reduce_mean(Y*tf.log(hypothesis) + (1-Y)*tf.log(1-hypothesis))

# Minimize
a = tf.Variable(0.1) # Learning rate, alpha
optimizer = tf.train.GradientDescentOptimizer(a)
train = optimizer.minimize(cost)

# Before starting, initialize the variables. We will 'run' this first.
init = tf.initialize_all_variables()

# Launch the graph.
sess = tf.Session()
sess.run(init)

# Fit the line.
for step in xrange(2001):
    sess.run(train, feed_dict={X:x_data, Y:y_data})
    if step % 20 == 0:
        print step, sess.run(cost, feed_dict={X:x_data, Y:y_data}), sess.run(W)

```

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

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

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

Training Data

#	x0	x1	x2	y
1	2	1		0
1	3	2		0
1	3	4		0
1	5	5		1
1	7	5		1
1	2	5		1

```
# http://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.loadtxt.html
xy = np.loadtxt('train.txt', unpack=True, dtype='float32')
x_data = xy[0:-1]
y_data = xy[-1];
```

Ask to ML

Ask to ML

```
print '-----'  
# study_hour attendance  
print sess.run(hypothesis, feed_dict={X:[[1], [2], [2]]})>0.5  
print sess.run(hypothesis, feed_dict={X:[[1], [5], [5]]})>0.5  
  
print sess.run(hypothesis, feed_dict={X:[[1, 1], [4, 3], [3, 5]]})>0.5
```

output:

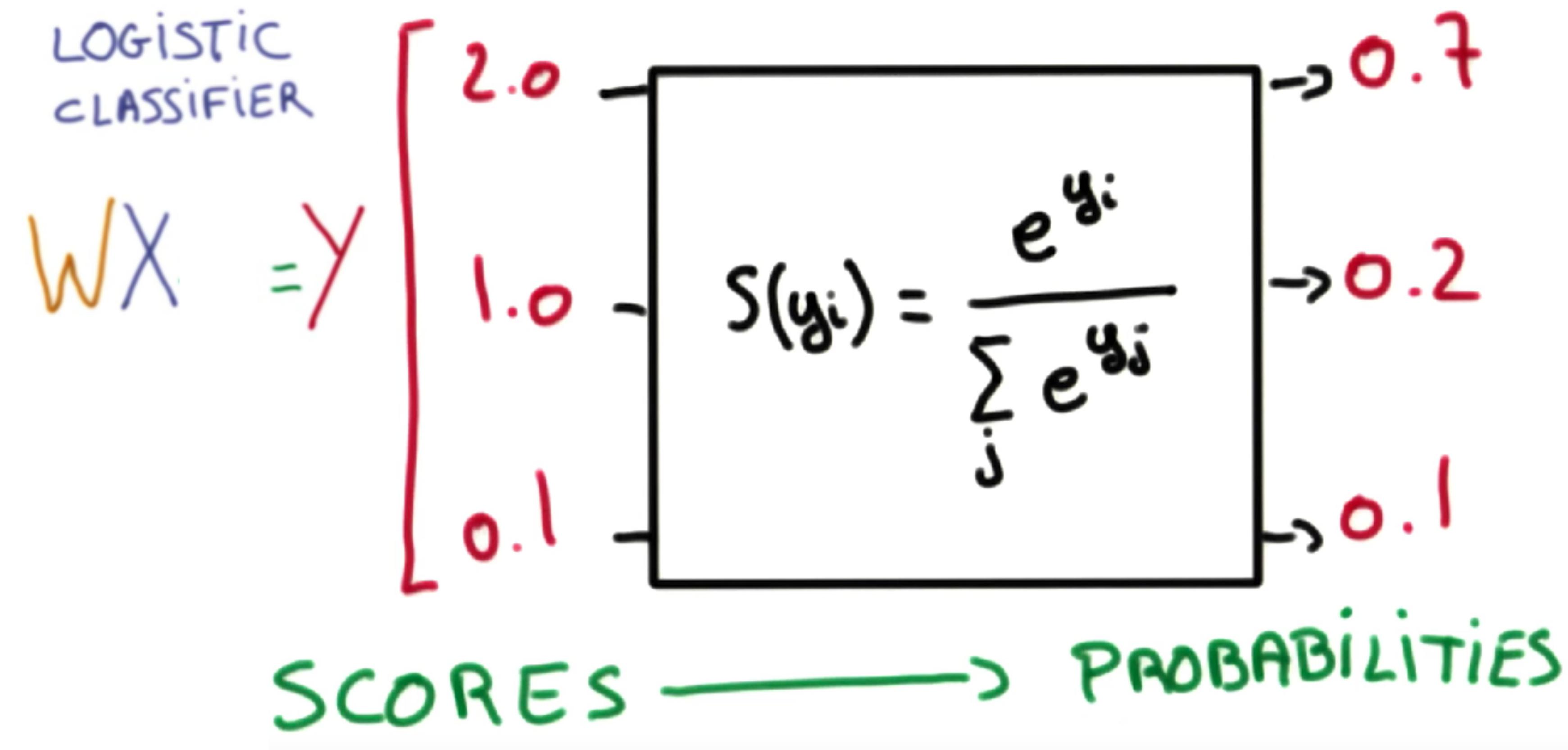
```
-----  
[[False]]  
[[ True]]  
[[False True]]
```

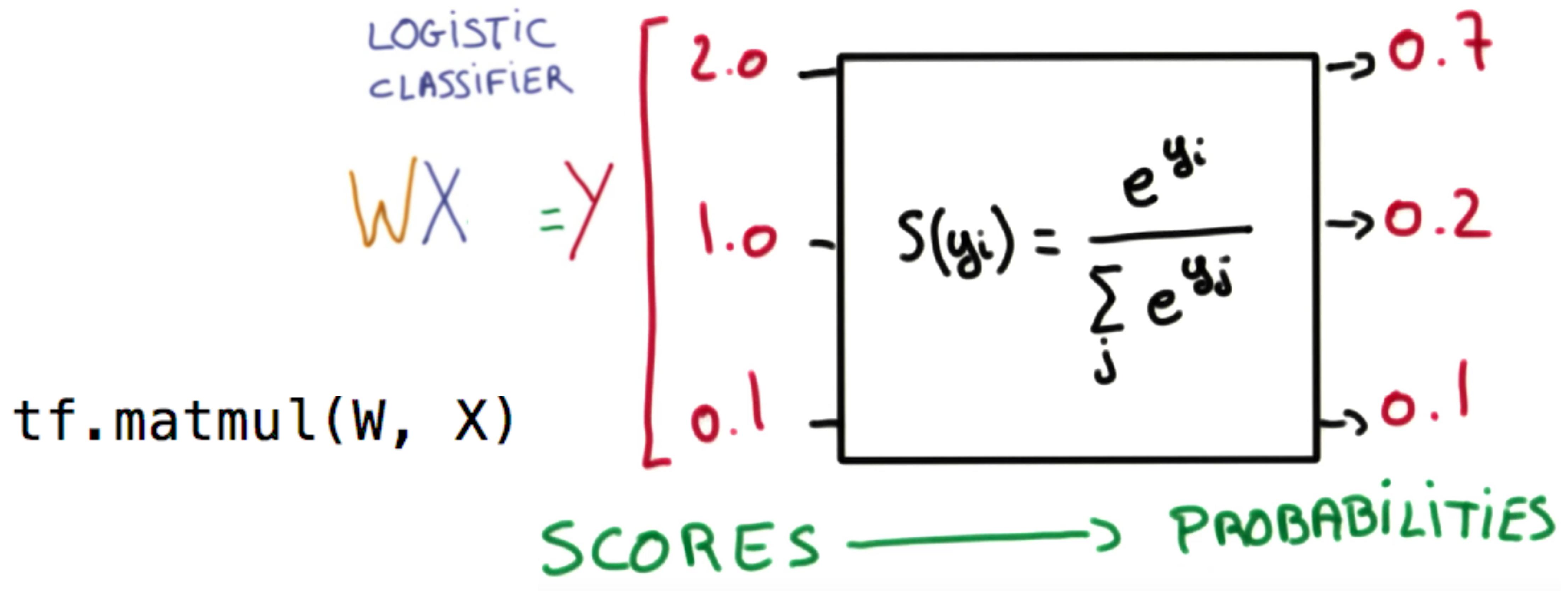
Lab 6

Softmax classifier

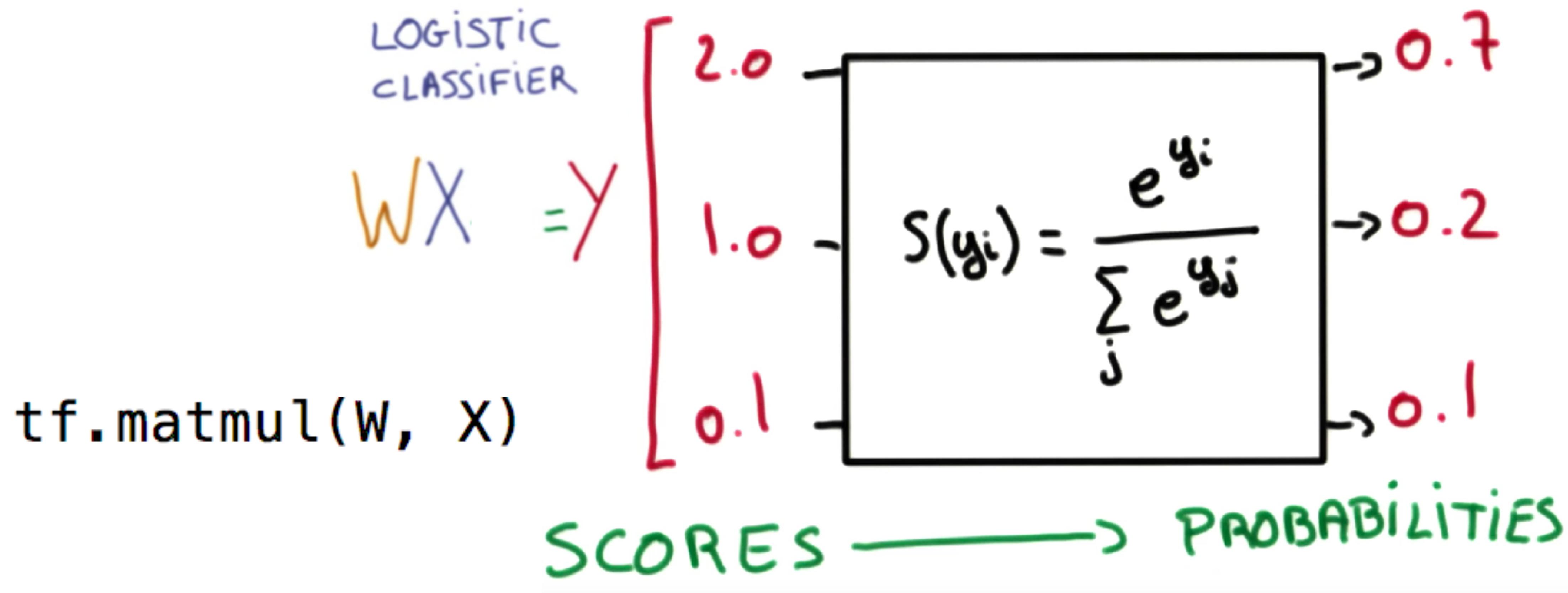
Sung Kim <hunkim+ml@gmail.com>

Softmax function





```
hypothesis = tf.nn.softmax(tf.matmul(w, x))
```



WX vs XW

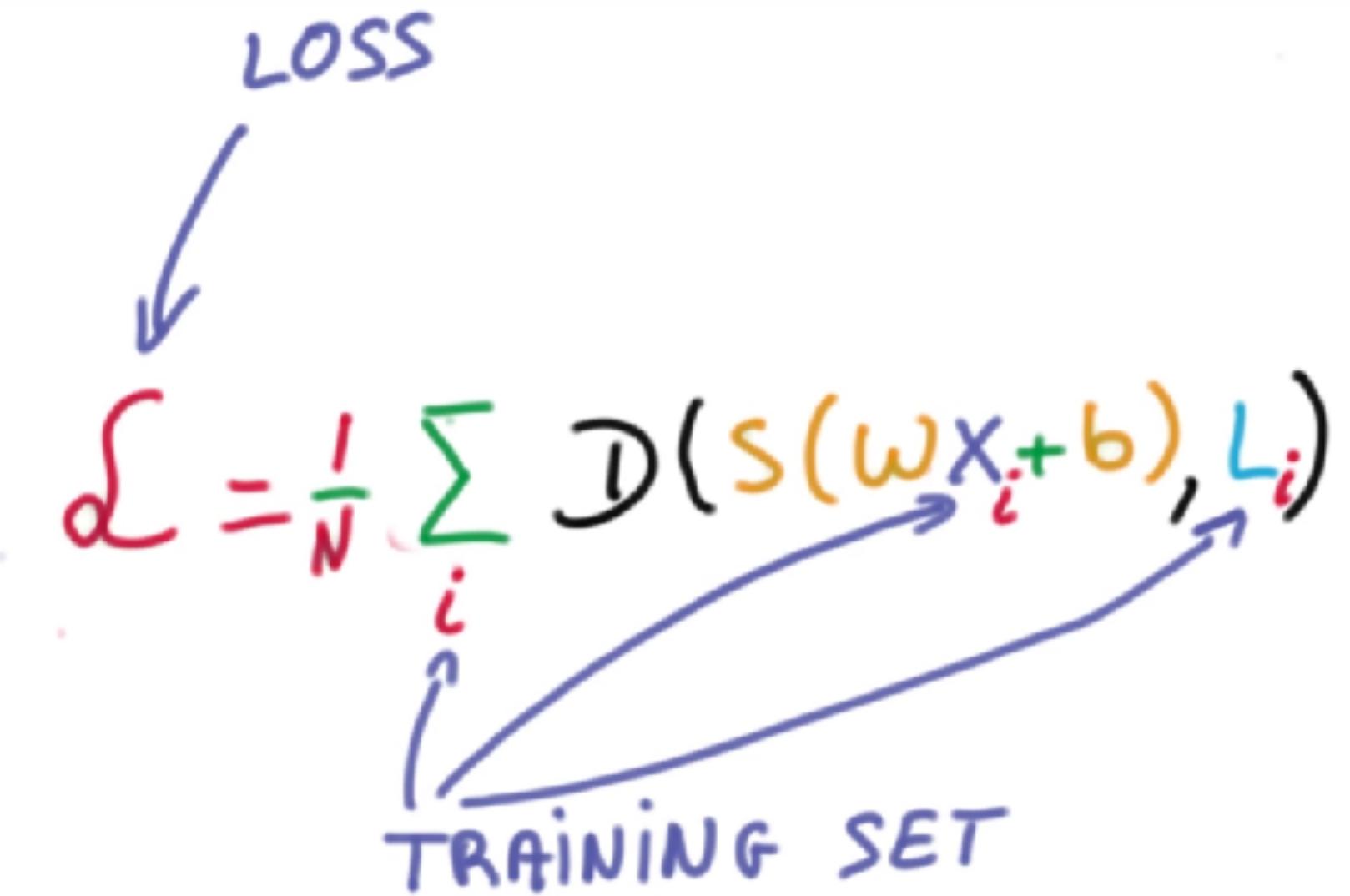
```
# Construct model
# https://www.tensorflow.org/versions/r0.7/tutorials/mnist/beginners/inde
# First, we multiply x by W with the expression tf.matmul(x, W).
# This is flipped from when we multiplied them in our equation,
# where we had Wx, as a small trick to deal with x being
# a 2D tensor with multiple inputs.
#💡 We then add b, and finally apply tf.nn.softmax.
hypothesis = tf.nn.softmax(tf.matmul(w, X)) # Softmax
```

Cost function

```
# Minimize error using cross entropy  
learning_rate = 0.001
```

```
# Cross entropy  
cost = tf.reduce_mean(-tf.reduce_sum(Y*tf.log(hypothesis)), reduction_indices=1)
```

```
# Gradient Descent  
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
```



STEP

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

```

xy = np.loadtxt('train.txt', unpack=True, dtype='float32')
x_data = np.transpose(xy[0:3])
y_data = np.transpose(xy[3:])

#tf Graph Input
X = tf.placeholder("float", [None, 3]) # x1, x2 and 1 (for bias)
Y = tf.placeholder("float", [None, 3]) # A, B, C => 3 classes
# Set model weights
W = tf.Variable(tf.zeros([3, 3]))

# Construct model
hypothesis = tf.nn.softmax(tf.matmul(W, X)) # Softmax

# Minimize error using cross entropy
learning_rate = 0.001

# Cross entropy
cost = tf.reduce_mean(-tf.reduce_sum(Y*tf.log(hypothesis), reduction_indices=1))

# Gradient Descent
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

# Initializing the variables
init = tf.initialize_all_variables()

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

    for step in xrange(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}), sess.run(W)

```

#	x0	x1	x2	y	[A	B	C]
1	2	1	0	0	0	1	
1	3	2	0	0	0	1	
1	3	4	0	0	0	1	
1	5	5	0	1	0		
1	7	5	0	1	0		
1	2	5	0	1	0		
1	6	6	1	0	0		
1	7	7	1	0	0		

Test & one-hot encoding

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

b = sess.run(hypothesis, feed_dict={X: [[1, 3, 4]]})
print b, sess.run(tf.argmax(b, 1))

c = sess.run(hypothesis, feed_dict={X: [[1, 1, 0]]})
print c, sess.run(tf.argmax(c, 1))

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

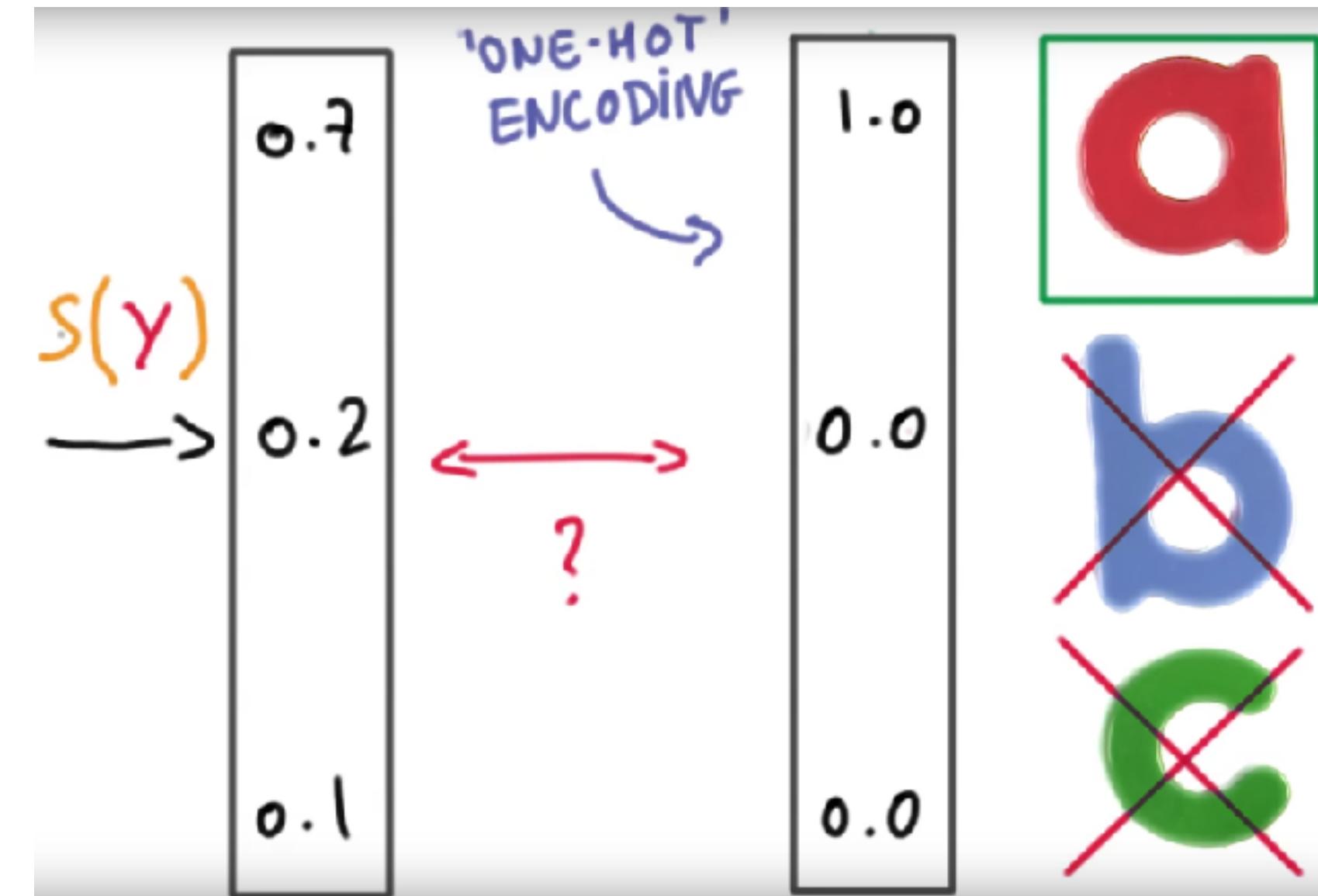
Test & one-hot encoding

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

b = sess.run(hypothesis, feed_dict={X: [[1, 3, 4]]})
print b, sess.run(tf.argmax(b, 1))

c = sess.run(hypothesis, feed_dict={X: [[1, 1, 0]]})
print c, sess.run(tf.argmax(c, 1))

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



Lab 7

Learning rate, Evaluation

Sung Kim <hunkim+ml@gmail.com>

Examples

- <https://github.com/aymericdamien/TensorFlow-Examples>

Learning rate

```
# Minimize error using cross entropy
learning_rate = 10.

# Cross entropy
cost = tf.reduce_mean(-tf.reduce_sum(Y*tf.log(hypothesis), reduction_indices=1))

# Gradient Descent
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

# Initializing the variables
init = tf.initialize_all_variables()

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

    for step in xrange(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}), sess.run(w)
```

Learning rate

```
0 nan [[-0.83333319  0.4166666  0.41666645]
      [ 1.66666687  2.91666746 -4.58333397]
      [ 1.66666627  4.16666698 -5.83333397]]
200 nan [[ nan  nan  nan]
           [ nan  nan  nan]
           [ nan  nan  nan]]
400 nan [[ nan  nan  nan]
           [ nan  nan  nan]
           [ nan  nan  nan]]
```

MNIST Dataset



[train-images-idx3-ubyte.gz](#): training set images (9912422 bytes)

[train-labels-idx1-ubyte.gz](#): training set labels (28881 bytes)

[t10k-images-idx3-ubyte.gz](#): test set images (1648877 bytes)

[t10k-labels-idx1-ubyte.gz](#): test set labels (4542 bytes)

Reading data

```
"""Functions for downloading and reading MNIST data."""
import ...
SOURCE_URL = 'http://yann.lecun.com/exdb/mnist/'
def maybe_download(filename, work_directory):
    """Download the data from Yann's website, unless it's already here."""
    if not os.path.exists(work_directory):
        os.mkdir(work_directory)
    filepath = os.path.join(work_directory, filename)
    if not os.path.exists(filepath):
        filepath, _ = urllib.urlretrieve(SOURCE_URL + filename, filepath)
        statinfo = os.stat(filepath)
        print('Successfully downloaded', filename, statinfo.st_size, 'bytes.')
    return filepath

def _read32(bytestream):
    dt = numpy.dtype(numpy.uint32).newbyteorder('>')
    return numpy.frombuffer(bytestream.read(4), dtype=dt)

def extract_images(filename):
    """Extract the images into a 4D uint8 numpy array [index, y, x, depth]."""
    print('Extracting', filename)
    with gzip.open(filename) as bytestream:
        magic = _read32(bytestream)
        if magic != 2051:
            raise ValueError(
                'Invalid magic number %d in MNIST image file: %s' %
                (magic, filename))
        num_images = _read32(bytestream)
        rows = _read32(bytestream)
        cols = _read32(bytestream)
        buf = bytestream.read(rows * cols * num_images)
        data = numpy.frombuffer(buf, dtype=numpy.uint8)
```

Reading data and set variables

```
# tf Graph Input
x = tf.placeholder("float", [None, 784]) # mnist data image of shape 28*28=784
y = tf.placeholder("float", [None, 10]) # 0-9 digits recognition => 10 classes

# Create model

# Set model weights
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
```

Activation (hypothesis) and cost

```
# Construct model
activation = tf.nn.softmax(tf.matmul(x, w) + b) # Softmax

# Minimize error using cross entropy
cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(activation)), reduction_indices=1)
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost) # Gradient descent
```

Training batch

```
# Training cycle
for epoch in range(training_epochs):
    avg_cost = 0.
    total_batch = int(mnist.train.num_examples/batch_size)
    # Loop over all batches
    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        # Fit training using batch data
        sess.run(optimizer, feed_dict={x: batch_xs, y: batch_ys})
        # Compute average loss
        avg_cost += sess.run(cost, feed_dict={x: batch_xs, y: batch_ys})/total_batch
    # Display logs per epoch step
    if epoch % display_step == 0:
        print "Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}".format(avg_cost)

print "Optimization Finished!"
```

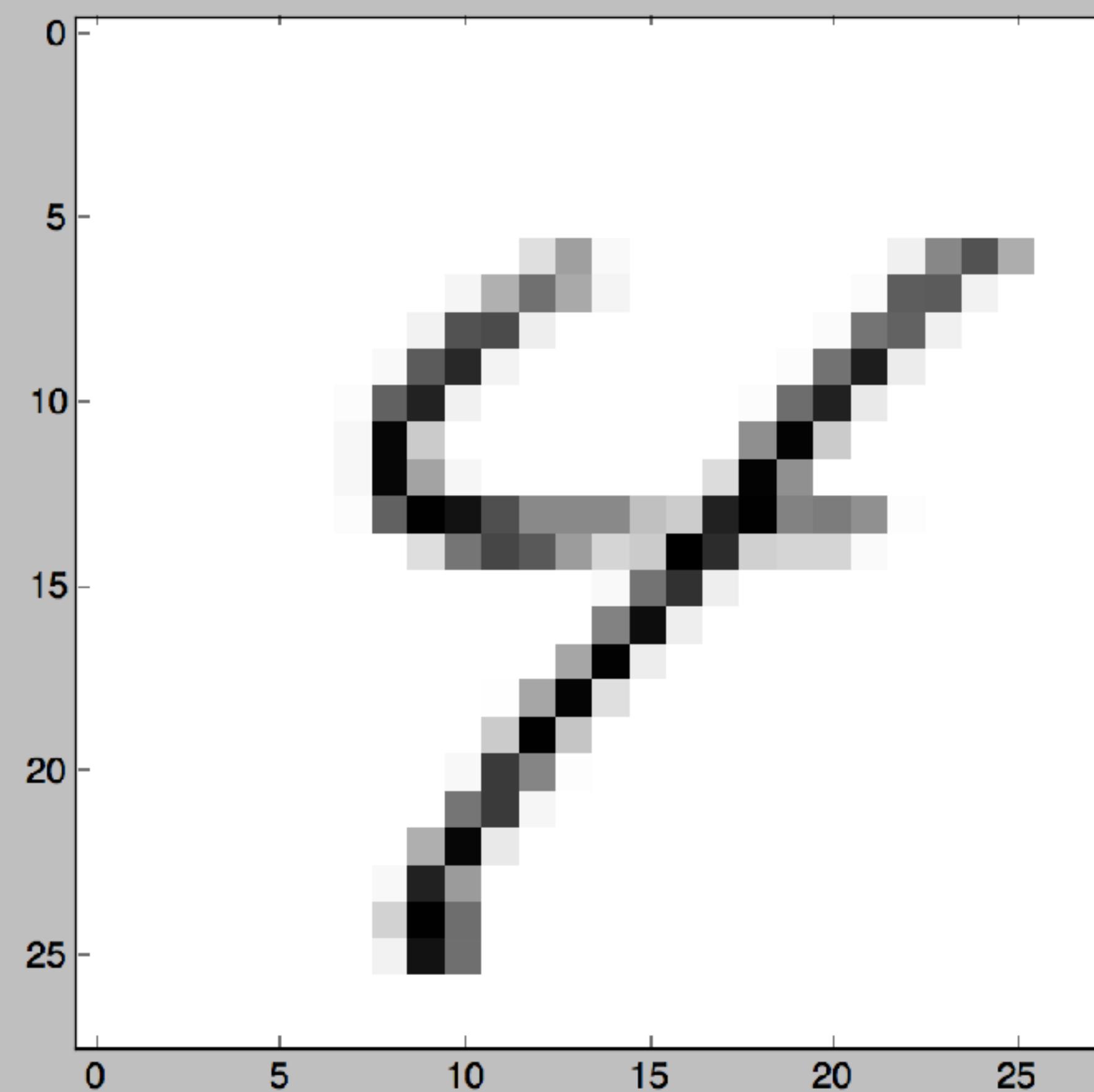
Predict & Show

```
# Get one and predict
r = randint(0, mnist.test.num_examples -1)
print "Label: ", sess.run(tf.argmax(mnist.test.labels[r:r+1], 1))
print "Prediction: ", sess.run(tf.argmax(activation,1), {x: mnist.test.images[r:r+1]})

# Show the img
plt.imshow(mnist.test.images[r:r+1].reshape(28, 28), cmap='Greys', interpolation='nearest')
plt.show()
```

Figure 1

02-mnist-show



&Tips/01-lettergrade.py

03-mnist-eval.py

02-mnist-show.py

```
is):  
  
    print("Total Examples: ", mnist.train.images.shape[0])  
    print("Batch Size: ", batch_size)  
    print("Total Batches: ", int(mnist.train.images.shape[0]/batch_size))  
  
    for epoch in range(0, num_epochs):  
        avg_cost = 0.  
        total_batch = int(mnist.train.images.shape[0]/batch_size)  
  
        for i in range(0, total_batch):  
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)  
            _, feed_dict={x: batch_xs, y: batch_ys})  
  
            avg_cost += sess.run(cost, feed_dict=feed_dict)  
            total_cost = avg_cost/total_batch  
  
            if epoch % display_step == 0:  
                print("Epoch: " + str(epoch+1), "cost=", "{:.9f}".format(avg_cost))  
  
    print("Optimization Finished!")  
  
    r = 25  
    activation = sess.run(y, feed_dict={x: mnist.test.images[r:r+1]})  
    print("Label: ", np.argmax(mnist.test.labels[r:r+1], 1))  
    print("Prediction: ", np.argmax(activation, 1))
```



Epoch: 0024 cost= 0.334489702
Epoch: 0025 cost= 0.332481820
Optimization Finished!
Label: [4]
Prediction: [4]

Evaluation

```
print "Optimization Finished!"  
  
# Test model  
correct_prediction = tf.equal(tf.argmax(activation, 1), tf.argmax(y, 1))  
# Calculate accuracy  
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))  
print "Accuracy:", accuracy.eval({x: mnist.test.images, y: mnist.test.labels})
```

Evaluation

```
print "Optimization Finished!"  
  
# Test model  
correct_prediction = tf.equal(tf.argmax(activation, 1), tf.argmax(y, 1))  
# Calculate accuracy  
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))  
print "Accuracy:", accuracy.eval({x: mnist.test.images, y: mnist.test.labels})
```

```
.  
Epoch: 0015 cost= 0.361379657  
Epoch: 0016 cost= 0.357280465  
Epoch: 0017 cost= 0.353490356  
Epoch: 0018 cost= 0.350175499  
Epoch: 0019 cost= 0.347017571  
Epoch: 0020 cost= 0.344151569  
Epoch: 0021 cost= 0.341454394  
Epoch: 0022 cost= 0.339011900  
Epoch: 0023 cost= 0.336664148  
Epoch: 0024 cost= 0.334490953  
Epoch: 0025 cost= 0.332419457  
Optimization Finished!  
Accuracy: 0.9146
```

Lab 9- I

NN for XOR

Sung Kim <hunkim+ml@gmail.com>

Data set

#	XOR		
#	x1	x2	y
0	0		0
0	1		1
1	0		1
1	1		0

XOR with logistic regression?

```
# http://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.loadtxt.html
xy = np.loadtxt('train.txt', unpack=True)
x_data = xy[0:-1]
y_data = xy[-1]

X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)

W = tf.Variable(tf.random_uniform([1, len(x_data)], -1.0, 1.0))

# Our hypothesis
h = tf.matmul(W, X)
hypothesis = tf.div(1., 1.+tf.exp(-h))
# cost function
cost = -tf.reduce_mean(Y*tf.log(hypothesis) + (1-Y)*tf.log(1-hypothesis))

# Minimize
a = tf.Variable(0.01) # Learning rate, alpha
optimizer = tf.train.GradientDescentOptimizer(a)
train = optimizer.minimize(cost)

# Before starting, initialize the variables. We will 'run' this first.
init = tf.initialize_all_variables()
```

XOR with logistic regression?

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

# Fit the line.
for step in xrange(1000):
    sess.run(train, 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}), sess.run(W)

# Test model
correct_prediction = tf.equal(tf.floor(hypothesis+0.5), Y)
# Calculate accuracy
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
print sess.run([hypothesis, tf.floor(hypothesis+0.5), correct_prediction, accuracy], feed_dict={X:x_data, Y:y_data})
print "Accuracy:", accuracy.eval({X:x_data, Y:y_data})
```

Does not work!

```
print sess.run([hypothesis, tf.floor(hypothesis+0.5), correct_prediction, accuracy], feed_dict={X:x_data, Y:y_data})
print "Accuracy:", accuracy.eval({X:x_data, Y:y_data})
```



```
0 0.707739 [[-0.5306738  0.10871095]]
200 0.702072 [[-0.42843163  0.13686776]]
400 0.698917 [[-0.35024348  0.14920615]]
600 0.697075 [[-0.28977284  0.15132181]]
800 0.695938 [[-0.24237214  0.14709207]]
[array([[ 0.5        ,  0.53474092,  0.44896561,  0.48359016]], dtype=float32), array([[ 1.,  1.,  0.,  0.]])
Accuracy: 0.5
```

NN

```
# http://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.loadtxt.html
xy = np.loadtxt('train.txt', unpack=True)
x_data = xy[0:-1]
y_data = xy[-1]

X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)

W = tf.Variable(tf.random_uniform([1, len(x_data)], -1.0, 1.0))

# Our hypothesis
h = tf.matmul(W, X)
hypothesis = tf.div(1., 1.+tf.exp(-h))

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

# Minimize
a = tf.Variable(0.01) # Learning rate, alpha
optimizer = tf.train.GradientDescentOptimizer(a)
train = optimizer.minimize(cost)

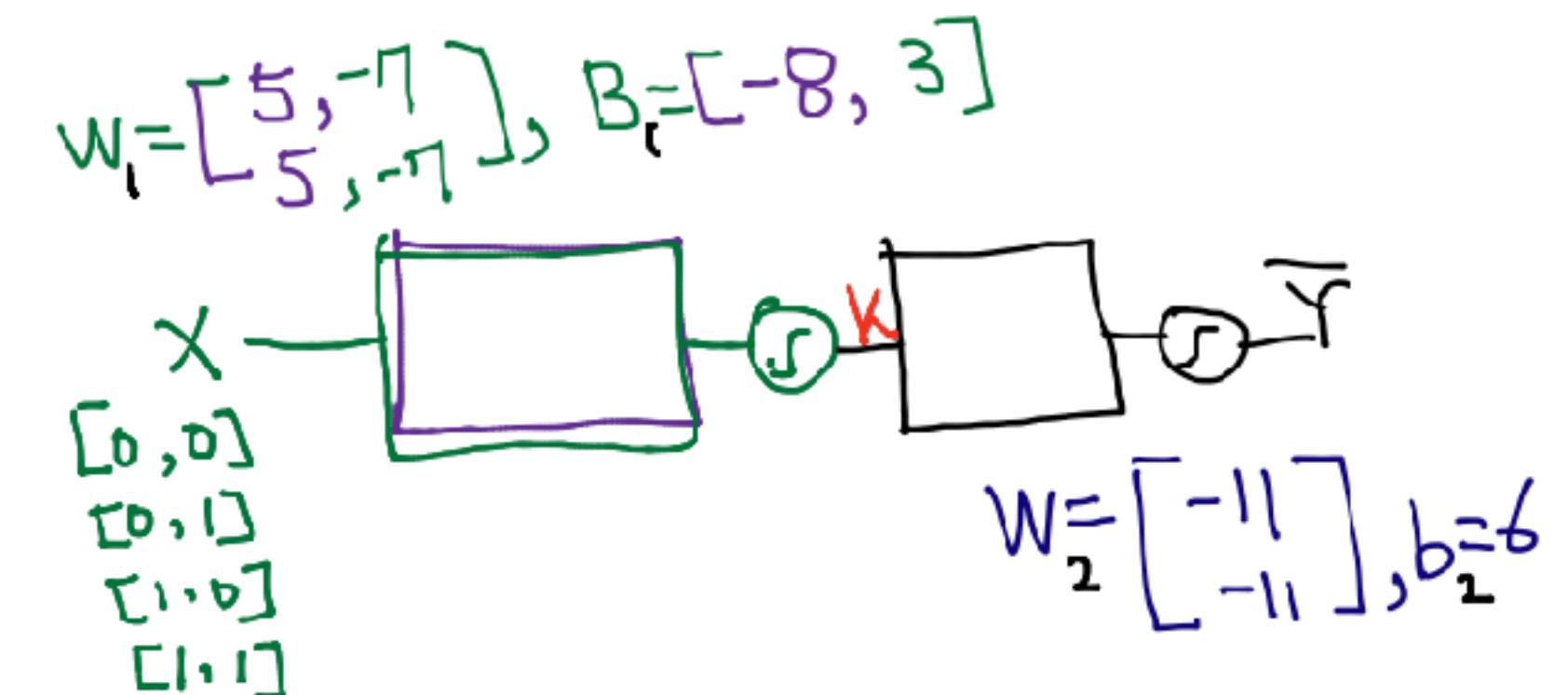
# Before starting, initialize the variables. We will 'run' this first.
init = tf.initialize_all_variables()

W1 = tf.Variable(tf.random_uniform([2, 2], -1.0, 1.0))
W2 = tf.Variable(tf.random_uniform([2, 1], -1.0, 1.0))

b1 = tf.Variable(tf.zeros([2]), name="Bias1")
b2 = tf.Variable(tf.zeros([1]), name="Bias2")

# Our hypothesis
L2 = tf.sigmoid(tf.matmul(X, W1) + b1)
hypothesis = tf.sigmoid(tf.matmul(L2, W2) + b2)
```

NN for XOR



```
W1 = tf.Variable(tf.random_uniform([2, 2], -1.0, 1.0))
W2 = tf.Variable(tf.random_uniform([2, 1], -1.0, 1.0))

b1 = tf.Variable(tf.zeros([2]), name="Bias1")
b2 = tf.Variable(tf.zeros([1]), name="Bias2")

# Our hypothesis
L2 = tf.sigmoid(tf.matmul(X, W1) + b1)
hypothesis = tf.sigmoid(tf.matmul(L2, W2) + b2)
```

NN for XOR

```
198000 cost 0.00878375
    predict [[ 0.01067145]
[ 0.99207062]
[ 0.99207932]
[ 0.00845705]]
    W1, B1 [array([[ 6.84843874,  5.02159739],
[ 6.83831406,  5.0197401 ]], dtype=float32), array([-5.00321722], dtype=float32)]
    W2, B2 [array([[ 10.83042145],
[-11.59721661]]], dtype=float32), array([-5.00321722], dtype=float32)]
[array([[ 0.],
[ 1.],
[ 1.],
[ 0.]], dtype=float32), array([[ True],
[ True],
[ True],
[ True]]], dtype=bool)]
Accuracy: 1.0
```

Wide NN for XOR

```
W1 = tf.Variable(tf.random_uniform([2, 10], -1.0, 1.0))
W2 = tf.Variable(tf.random_uniform([10, 1], -1.0, 1.0))

b1 = tf.Variable(tf.zeros([10]), name="Bias1")
b2 = tf.Variable(tf.zeros([1]), name="Bias2")

# Our hypothesis
L2 = tf.sigmoid(tf.matmul(X, W1) + b1)
hypothesis = tf.sigmoid(tf.matmul(L2, W2) + b2)
```

Wide NN for XOR

```
198000 cost 0.00357563
    predict [[ 0.00214934]
[ 0.99605185]
[ 0.99672902]
[ 0.00490645]]
W1, B1 [array([[-1.64474547,  0.8943826 ,  0.28936383, -0.56383854,  3.85756803,
   -2.07212377,  1.19432867, -1.34555447,  6.40112543,  6.01214457],
[-1.12384748,  0.70554543,  1.48841226, -0.80537623, -5.49986124,
   4.09881163,  0.32355428, -0.79951024, -4.84555626,  6.11010885]], dtype=float32), array([-0.45977819], dtype=float32)]
W2, B2 [array([[ 2.93114686],
[-1.35010254],
[-1.59123349],
[ 1.30858362],
[ 7.05565166],
[-4.19236469],
[-1.34470844],
[ 2.38127279],
[-9.47126675],
[ 9.30797291]], dtype=float32), array([-0.45977819], dtype=float32)]
[array([[ 0.],
[ 1.],
[ 1.],
[ 0.]], dtype=float32), array([[ True],
[ True],
[ True],
[ True]], dtype=bool)]
Accuracy: 1.0
```

Deep NN for XOR

```
W1 = tf.Variable(tf.random_uniform([2, 5], -1.0, 1.0))
W2 = tf.Variable(tf.random_uniform([5, 4], -1.0, 1.0))
W3 = tf.Variable(tf.random_uniform([4, 1], -1.0, 1.0))

b1 = tf.Variable(tf.zeros([5]), name="Bias1")
b2 = tf.Variable(tf.zeros([4]), name="Bias2")
b3 = tf.Variable(tf.zeros([1]), name="Bias2")

# Our hypothesis
L2 = tf.sigmoid(tf.matmul(X, W1) + b1)
L3 = tf.sigmoid(tf.matmul(L2, W2) + b2)
hypothesis = tf.sigmoid(tf.matmul(L3, W3) + b3)
```

Deep NN for XOR

```
198000 cost 0.00111391
predict [[ 5.58296102e-04]
[ 9.98885453e-01]
[ 9.98768628e-01]
[ 1.54864462e-03]]
W1, B1 [array([[ 5.39242172,  2.17747116,  5.26371241, -0.18648638, -0.05985435],
   [ 4.86114264, -4.64419937, -3.61568236, -1.43215752, -2.03103042]], dtype=float32), array([ 0.
W2, B2 [array([[[-2.3721838 , -3.82411337,  2.479671 ,  1.95326269],
   [-2.15816069, -3.55406022,  2.87128544,  2.1189158 ],
   [ 2.92693424,  4.71013832, -3.4270494 , -2.09762478],
   [-0.03848177, -0.86011964,  1.32942307,  0.62142414],
   [-1.01190925, -1.92403996,  0.65061325, -0.27596042]], dtype=float32), array([ 0.70831299,  1.
[array([[ 0.],
[ 1.],
[ 1.],
[ 0.]], dtype=float32), array([[ True],
[ True],
[ True],
[ True]]], dtype=bool)]
Accuracy: 1.0|
```

Let's go deep & wide!

```
W1 = tf.Variable(tf.random_uniform([2, 5], -1.0, 1.0))
W2 = tf.Variable(tf.random_uniform([5, 4], -1.0, 1.0))
W3 = tf.Variable(tf.random_uniform([4, 1], -1.0, 1.0))

b1 = tf.Variable(tf.zeros([5]), name="Bias1")
b2 = tf.Variable(tf.zeros([4]), name="Bias2")
b3 = tf.Variable(tf.zeros([1]), name="Bias2")

# Our hypothesis
L2 = tf.sigmoid(tf.matmul(X, W1) + b1)
L3 = tf.sigmoid(tf.matmul(L2, W2) + b2)
hypothesis = tf.sigmoid(tf.matmul(L3, W3) + b3)
```

Lab 9-2

Tensorboard

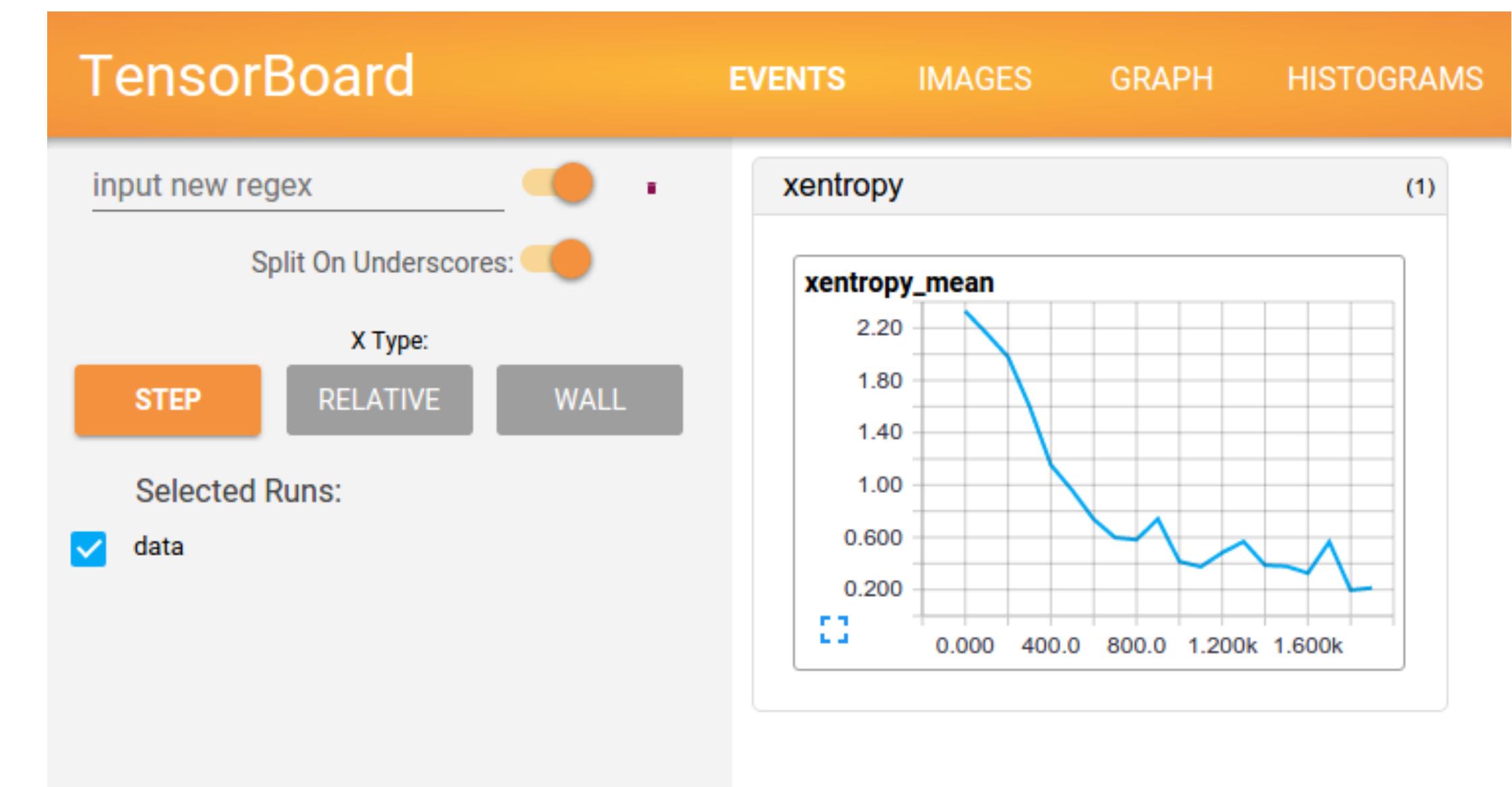
Sung Kim <hunkim+ml@gmail.com>

Visualizing your Deep learning using TensorBoard (TensorFlow)

Sung Kim <hunkim+ml@gmail.com>

TensorBoard: TF logging/debugging tool

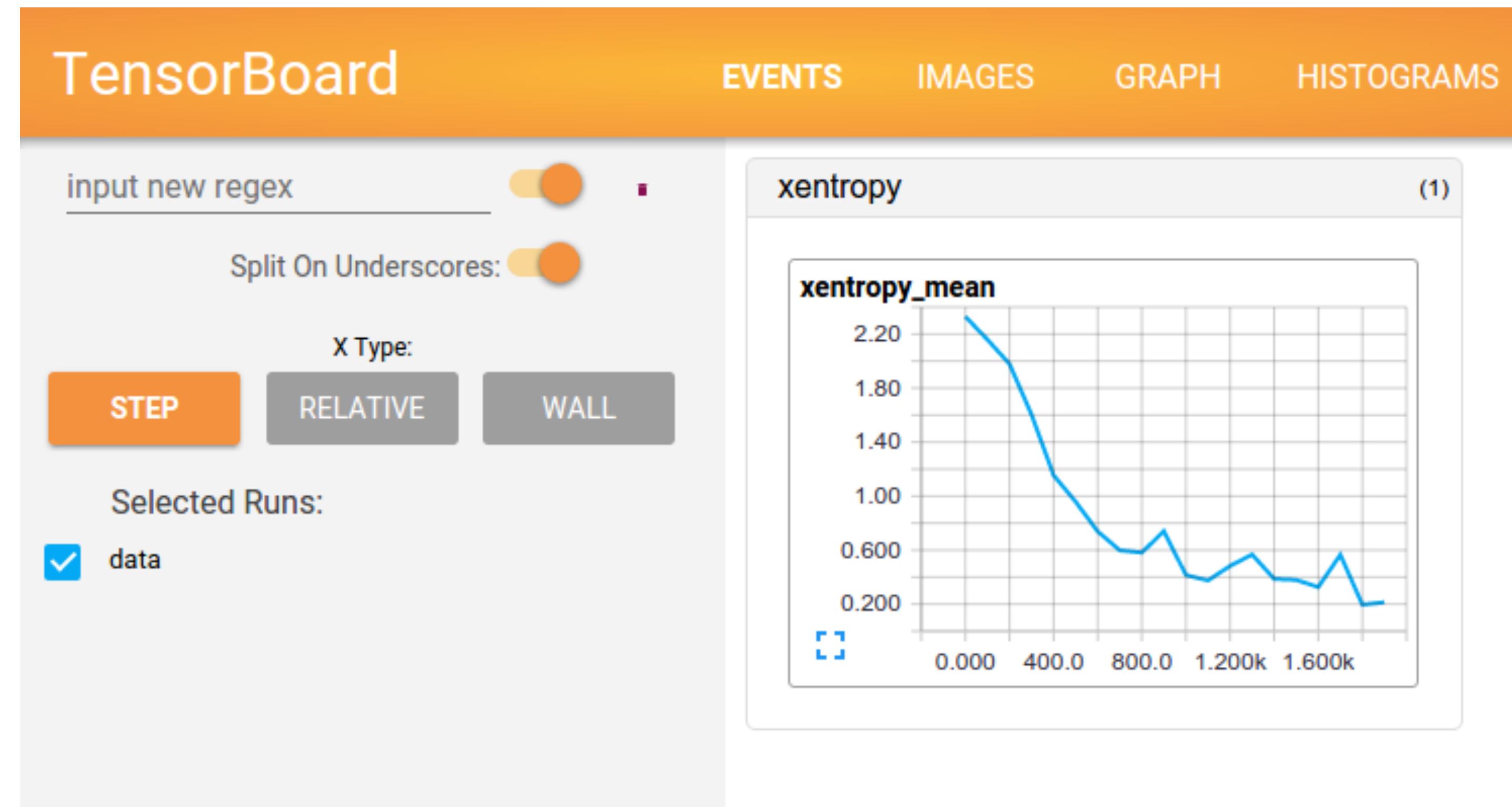
- Visualize your TF graph
- Plot quantitative metrics
- Show additional data



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!



5 steps of using tensorboard

- From TF graph, decide which node you want to annotate
 - with `tf.name_scope("test")` as scope:
 - `tf.histogram_summary("weights", W), tf.scalar_summary("accuracy", accuracy)`
- Merge all summaries
 - `merged = tf.merge_all_summaries()`
- Create writer
 - `writer = tf.train.SummaryWriter("/tmp/mnist_logs", sess.graph_def)`
- Run summary merge and `add_summary`
 - `summary = sess.run(merged, ...); writer.add_summary(summary);`
- Launch Tensorboard
 - `tensorboard --logdir=/tmp/mnist_logs`

Name variables

```
X = tf.placeholder(tf.float32, name = 'X-input')
Y = tf.placeholder(tf.float32, name = 'Y-input')

W1 = tf.Variable(tf.random_uniform([2, 2], -1.0, 1.0), name = "Weight1")
W2 = tf.Variable(tf.random_uniform([2, 1], -1.0, 1.0), name = "Weight2")

b1 = tf.Variable(tf.zeros([2]), name="Bias1")
b2 = tf.Variable(tf.zeros([1]), name="Bias2")
```

Add scope for better graph hierarch

```
# Our hypothesis
with tf.name_scope("layer2") as scope:
    L2 = tf.sigmoid(tf.matmul(X, W1) + b1)

with tf.name_scope("layer3") as scope:
    hypothesis = tf.sigmoid(tf.matmul(L2, W2) + b2)

# cost function
with tf.name_scope("cost") as scope:
    cost = -tf.reduce_mean(Y*tf.log(hypothesis) + (1-Y)*tf.log(1-hypothesis))
    cost_summ = tf.scalar_summary("cost", cost)

# Minimize
with tf.name_scope("train") as scope:
    optimizer = tf.train.GradientDescentOptimizer(0.01) # Learning rate, alpha
    train = optimizer.minimize(cost)
```

Add histogram

```
w1_hist = tf.histogram_summary("weights1", W1)
w2_hist = tf.histogram_summary("weights2", W2)

b1_hist = tf.histogram_summary("biases1", b1)
b2_hist = tf.histogram_summary("biases2", b2)

y_hist = tf.histogram_summary("y", Y)
```

Add scalar variables

```
# cost function
with tf.name_scope("cost") as scope:
    cost = -tf.reduce_mean(Y*tf.log(hypothesis) + (1-Y)*tf.log(1-hypothesis))
    cost_summ = tf.scalar_summary("cost", cost)
```

Merge summaries and create writer after creating session

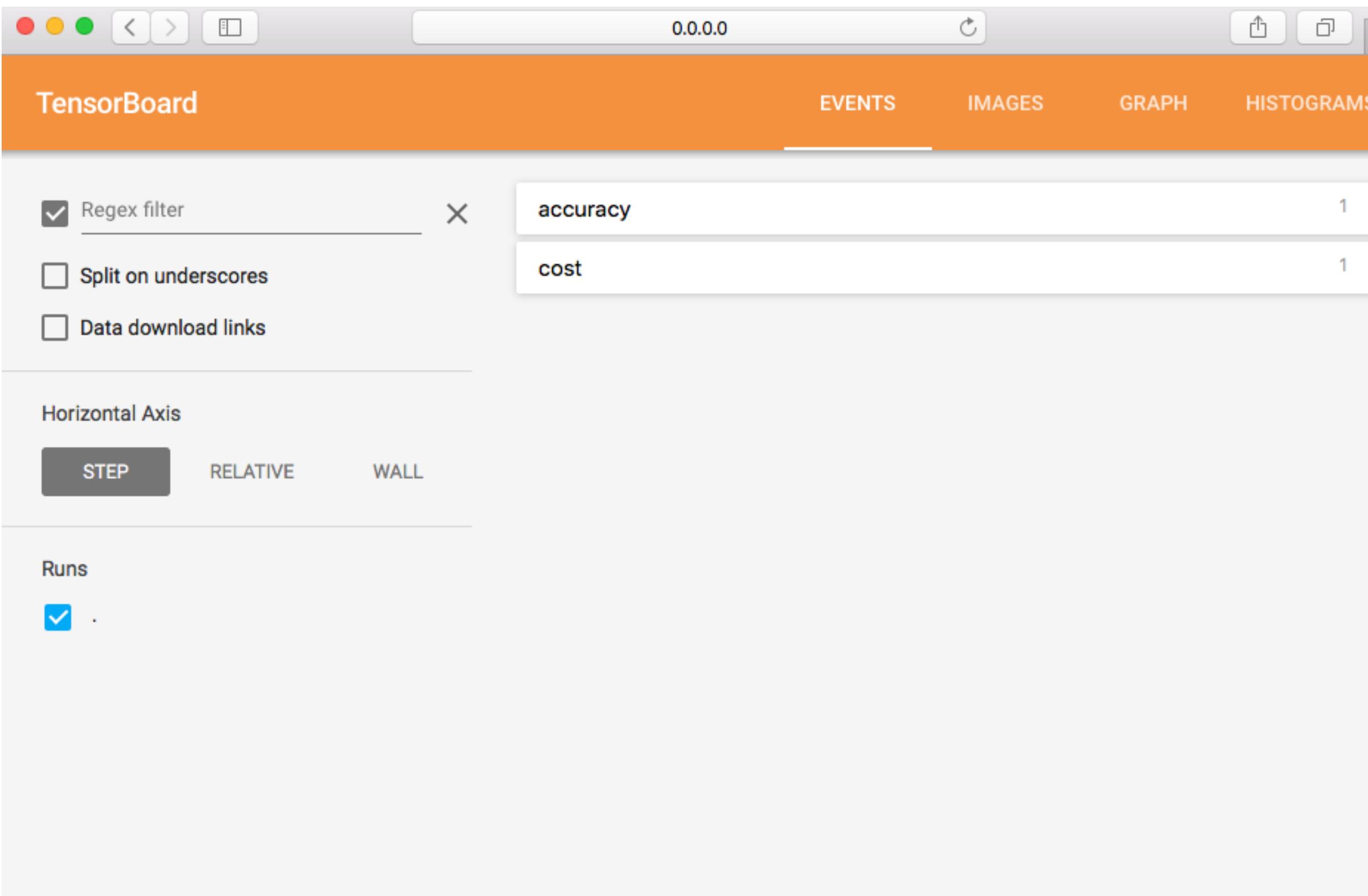
```
# Launch the graph.  
with tf.Session() as sess:  
  
    #tensorboard --logdir=./logs/xor_logs  
    merged = tf.merge_all_summaries()  
    writer = tf.train.SummaryWriter("./logs/xor_logs", sess.graph_def)
```

Run merged summary and write (add summary)

```
# Fit the line.  
for step in xrange(200000):  
    summary, _ = sess.run([merged, train], feed_dict={X:x_data, Y:y_data})  
    writer.add_summary(summary, step)  
  
# Fit the line.  
for step in xrange(200000):  
    sess.run(train, feed_dict={X:x_data, Y:y_data})  
    if step % 2000 == 0:      [merged, train]  
        summary = sess.run(merged, feed_dict={X:x_data, Y:y_data})  
        writer.add_summary(summary, step)
```

Launch tensorflow

- `tensorboard —logdir=/tmp/mnist_logs`
- (You can navigate to `http://0.0.0.0:6006`)



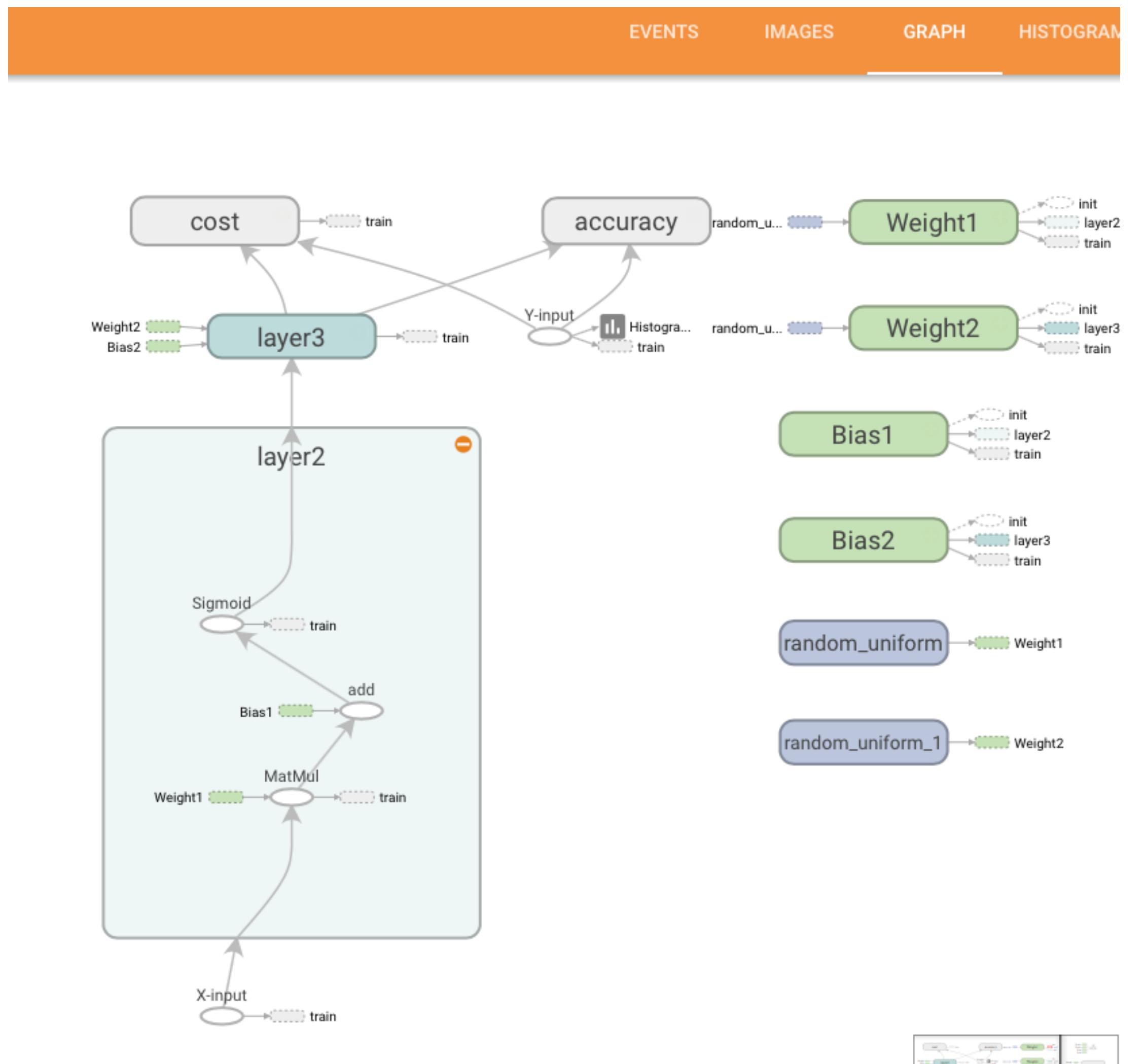
Add scope for better graph hierarch

```
# Our hypothesis
with tf.name_scope("layer2") as scope:
    L2 = tf.sigmoid(tf.matmul(X, W1) + b1)

with tf.name_scope("layer3") as scope:
    hypothesis = tf.sigmoid(tf.matmul(L2, W2) + b2)

# cost function
with tf.name_scope("cost") as scope:
    cost = -tf.reduce_mean(Y*tf.log(hypothesis) + (1-Y)*tf.log(1-hypothesis))
    cost_summ = tf.scalar_summary("cost", cost)

# Minimize
with tf.name_scope("train") as scope:
    optimizer = tf.train.GradientDescentOptimizer(0.01) # Learning rate, alpha
    train = optimizer.minimize(cost)
```



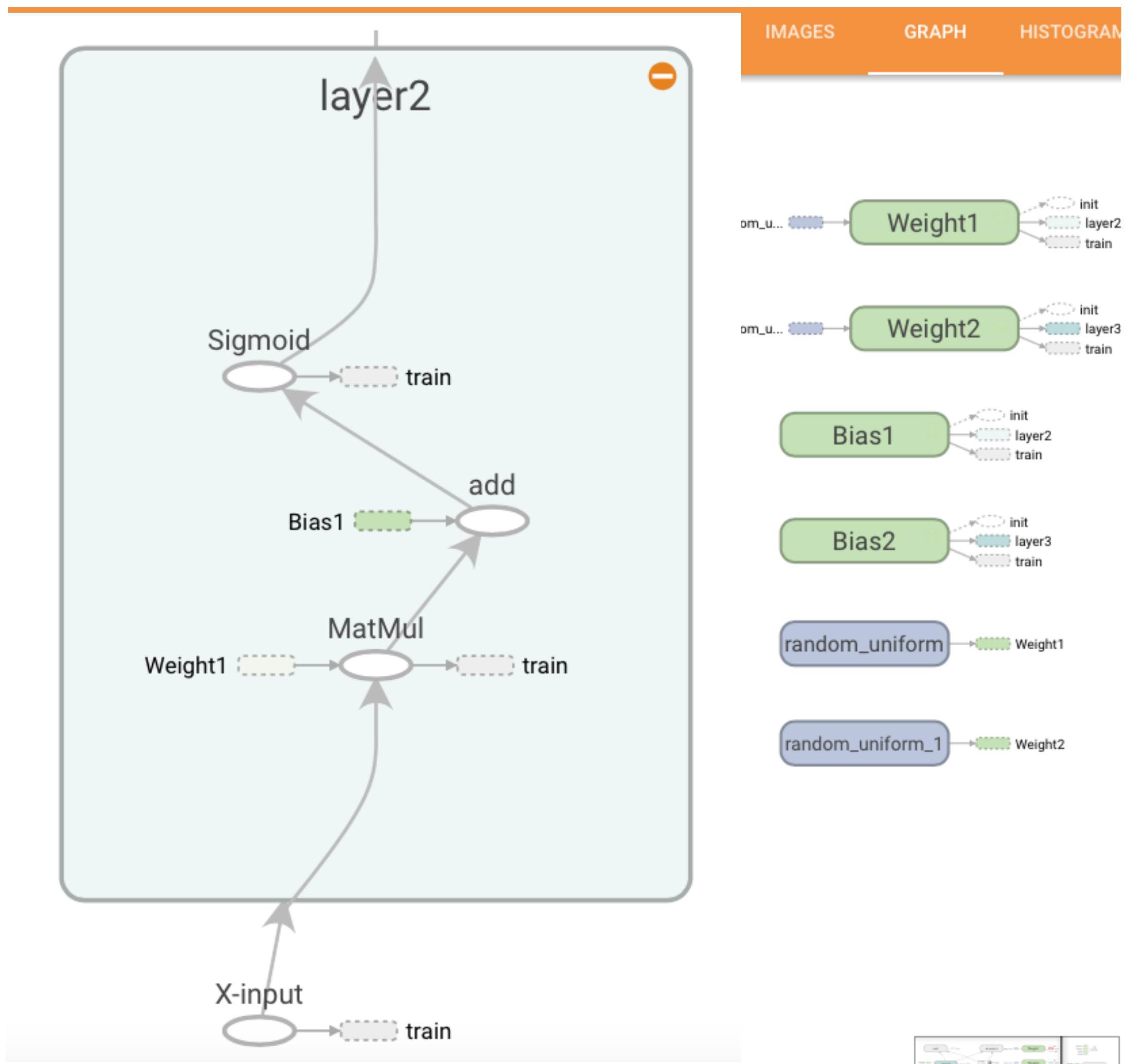
Add scope for better graph hierarch

```
# Our hypothesis
with tf.name_scope("layer2") as scope:
    L2 = tf.sigmoid(tf.matmul(X, W1) + b1)

with tf.name_scope("layer3") as scope:
    hypothesis = tf.sigmoid(tf.matmul(L2, W2) + b2)

# cost function
with tf.name_scope("cost") as scope:
    cost = -tf.reduce_mean(Y*tf.log(hypothesis) + (1-Y)*tf.log(1-hypothesis))
    cost_summ = tf.scalar_summary("cost", cost)

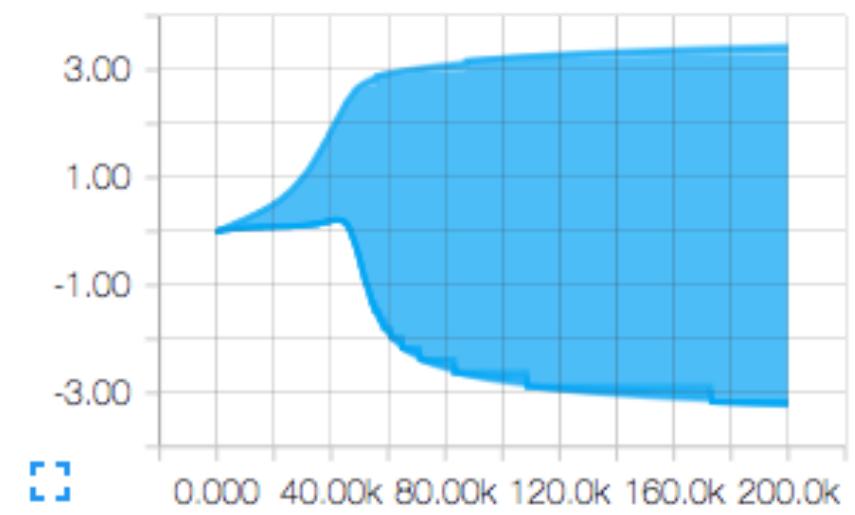
# Minimize
with tf.name_scope("train") as scope:
    optimizer = tf.train.GradientDescentOptimizer(0.01) # Learning rate, alpha
    train = optimizer.minimize(cost)
```



biases1

1

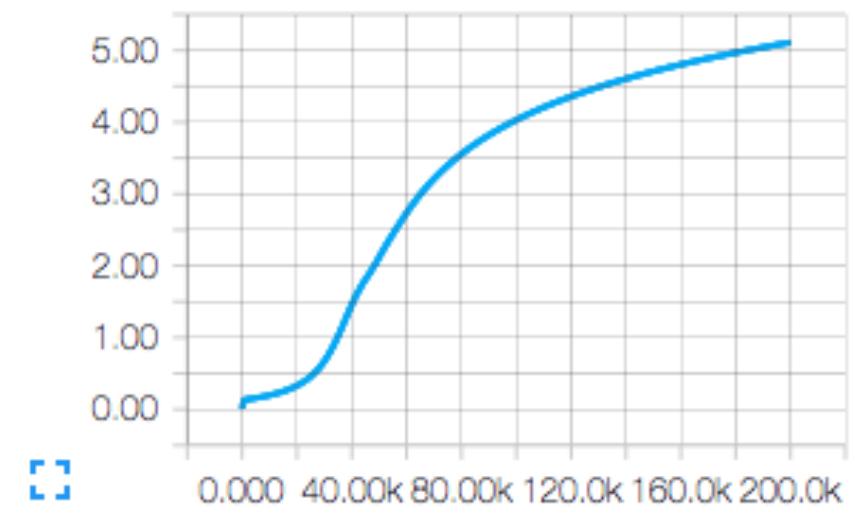
biases1



biases2

1

biases2



```
w1_hist = tf.histogram_summary("weights1", W1)
w2_hist = tf.histogram_summary("weights2", W2)

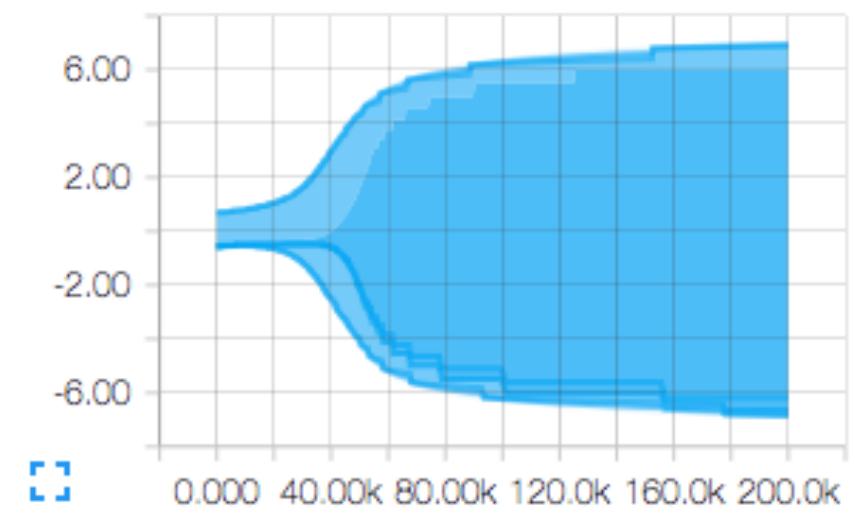
b1_hist = tf.histogram_summary("biases1", b1)
b2_hist = tf.histogram_summary("biases2", b2)

y_hist = tf.histogram_summary("y", Y)
```

weights1

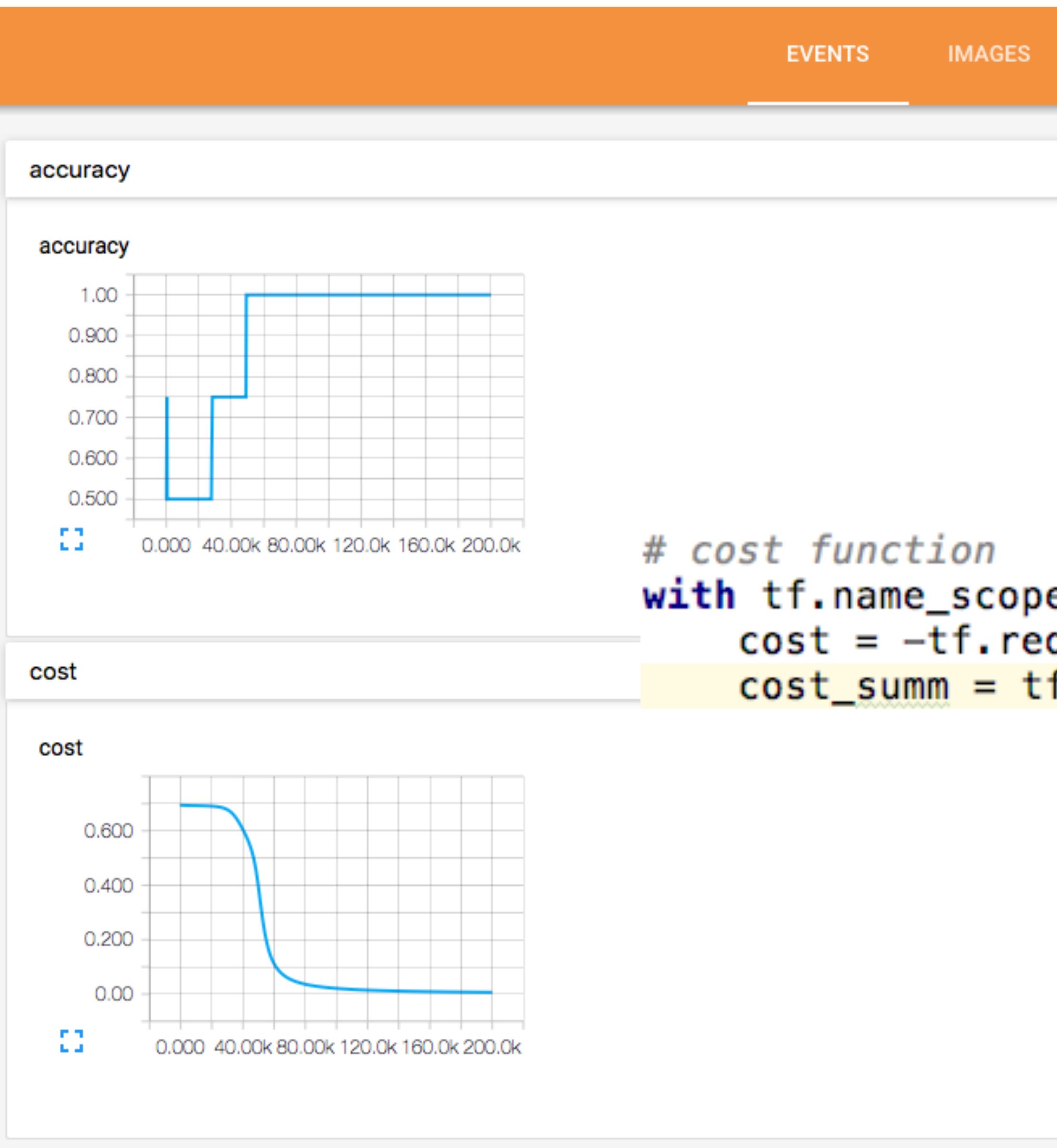
1

weights1



Add histogram

Add scalar variables



```
# cost function
with tf.name_scope("cost") as scope:
    cost = -tf.reduce_mean(Y*tf.log(hypothesis) + (1-Y)*tf.log(1-hypothesis))
    cost_summ = tf.scalar_summary("cost", cost)
```

5 steps of using tensorboard

- From TF graph, decide which node you want to annotate
 - with `tf.name_scope("test")` as scope:
 - `tf.histogram_summary("weights", W), tf.scalar_summary("accuracy", accuracy)`
- Merge all summaries
 - `merged = tf.merge_all_summaries()`
- Create writer
 - `writer = tf.train.SummaryWriter("/tmp/mnist_logs", sess.graph_def)`
- Run summary merge and `add_summary`
 - `summary = sess.run(merged, ...); writer.add_summary(summary);`
- Launch Tensorboard
 - `tensorboard --logdir=/tmp/mnist_logs`

Lab 10

NN, ReLu, Xavier, Dropout, and Adam

Sung Kim <hunkim+ml@gmail.com>
<http://hunkim.github.io/ml/>

Examples

- <https://github.com/aymericdamien/TensorFlow-Examples>

Softmax classifier for MNIST

```
# tf Graph Input
x = tf.placeholder("float", [None, 784]) # mnist data image of shape 28*28=784
y = tf.placeholder("float", [None, 10]) # 0-9 digits recognition => 10 classes

# Create model

# Set model weights
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))

# Construct model
activation = tf.nn.softmax(tf.matmul(x, W) + b) # Softmax

# Minimize error using cross entropy
cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(activation), reduction_indices=1)) # Cross entropy
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost) # Gradient Descent

# Initializing the variables
init = tf.initialize_all_variables()

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

    # Training cycle
    for epoch in range(training_epochs):
        avg_cost = 0.
        total_batch = int(mnist.train.num_examples/batch_size)
        # Loop over all batches
        for i in range(total_batch):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            # Fit training using batch data
            sess.run(optimizer, feed_dict={x: batch_xs, y: batch_ys})
            # Compute average loss
            avg_cost += sess.run(cost, feed_dict={x: batch_xs, y: batch_ys})/total_batch
        # Display logs per epoch step
        if epoch % display_step == 0:
            print "Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}".format(avg_cost)

    print "Optimization Finished!"

    # Test model
    correct_prediction = tf.equal(tf.argmax(activation, 1), tf.argmax(y, 1))
    # Calculate accuracy
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
    print "Accuracy:", accuracy.eval({x: mnist.test.images, y: mnist.test.labels})
```

softmax classifier

Epoch: 0001 cost= 1.174406660
Epoch: 0002 cost= 0.661967539
Epoch: 0003 cost= 0.550489192
Epoch: 0004 cost= 0.496657414
Epoch: 0005 cost= 0.463665792
Epoch: 0006 cost= 0.440912077
Epoch: 0007 cost= 0.423909424
Epoch: 0008 cost= 0.410630655
Epoch: 0009 cost= 0.399893884
Epoch: 0010 cost= 0.390907963
Epoch: 0011 cost= 0.383317497
Epoch: 0012 cost= 0.376792131
Epoch: 0013 cost= 0.371025368
Epoch: 0014 cost= 0.365951805
Epoch: 0015 cost= 0.361361689
Epoch: 0016 cost= 0.357238019
Epoch: 0017 cost= 0.353540161
Epoch: 0018 cost= 0.350144092
Epoch: 0019 cost= 0.347053342
Epoch: 0020 cost= 0.344076798
Epoch: 0021 cost= 0.341447881
Epoch: 0022 cost= 0.339008725
Epoch: 0023 cost= 0.336701365
Epoch: 0024 cost= 0.334450486
Epoch: 0025 cost= 0.332461696
Optimization Finished!

Accuracy: 0.9139

Neural Nets (NN) for MNIST

```
# Parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100
display_step = 1

# tf Graph input
X = tf.placeholder("float", [None, 784]) # MNIST data input (img shape: 28*28)
Y = tf.placeholder("float", [None, 10]) # MNIST total classes (0-9 digits)

# Store layers weight & bias
W1 = tf.Variable(tf.random_normal([784, 256]))
W2 = tf.Variable(tf.random_normal([256, 256]))
W3 = tf.Variable(tf.random_normal([256, 10]))

B1 = tf.Variable(tf.random_normal([256]))
B2 = tf.Variable(tf.random_normal([256]))
B3 = tf.Variable(tf.random_normal([10]))

# Construct model
L1 = tf.nn.relu(tf.add(tf.matmul(X, W1), B1))
L2 = tf.nn.relu(tf.add(tf.matmul(L1, W2), B2)) #Hidden layer with RELU activation
hypothesis = tf.add(tf.matmul(L2, W3), B3) # No need to use softmax here

# Define loss and optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(hypothesis, Y)) # Softmax loss
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost) # Adam Optimizer
```

Neural Nets (NN) for MNIST

```
# Initializing the variables
init = tf.initialize_all_variables()

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

    # Training cycle
    for epoch in range(training_epochs):
        avg_cost = 0.
        total_batch = int(mnist.train.num_examples/batch_size)
        # Loop over all batches
        for i in range(total_batch):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            # Fit training using batch data
            sess.run(optimizer, feed_dict={X: batch_xs, Y: batch_ys})
            # Compute average loss
            avg_cost += sess.run(cost, feed_dict={X: batch_xs, Y: batch_ys})/total_batch
        # Display logs per epoch step
        if epoch % display_step == 0:
            print "Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}".format(avg_cost)

print "Optimization Finished!"

# Test model
correct_prediction = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
# Calculate accuracy
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
print "Accuracy:", accuracy.eval({X: mnist.test.images, Y: mnist.test.labels})
```

softmax classifier

Epoch: 0001 cost= 1.174406660
Epoch: 0002 cost= 0.661967539
Epoch: 0003 cost= 0.550489192
Epoch: 0004 cost= 0.496657414
Epoch: 0005 cost= 0.463665792
Epoch: 0006 cost= 0.440912077
Epoch: 0007 cost= 0.423909424
Epoch: 0008 cost= 0.410630655
Epoch: 0009 cost= 0.399893884
Epoch: 0010 cost= 0.390907963
Epoch: 0011 cost= 0.383317497
Epoch: 0012 cost= 0.376792131
Epoch: 0013 cost= 0.371025368
Epoch: 0014 cost= 0.365951805
Epoch: 0015 cost= 0.361361689
Epoch: 0016 cost= 0.357238019
Epoch: 0017 cost= 0.353540161
Epoch: 0018 cost= 0.350144092
Epoch: 0019 cost= 0.347053342
Epoch: 0020 cost= 0.344076798
Epoch: 0021 cost= 0.341447881
Epoch: 0022 cost= 0.339008725
Epoch: 0023 cost= 0.336701365
Epoch: 0024 cost= 0.334450486
Epoch: 0025 cost= 0.332461696
Optimization Finished!

Accuracy: 0.9139

NN

Epoch: 0001 cost= 153.374492868
Epoch: 0002 cost= 41.126819546
Epoch: 0003 cost= 25.309642092
Epoch: 0004 cost= 17.206465834
Epoch: 0005 cost= 12.155490249
Epoch: 0006 cost= 8.755095852
Epoch: 0007 cost= 6.392030562
Epoch: 0008 cost= 4.629136964
Epoch: 0009 cost= 3.347306573
Epoch: 0010 cost= 2.372126589
Epoch: 0011 cost= 1.667233310
Epoch: 0012 cost= 1.202339336
Epoch: 0013 cost= 0.837206638
Epoch: 0014 cost= 0.593220934
Epoch: 0015 cost= 0.431912481
Optimization Finished!

Accuracy: 0.9446

Xavier initialization

```
def xavier_init(n_inputs, n_outputs, uniform=True):
    """Set the parameter initialization using the method described.
    This method is designed to keep the scale of the gradients roughly the same
    in all layers.
    Xavier Glorot and Yoshua Bengio (2010):
        Understanding the difficulty of training deep feedforward neural
        networks. International conference on artificial intelligence and
        statistics.

    Args:
        n_inputs: The number of input nodes into each output.
        n_outputs: The number of output nodes for each input.
        uniform: If true use a uniform distribution, otherwise use a normal.

    Returns:
        An initializer.

    """
    if uniform:
        # 6 was used in the paper.
        init_range = tf.sqrt(6.0 / (n_inputs + n_outputs))
        return tf.random_uniform_initializer(-init_range, init_range)
    else:
        # 3 gives us approximately the same limits as above since this repicks
        # values greater than 2 standard deviations from the mean.
        stddev = tf.sqrt(3.0 / (n_inputs + n_outputs))
        return tf.truncated_normal_initializer(stddev=stddev)

# Store layers weight & bias
W1 = tf.get_variable("W1", shape=[784, 256], initializer=xavier_init(784,256))
W2 = tf.get_variable("W2", shape=[256, 256], initializer=xavier_init(256,256))
W3 = tf.get_variable("W3", shape=[256, 10], initializer=xavier_init(256,10))
```

NN

Epoch: 0001 cost= 153.374492868
Epoch: 0002 cost= 41.126819546
Epoch: 0003 cost= 25.309642092
Epoch: 0004 cost= 17.206465834
Epoch: 0005 cost= 12.155490249
Epoch: 0006 cost= 8.755095852
Epoch: 0007 cost= 6.392030562
Epoch: 0008 cost= 4.629136964
Epoch: 0009 cost= 3.347306573
Epoch: 0010 cost= 2.372126589
Epoch: 0011 cost= 1.667233310
Epoch: 0012 cost= 1.202339336
Epoch: 0013 cost= 0.837206638
Epoch: 0014 cost= 0.593220934
Epoch: 0015 cost= 0.431912481
Optimization Finished!

Accuracy: 0.9446

NN with xavier initialization

Epoch: 0001 cost= 0.330929694
Epoch: 0002 cost= 0.110038888
Epoch: 0003 cost= 0.067369296
Epoch: 0004 cost= 0.045064388
Epoch: 0005 cost= 0.031090851
Epoch: 0006 cost= 0.022001974
Epoch: 0007 cost= 0.016603567
Epoch: 0008 cost= 0.011094349
Epoch: 0009 cost= 0.008923969
Epoch: 0010 cost= 0.007312808
Epoch: 0011 cost= 0.006277084
Epoch: 0012 cost= 0.004857574
Epoch: 0013 cost= 0.004891470
Epoch: 0014 cost= 0.004491583
Epoch: 0015 cost= 0.003429245
Optimization Finished!

Accuracy: 0.9779

More deep & dropout

```
# Construct model
dropout_rate = tf.placeholder("float")
_L1 = tf.nn.relu(tf.add(tf.matmul(X, W1), B1)) #Hidden layer with RELU activation
L1 = tf.nn.dropout(_L1, dropout_rate)
_L2 = tf.nn.relu(tf.add(tf.matmul(L1, W2), B2)) #Hidden layer with RELU activation
L2 = tf.nn.dropout(_L2, dropout_rate)
_L3 = tf.nn.relu(tf.add(tf.matmul(L2, W3), B3)) #Hidden layer with RELU activation
L3 = tf.nn.dropout(_L3, dropout_rate)
_L4 = tf.nn.relu(tf.add(tf.matmul(L3, W4), B4)) #Hidden layer with RELU activation
L4 = tf.nn.dropout(_L4, dropout_rate)

hypothesis = tf.add(tf.matmul(L4, W5), B5) # No need to use softmax here

for i in range(total_batch):
    batch_xs, batch_ys = mnist.train.next_batch(batch_size)
    # Fit training using batch data
    sess.run(optimizer, feed_dict={X: batch_xs, Y: batch_ys, dropout_rate: 0.7})

# Test model
correct_prediction = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
# Calculate accuracy
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
print "Accuracy:", accuracy.eval({X: mnist.test.images, Y: mnist.test.labels, dropout_rate: 1})
```

NN deep + dropout

```
Epoch: 0001 cost= 0.584449715
Epoch: 0002 cost= 0.215399251
Epoch: 0003 cost= 0.160561109
Epoch: 0004 cost= 0.132314345
Epoch: 0005 cost= 0.114490116
Epoch: 0006 cost= 0.103506013
Epoch: 0007 cost= 0.095571726
Epoch: 0008 cost= 0.084172901
Epoch: 0009 cost= 0.079563179
Epoch: 0010 cost= 0.073859323
Epoch: 0011 cost= 0.071492671
Epoch: 0012 cost= 0.066446339
Epoch: 0013 cost= 0.061337474
Epoch: 0014 cost= 0.058591939
Epoch: 0015 cost= 0.055077895
Optimization Finished!
```

Accuracy: 0.9803

NN with xavier initialization

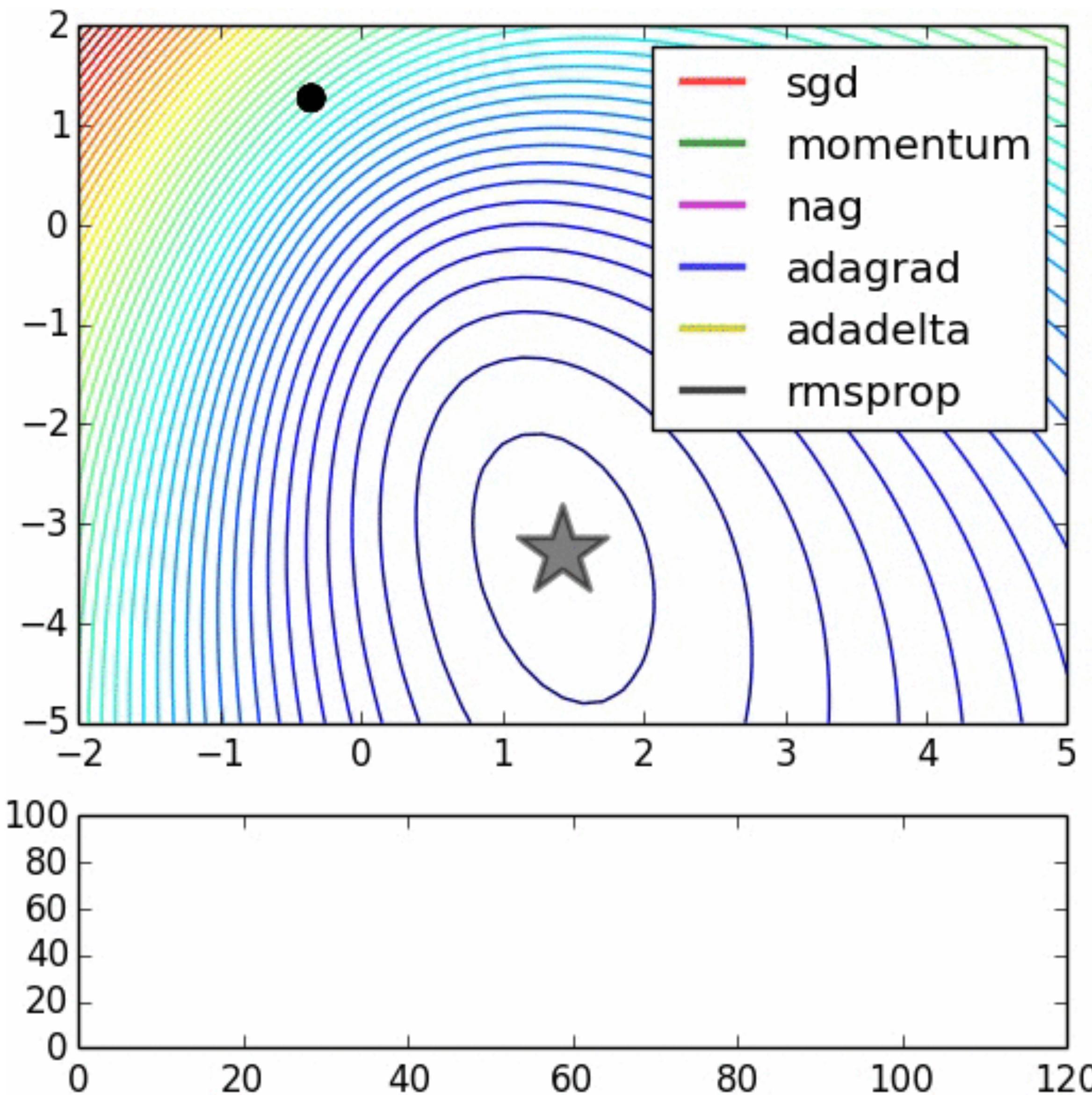
```
Epoch: 0001 cost= 0.330929694
Epoch: 0002 cost= 0.110038888
Epoch: 0003 cost= 0.067369296
Epoch: 0004 cost= 0.045064388
Epoch: 0005 cost= 0.031090851
Epoch: 0006 cost= 0.022001974
Epoch: 0007 cost= 0.016603567
Epoch: 0008 cost= 0.011094349
Epoch: 0009 cost= 0.008923969
Epoch: 0010 cost= 0.007312808
Epoch: 0011 cost= 0.006277084
Epoch: 0012 cost= 0.004857574
Epoch: 0013 cost= 0.004891470
Epoch: 0014 cost= 0.004491583
Epoch: 0015 cost= 0.003429245
Optimization Finished!
```

Accuracy: 0.9779

Optimizer

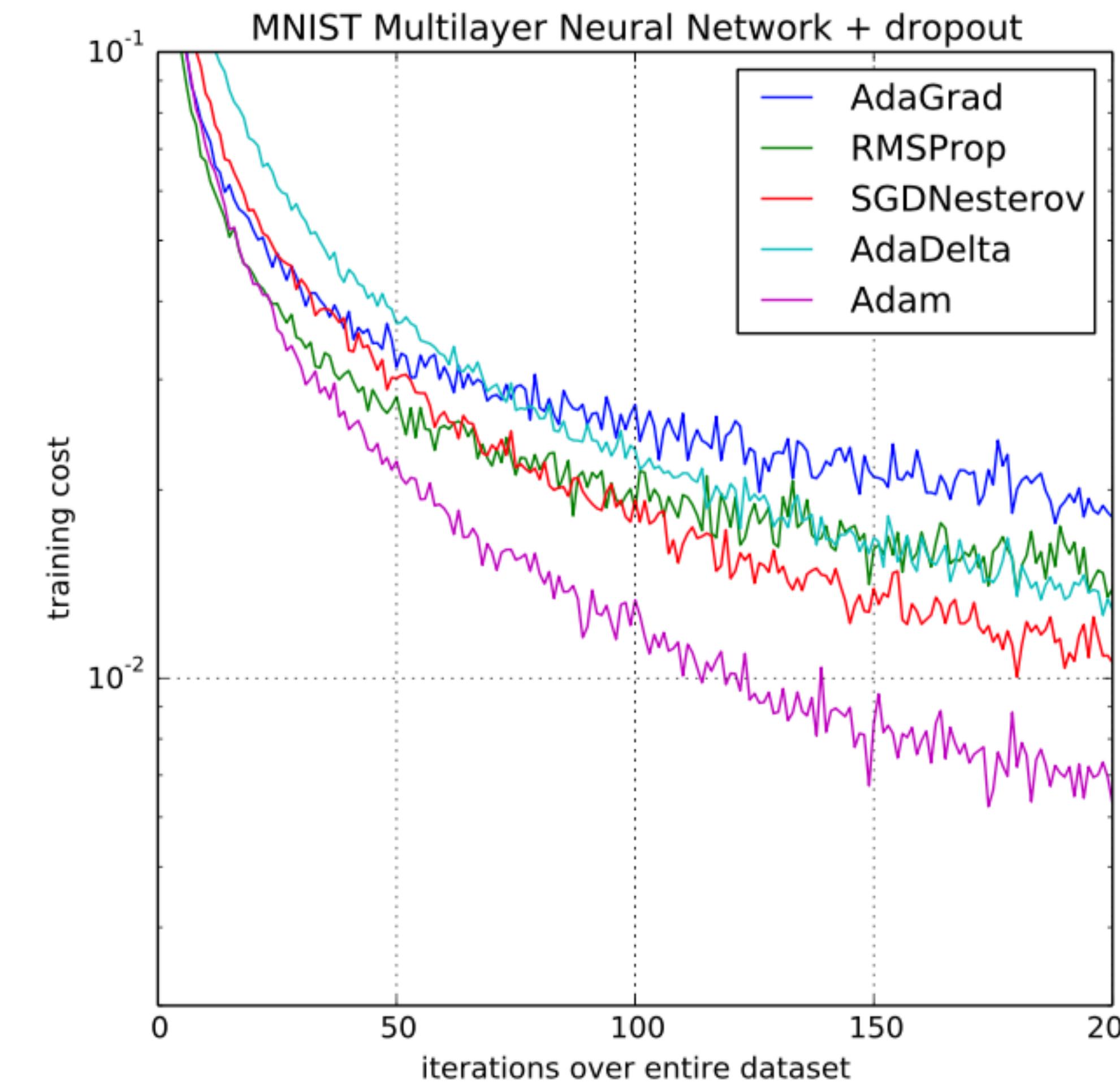
```
# Construct model
activation = tf.nn.softmax(tf.matmul(x, W) + b) # Softmax

# Minimize error using cross entropy
cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(activation), reduction_indices=1)) # Cross entropy
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost) # Gradient Descent
```



ADAM: a method for stochastic optimization

[Kingma et al. 2015]



Use Adam Optimizer

```
# Construct model
activation = tf.nn.softmax(tf.matmul(x, W) + b) # Softmax

# Minimize error using cross entropy
cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(activation), reduction_indices=1)) # Cross entropy
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost) # Gradient Descent

hypothesis = tf.add(tf.matmul(L4, W5), B5) # No need to use softmax here

# Define loss and optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(hypothesis, Y)) # Softmax loss
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost) # Adam Optimizer
```

Lab summary

- Softmax VS Neural Nets for MNIST, 91.4% and 94.4%
- Xavier initialization: 97.8%
- Deep Neural Nets and Dropout: 98%
- Adam optimizer

Lab 11

CNN

Sung Kim <hunkim+ml@gmail.com>
<http://hunkim.github.io/ml/>

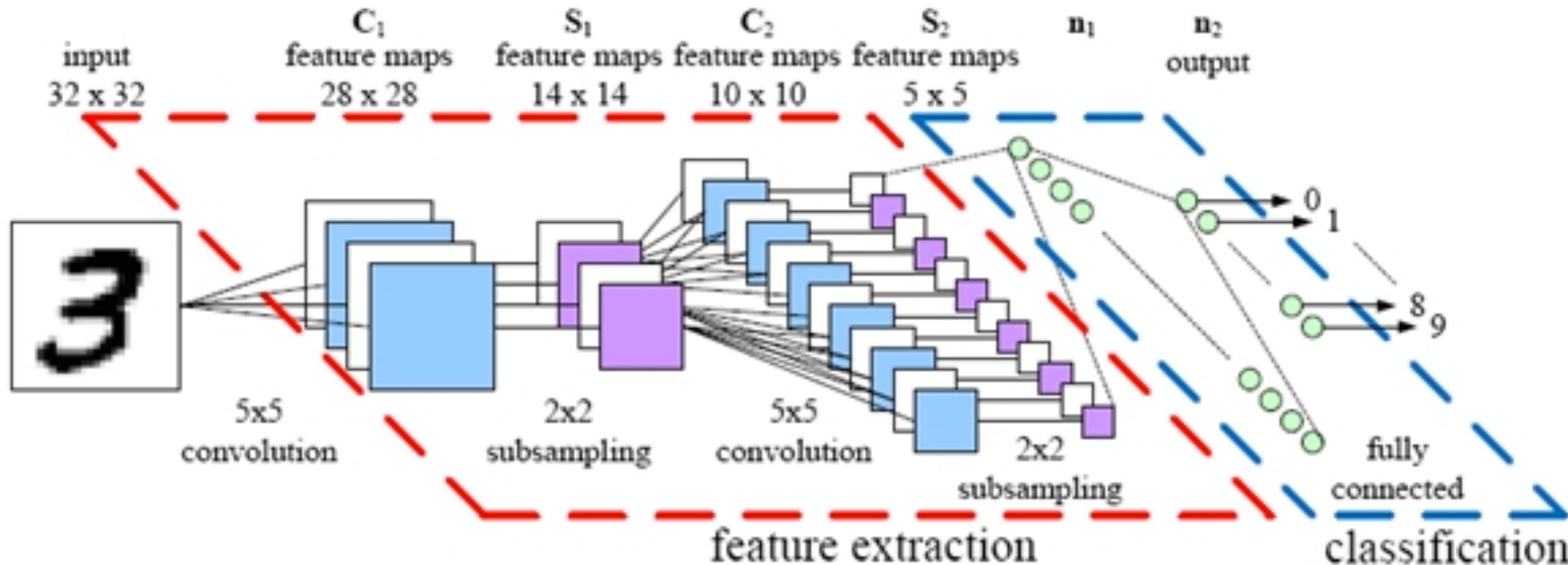
TensorFlow-Tutorials

Introduction to deep learning based on Google's TensorFlow framework. These tutorials are direct ports of Newmu's [Theano Tutorials](#)

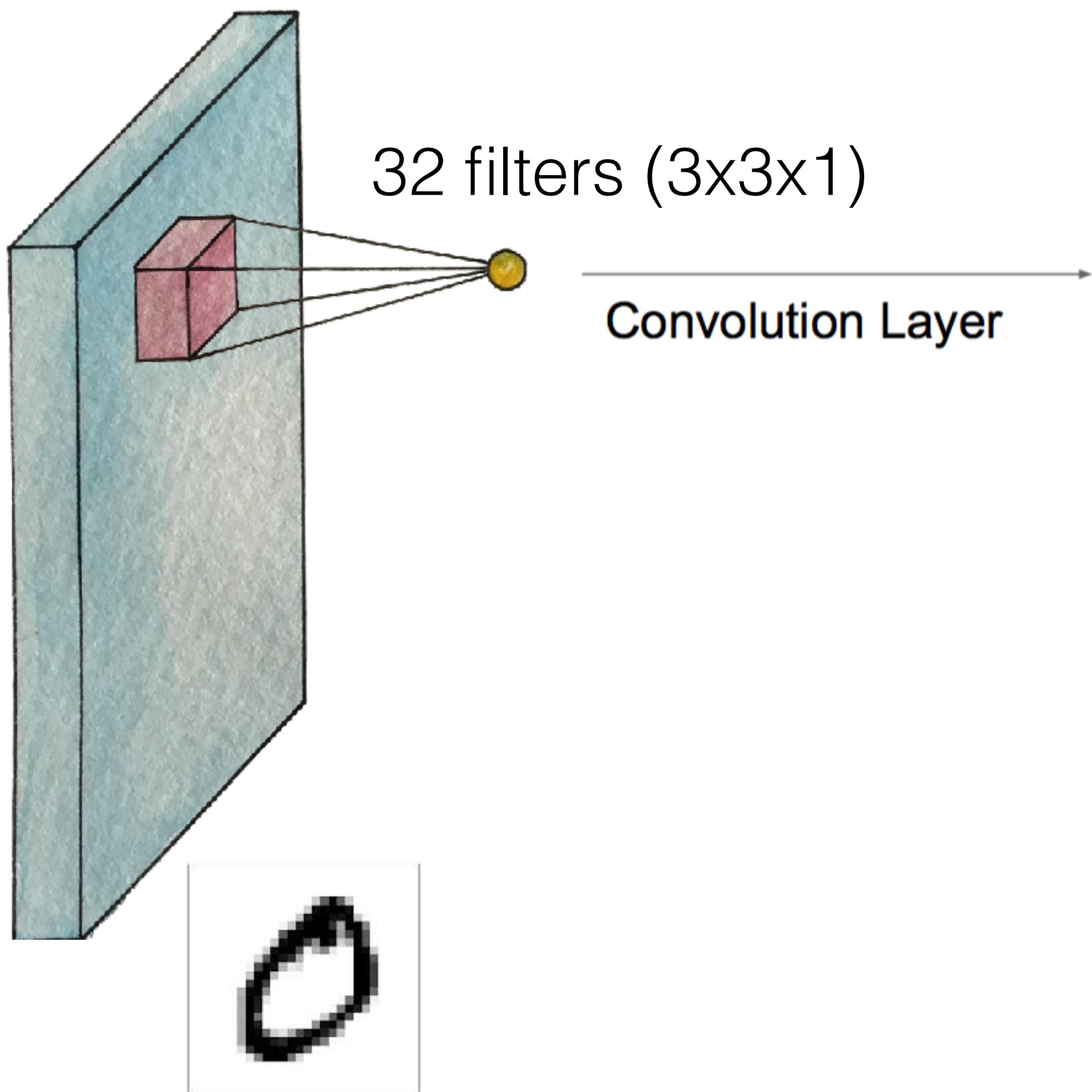
Topics

- Simple Multiplication
- Linear Regression
- Logistic Regression
- Feedforward Neural Network (Multilayer Perceptron)
- Deep Feedforward Neural Network (Multilayer Perceptron with 2 Hidden Layers O.o)
- Convolutional Neural Network
- Denoising Autoencoder
- LSTM

CNN

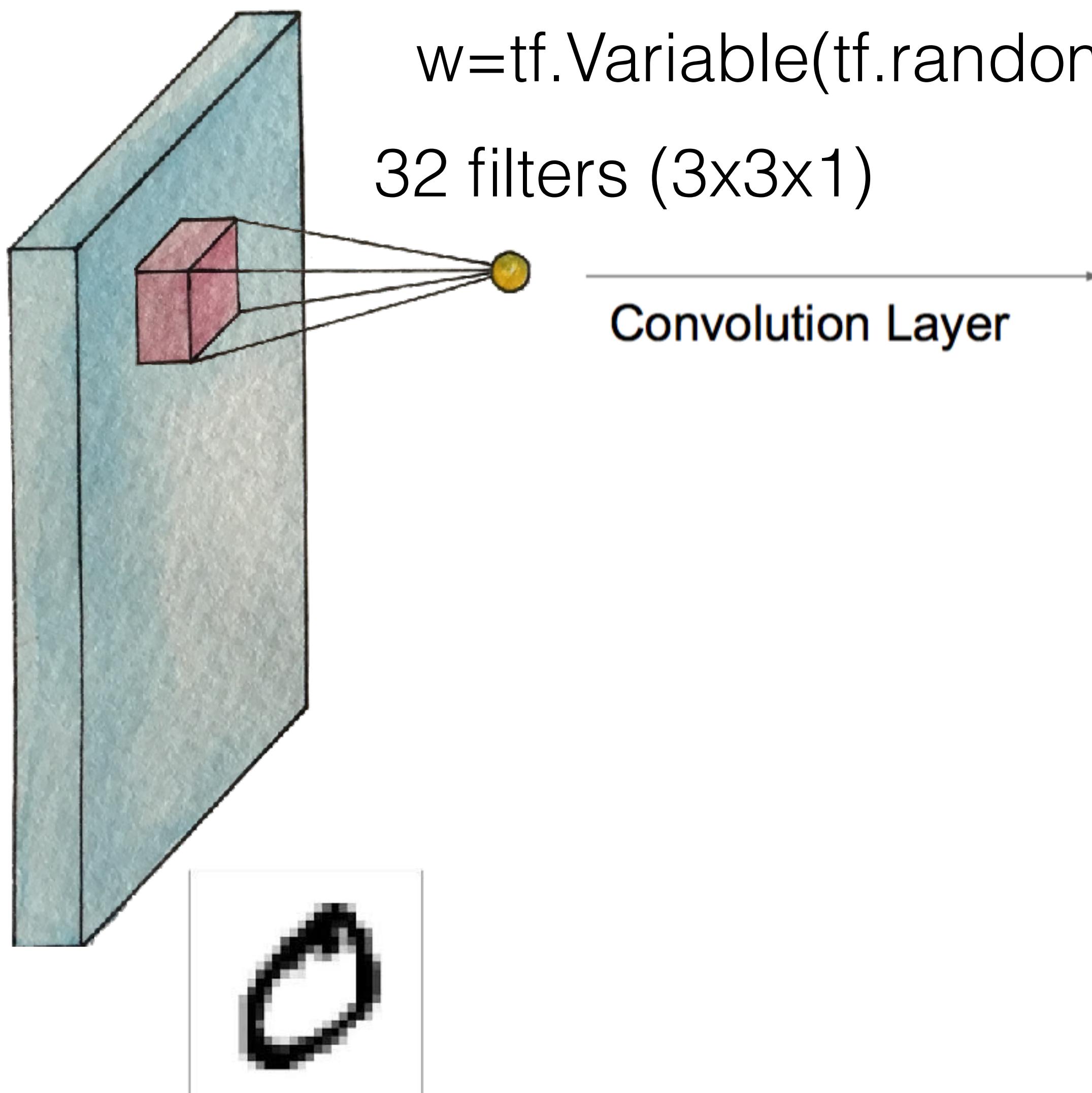


Convolutional layers



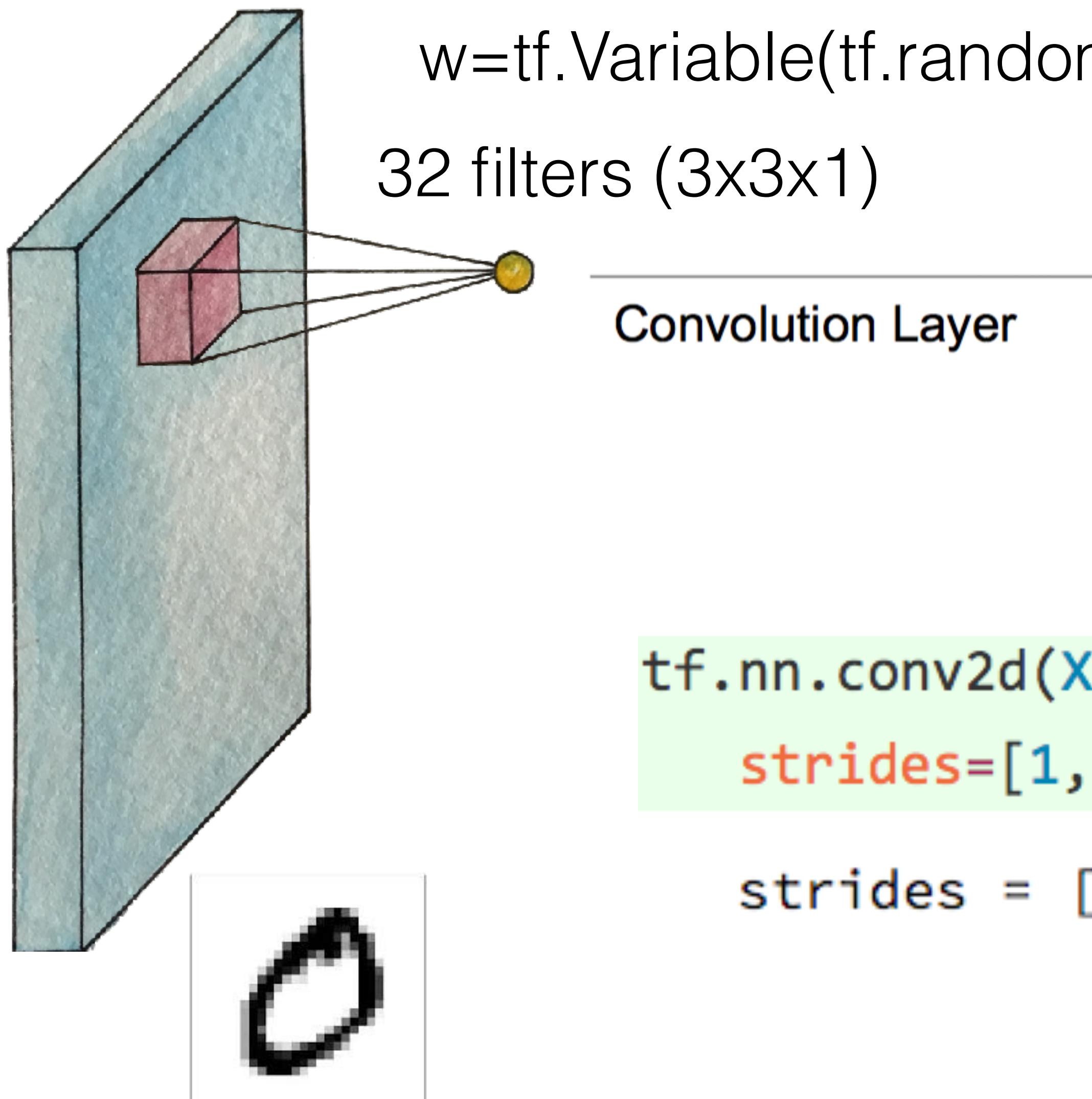
MNIST 28x28x1 image

Convolutional layers



28x28x1 image

Convolutional layers



w=tf.Variable(tf.random_normal([3,3,1,32], stddev=0.01))

32 filters ($3 \times 3 \times 1$)

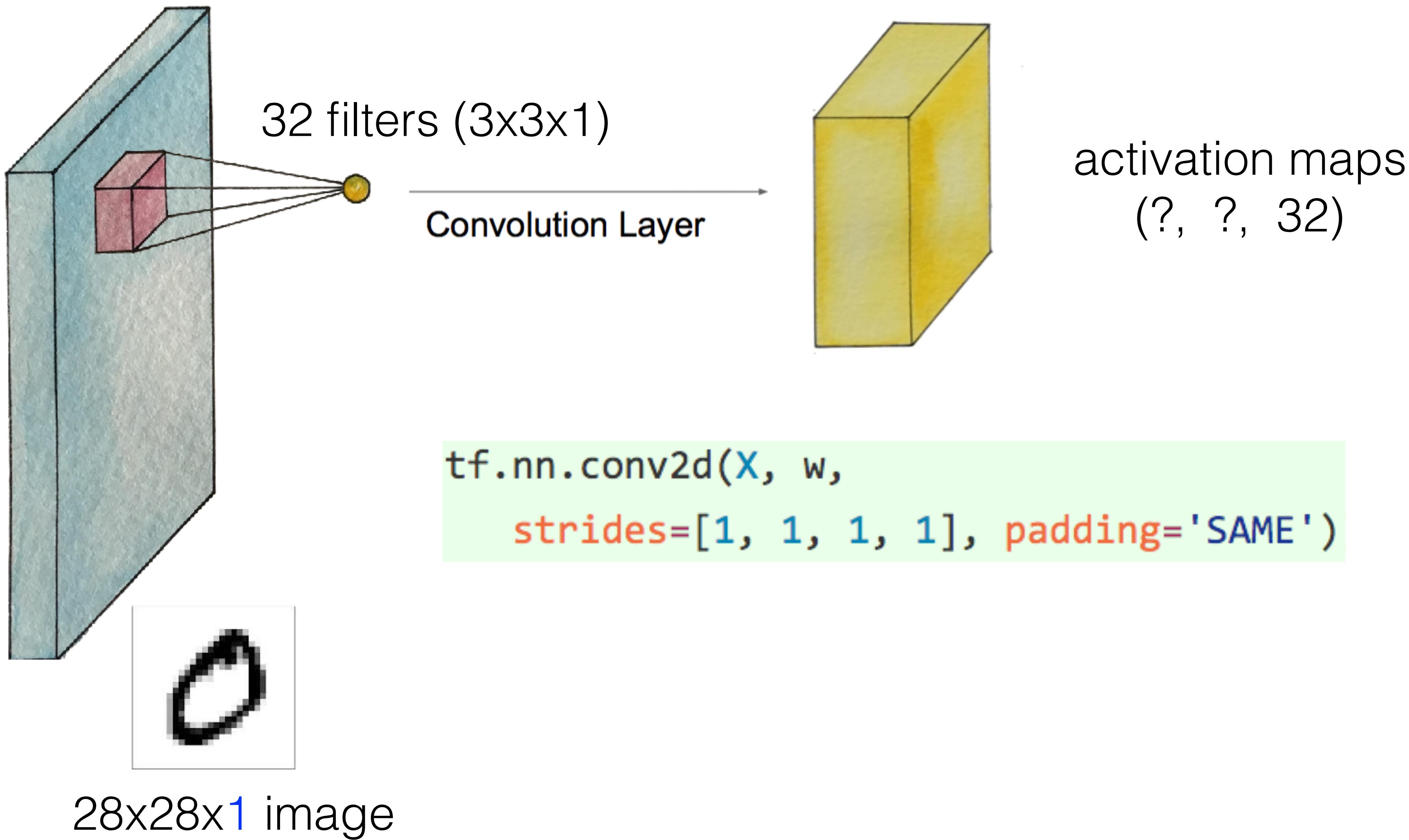
Convolution Layer

```
tf.nn.conv2d(X, w,  
            strides=[1, 1, 1, 1], padding='SAME')
```

strides = [1, stride, stride, 1].

$28 \times 28 \times 1$ image

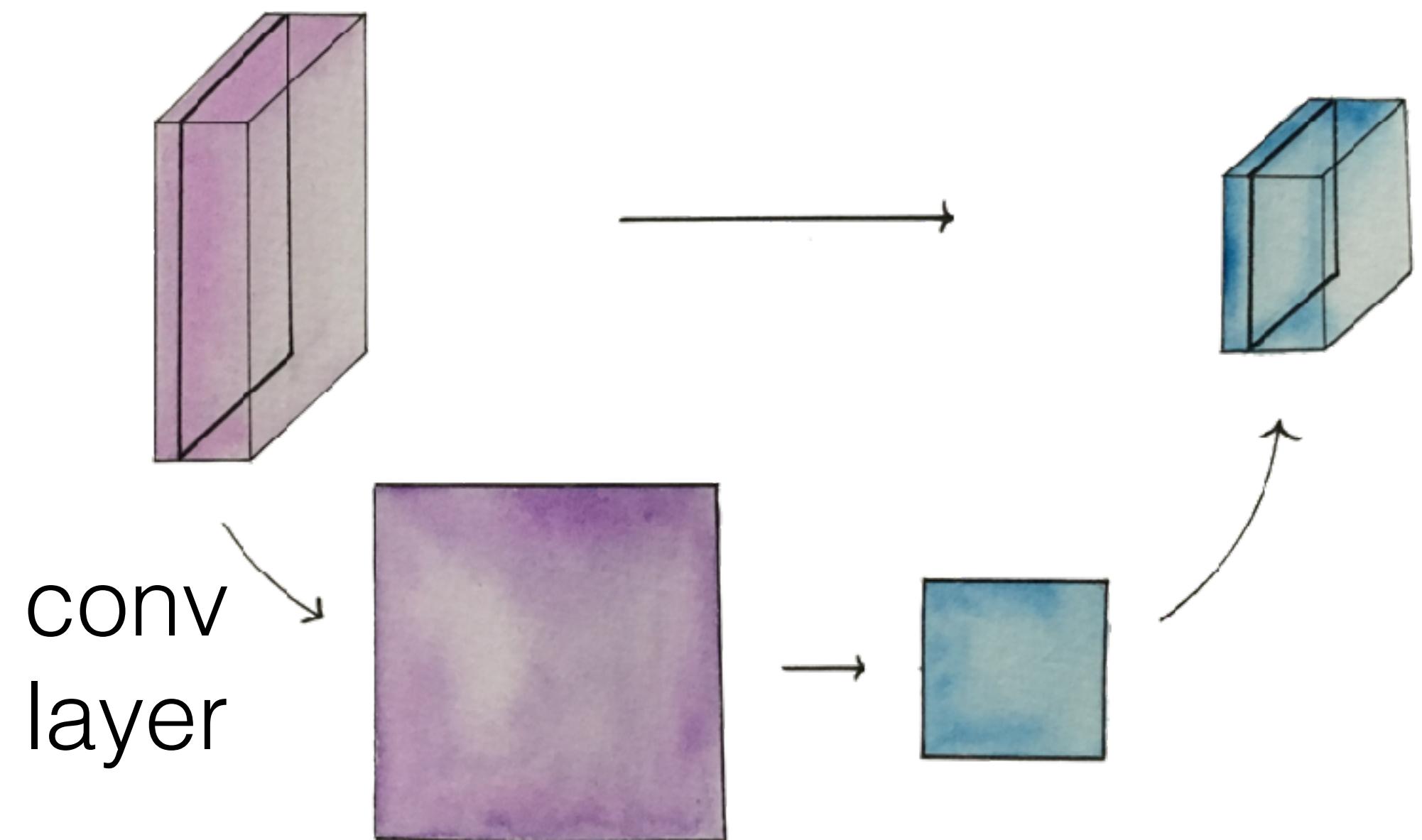
Convolutional layers



ReLU

```
l = tf.nn.conv2d(X, w, [1, 1, 1, 1], 'SAME')  
l1a = tf.nn.relu(tf.nn.conv2d(X, w, [1, 1, 1, 1], 'SAME'))
```

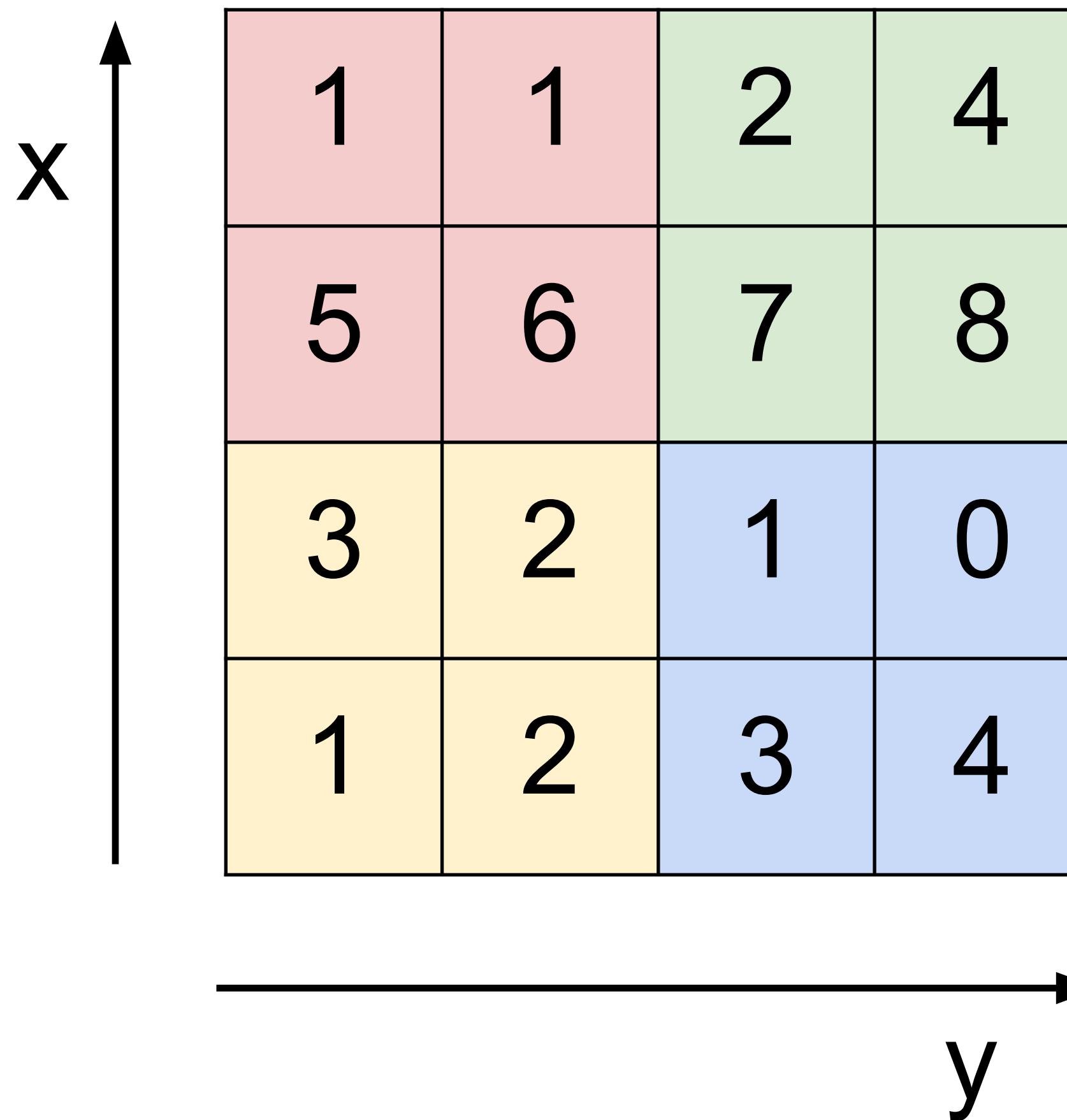
Pooling layer (sampling)



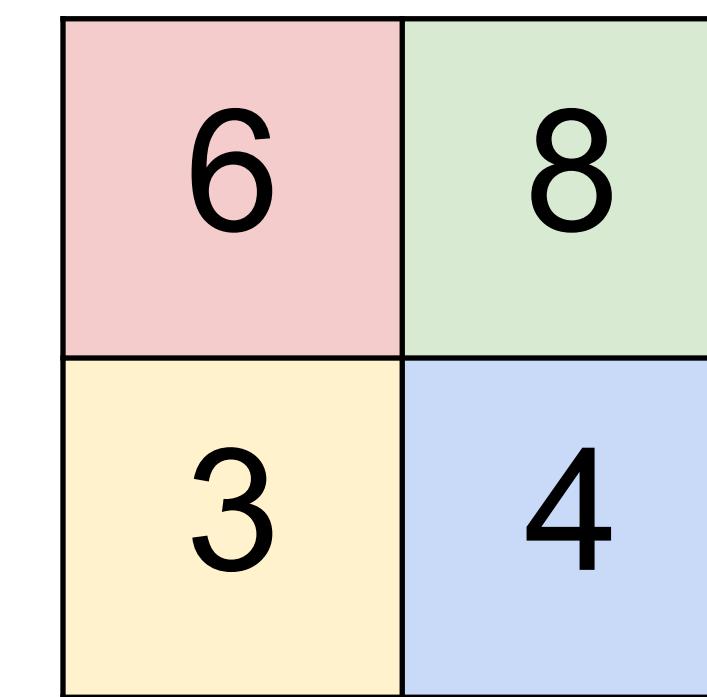
```
l1 = tf.nn.max_pool(c1, ksize=[1, 2, 2, 1],
                    strides=[1, 2, 2, 1], padding='SAME')
```

MAX POOLING

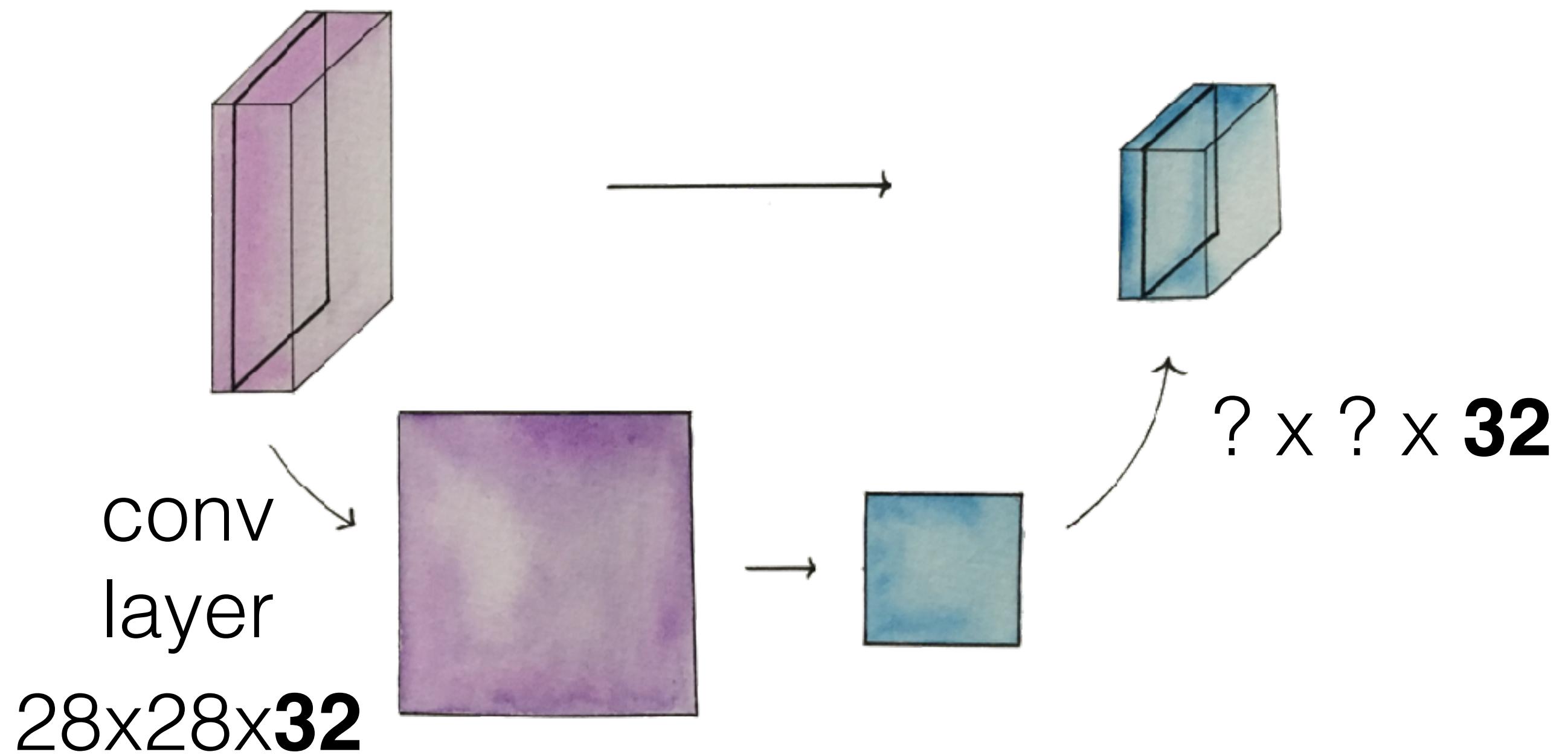
Single depth slice



max pool with 2x2 filters
and stride 2



Pooling layer (sampling)



Shape not sure? Print tensor

```
l1a = tf.nn.relu(tf.nn.conv2d(X, w, [1, 1, 1, 1], 'SAME'))  
print l1a
```

Tensor("Conv2D:0", shape=(?, 28, 28, 32), dtype=float32)

```
l1 = tf.nn.max_pool(l1a, ksize=[1, 2, 2, 1],  
|                      strides=[1, 2, 2, 1], padding='SAME')  
print l1
```

Tensor("MaxPool:0", shape=(?, 14, 14, 32), dtype=float32)

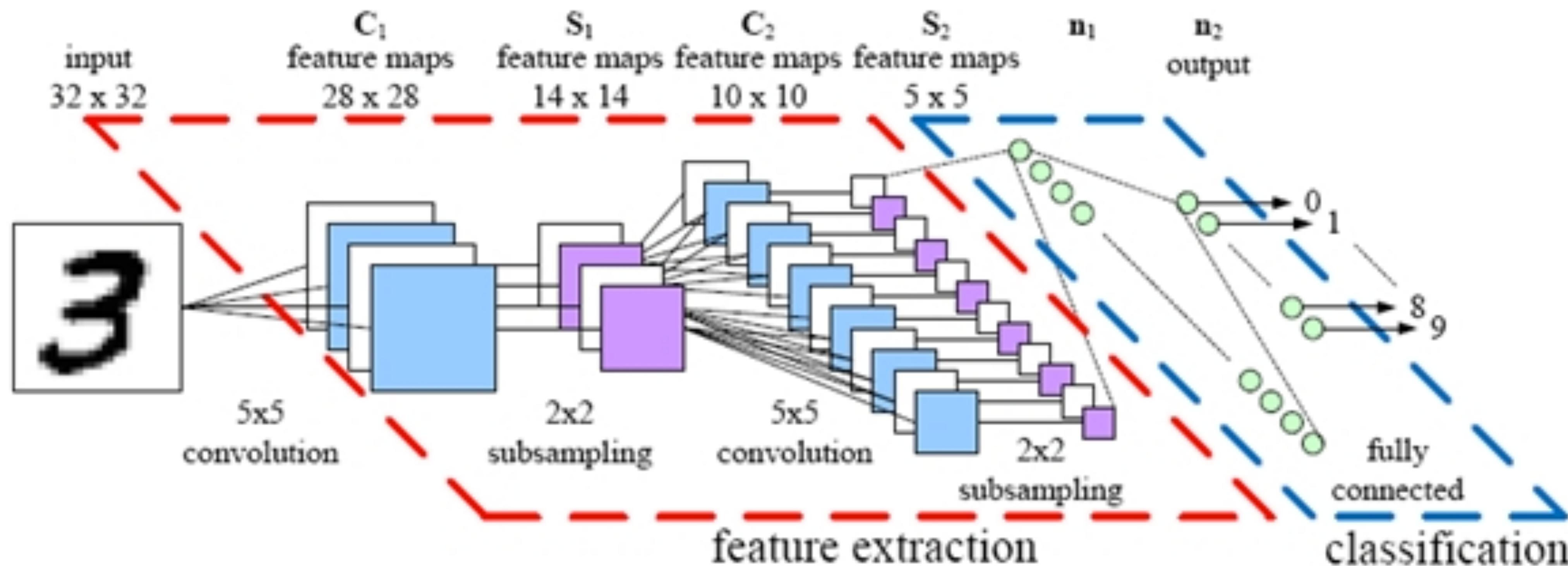
```
X = trX.reshape(-1, 28, 28, 1) w = init_weights([3, 3, 1, 32]) # 3x3x1 conv, 32 outputs  
w2 = init_weights([3, 3, 32, 64]) # 3x3x32 conv, 64 outputs  
w3 = init_weights([3, 3, 64, 128]) # 3x3x32 conv, 128 outputs  
  
l1a = tf.nn.relu(tf.nn.conv2d(X, w, # l1a shape=(?, 28, 28, 32)  
    strides=[1, 1, 1, 1], padding='SAME'))  
l1 = tf.nn.max_pool(l1a, ksize=[1, 2, 2, 1], # l1 shape=(?, 14, 14, 32)  
    strides=[1, 2, 2, 1], padding='SAME')  
  
l2a = tf.nn.relu(tf.nn.conv2d(l1, w2, # l2a shape=(?, 14, 14, 64)  
    strides=[1, 1, 1, 1], padding='SAME'))  
l2 = tf.nn.max_pool(l2a, ksize=[1, 2, 2, 1], # l2 shape=(?, 7, 7, 64)  
    strides=[1, 2, 2, 1], padding='SAME')  
  
l3a = tf.nn.relu(tf.nn.conv2d(l2, w3, # l3a shape=(?, 7, 7, 128)  
    strides=[1, 1, 1, 1], padding='SAME'))  
l3 = tf.nn.max_pool(l3a, ksize=[1, 2, 2, 1], # l3 shape=(?, 4, 4, 128)  
    strides=[1, 2, 2, 1], padding='SAME')
```

dropout

```
X = trX.reshape(-1, 28, 28, 1)
```

```
l1a = tf.nn.relu(tf.nn.conv2d(X, w,  
                           strides=[1, 1, 1, 1], padding='SAME')) # l1a shape=(?, 28, 28, 32)  
l1 = tf.nn.max_pool(l1a, ksize=[1, 2, 2, 1],  
                     strides=[1, 2, 2, 1], padding='SAME') # l1 shape=(?, 14, 14, 32)  
l1 = tf.nn.dropout(l1, p_keep_conv)  
  
l2a = tf.nn.relu(tf.nn.conv2d(l1, w2,  
                           strides=[1, 1, 1, 1], padding='SAME')) # l2a shape=(?, 14, 14, 64)  
l2 = tf.nn.max_pool(l2a, ksize=[1, 2, 2, 1],  
                     strides=[1, 2, 2, 1], padding='SAME') # l2 shape=(?, 7, 7, 64)  
l2 = tf.nn.dropout(l2, p_keep_conv)  
  
l3a = tf.nn.relu(tf.nn.conv2d(l2, w3,  
                           strides=[1, 1, 1, 1], padding='SAME')) # l3a shape=(?, 7, 7, 128)  
l3 = tf.nn.max_pool(l3a, ksize=[1, 2, 2, 1],  
                     strides=[1, 2, 2, 1], padding='SAME') # l3 shape=(?, 4, 4, 128)
```

CNN



Fully connected net

```
l1a = tf.nn.relu(tf.nn.conv2d(X, w,  
                           strides=[1, 1, 1, 1], padding='SAME')) # l1a shape=(?, 28, 28, 32)  
l1 = tf.nn.max_pool(l1a, ksize=[1, 2, 2, 1],  
                     strides=[1, 2, 2, 1], padding='SAME') # l1 shape=(?, 14, 14, 32)  
w4 = init_weights([128 * 4 * 4, 625]) # FC 128 * 4 * 4 inputs, 625 outputs  
w_o = init_weights([625, 10]) # FC 625 inputs, 10 outputs (labels)  
                           strides=[1, 1, 1, 1], padding='SAME'))  
l2 = tf.nn.max_pool(l2a, ksize=[1, 2, 2, 1],  
                     strides=[1, 2, 2, 1], padding='SAME') # l2 shape=(?, 7, 7, 64)  
l2 = tf.nn.dropout(l2, p_keep_conv)  
  
l3a = tf.nn.relu(tf.nn.conv2d(l2, w3,  
                           strides=[1, 1, 1, 1], padding='SAME')) # l3a shape=(?, 7, 7, 128)  
l3 = tf.nn.max_pool(l3a, ksize=[1, 2, 2, 1],  
                     strides=[1, 2, 2, 1], padding='SAME') # l3 shape=(?, 4, 4, 128)  
l3 = tf.reshape(l3, [-1, w4.get_shape().as_list()[0]]) # reshape to (?, 2048)  
l3 = tf.nn.dropout(l3, p_keep_conv)  
  
l4 = tf.nn.relu(tf.matmul(l3, w4))  
l4 = tf.nn.dropout(l4, p_keep_hidden)  
  
pyx = tf.matmul(l4, w_o)
```

cost and optimization

```
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(py_x, Y))
train_op = tf.train.RMSPropOptimizer(0.001, 0.9).minimize(cost)
predict_op = tf.argmax(py_x, 1)
```

```
tf.train.RMSPropOptimizer.__init__(learning_rate, decay=0.9,
momentum=0.0, epsilon=1e-10, use_locking=False,
name='RMSProp')
```

Construct a new RMSProp optimizer.

Args:

- `learning_rate`: A Tensor or a floating point value. The learning rate.
- `decay`: Discounting factor for the history/coming gradient

Other TF optimizers

```
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(py_x, Y))
train_op = tf.train.RMSPropOptimizer(0.001, 0.9).minimize(cost)
predict_op = tf.argmax(py_x, 1)
```

- `class tf.train.GradientDescentOptimizer`
- `class tf.train.AdadeltaOptimizer`
- `class tf.train.AdagradOptimizer`
- `class tf.train.MomentumOptimizer`
- `class tf.train.AdamOptimizer`
- `class tf.train.FtrlOptimizer`
- `class tf.train.RMSPropOptimizer`

Train and testing

Train and testing

```
# Launch the graph in a session
with tf.Session() as sess:
    # you need to initialize all variables
    tf.initialize_all_variables().run()
```

0	128
128	256
256	384
384	512

Train and testing

```
# Launch the graph in a session
with tf.Session() as sess:
    # you need to initialize all variables
    tf.initialize_all_variables().run()

    for i in range(100):
        for start, end in zip(range(0, len(trX), 128), range(128, len(trX))):
            sess.run(train_op, feed_dict={X: trX[start:end], Y: trY[start:end],
                                         p_keep_conv: 0.8, p_keep_hidden: 0.5})

    test_indices = np.arange(len(teX)) # Get A Test Batch
    np.random.shuffle(test_indices)
    test_indices = test_indices[0:256]

    print i, np.mean(np.argmax(teY[test_indices], axis=1) ==
                     sess.run(predict_op, feed_dict={X: teX[test_indices],
                                                     Y: teY[test_indices],
                                                     p_keep_conv: 1.0,
                                                     p_keep_hidden: 1.0}))
```

```
0 0.98046875
1 0.97265625
2 0.984375
3 0.9765625
4 0.9921875
5 0.9921875
6 0.9921875
```

<https://github.com/nlintz/TensorFlow-Tutorials>

Lab 12

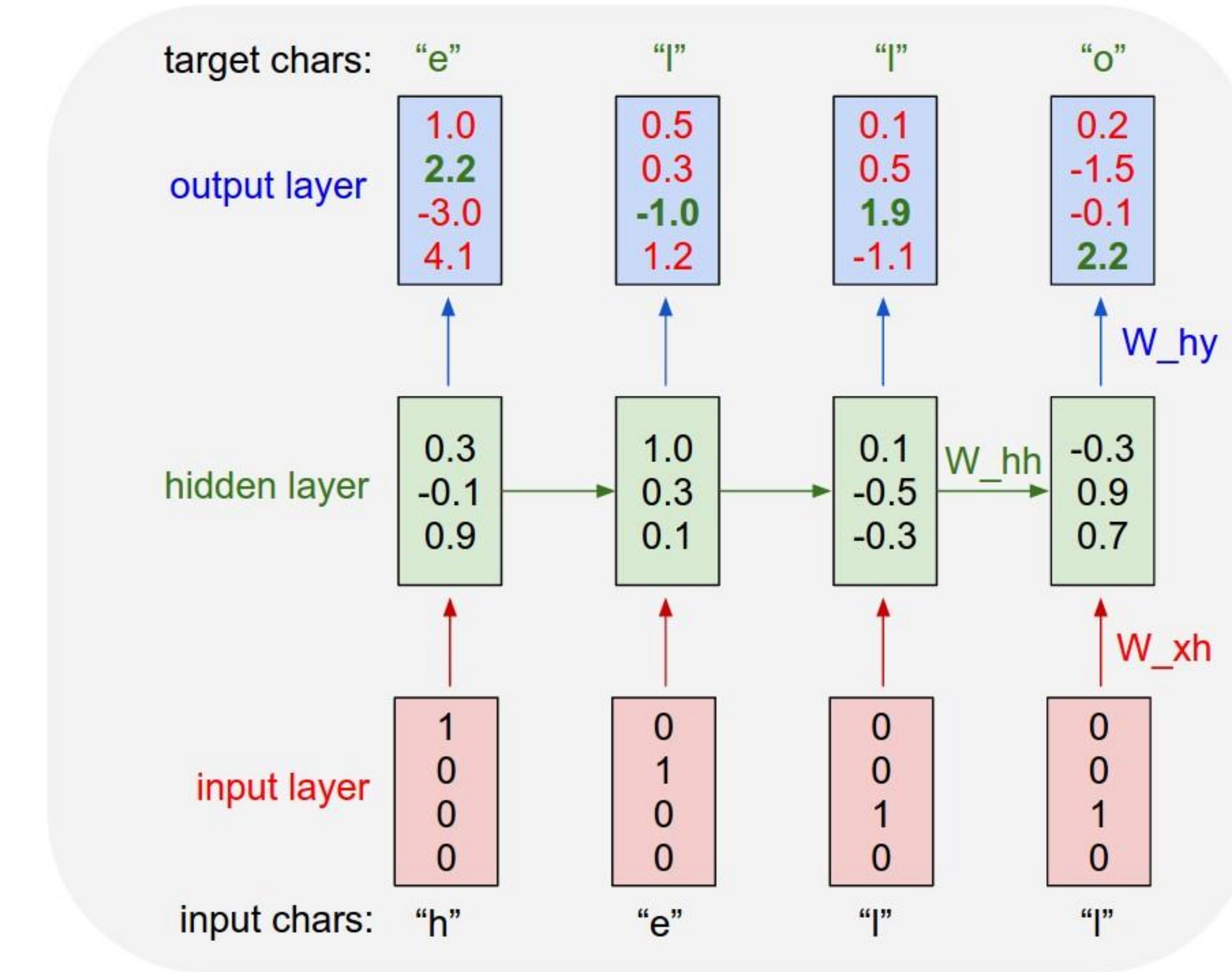
RNN

Sung Kim <hunkim+ml@gmail.com>
<http://hunkim.github.io/ml/>

Character-level language model example

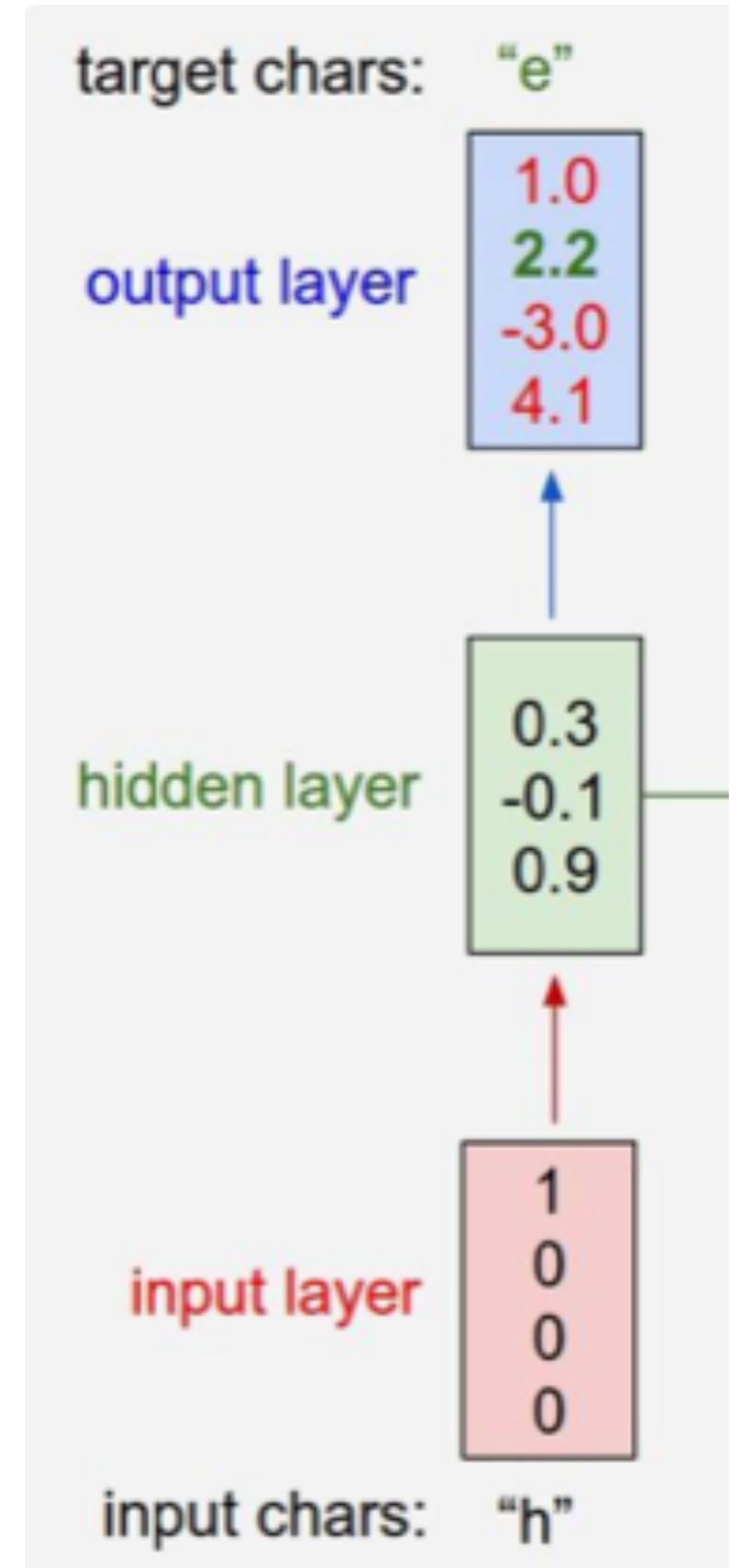
Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”



Creating rnn cell

```
# RNN model  
rnn_cell = rnn_cell.BasicRNNCell(rnn_size)
```



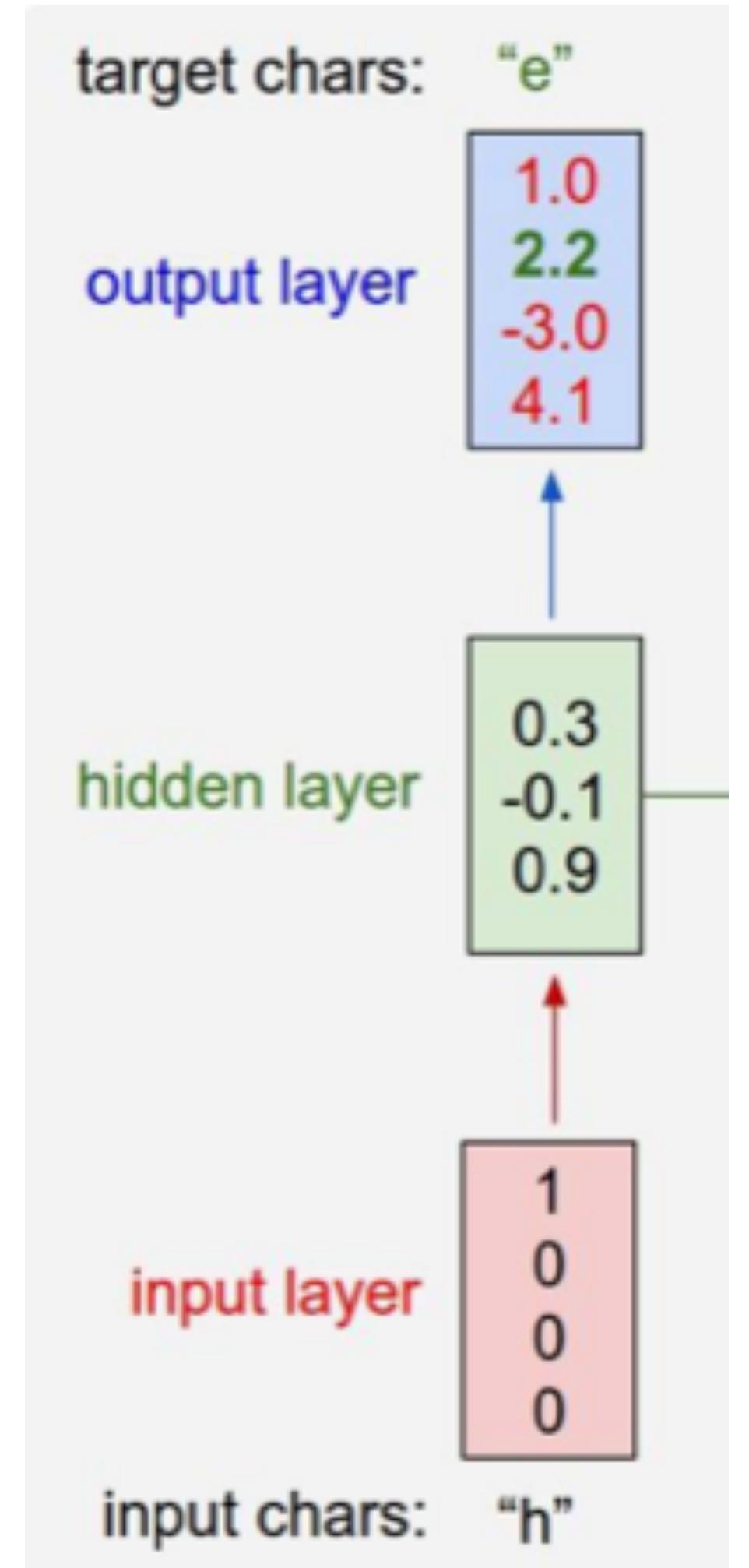
Creating rnn cell

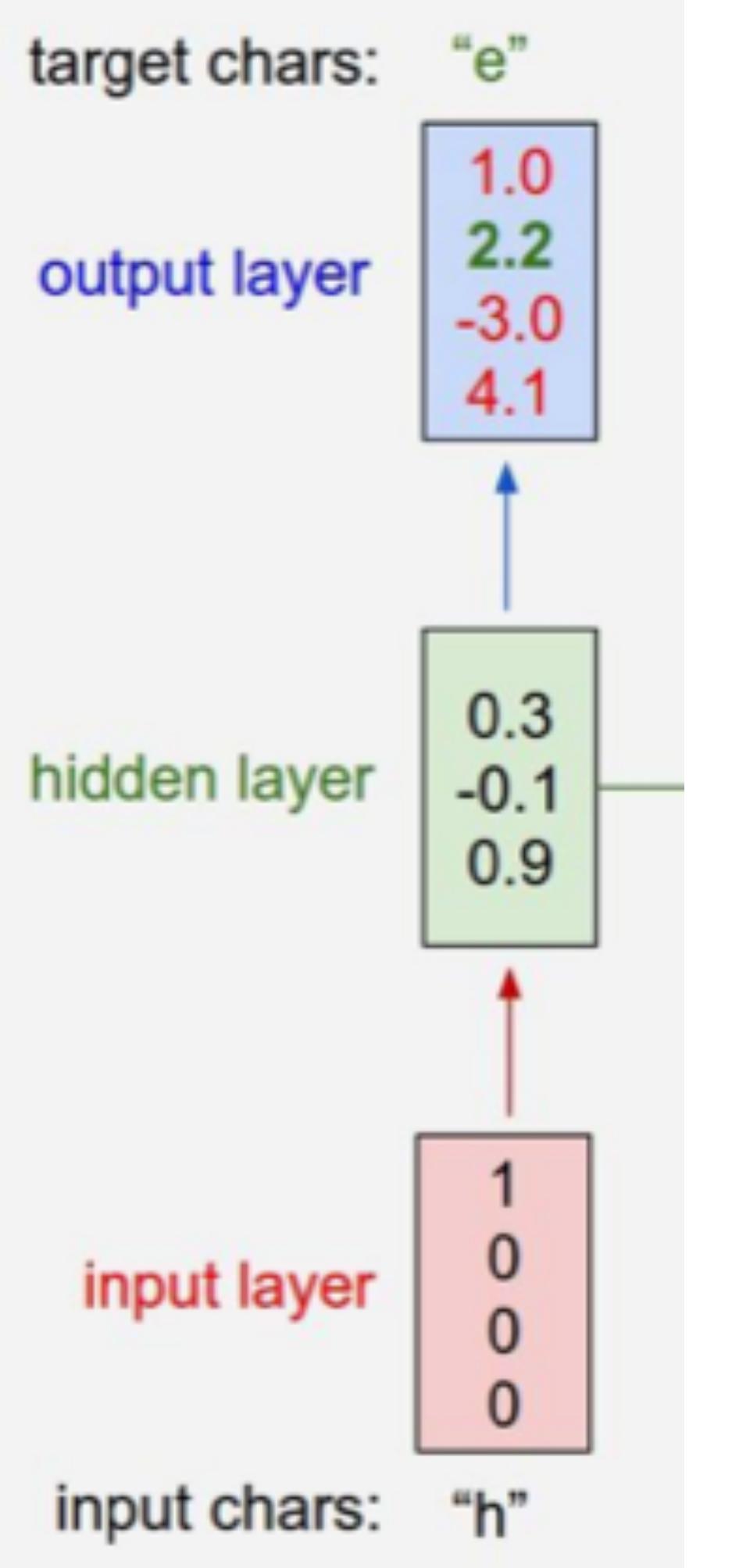
```
# RNN model
```

```
rnn_cell = rnn_cell.BasicRNNCell(rnn_size)
```

```
rnn_cell = rnn_cell.BasicLSTMCell(rnn_size)
```

```
rnn_cell = rnn_cell.GRUCell(rnn_size)
```



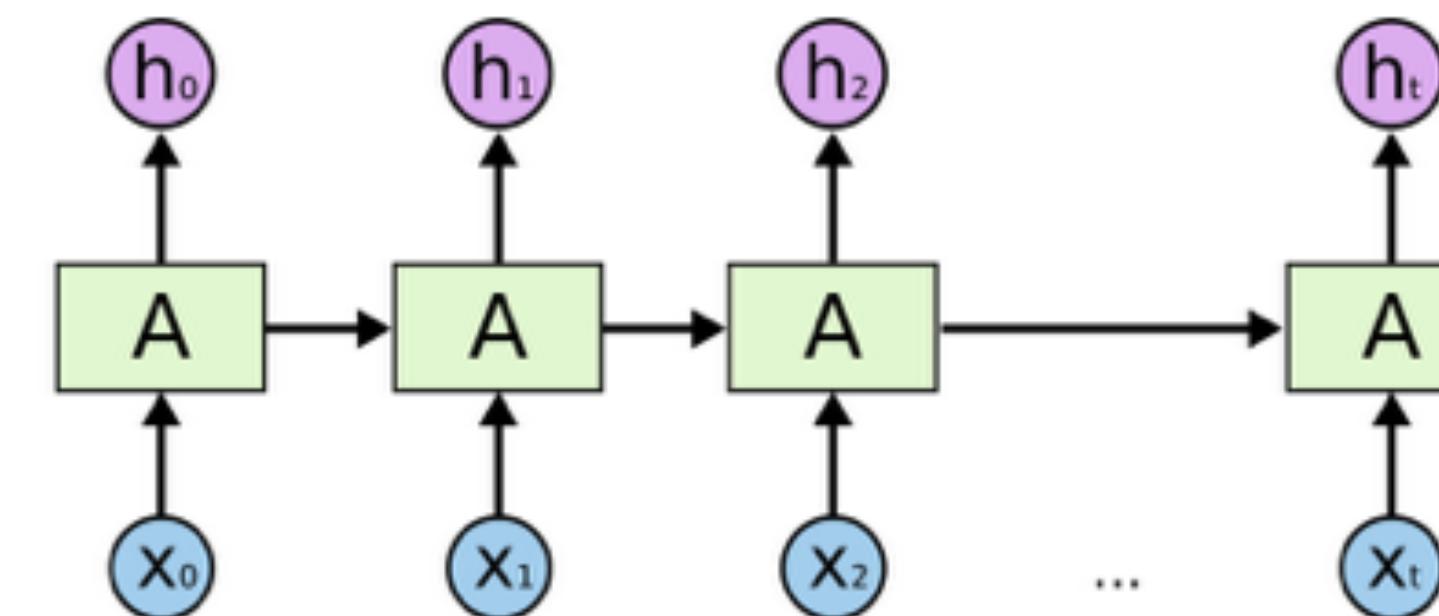


RNN in TensorFlow

```
# RNN model  
rnn_cell = rnn_cell.BasicRNNCell(4)
```

```
outputs, state = rnn.rnn(rnn_cell, X_split, state)
```

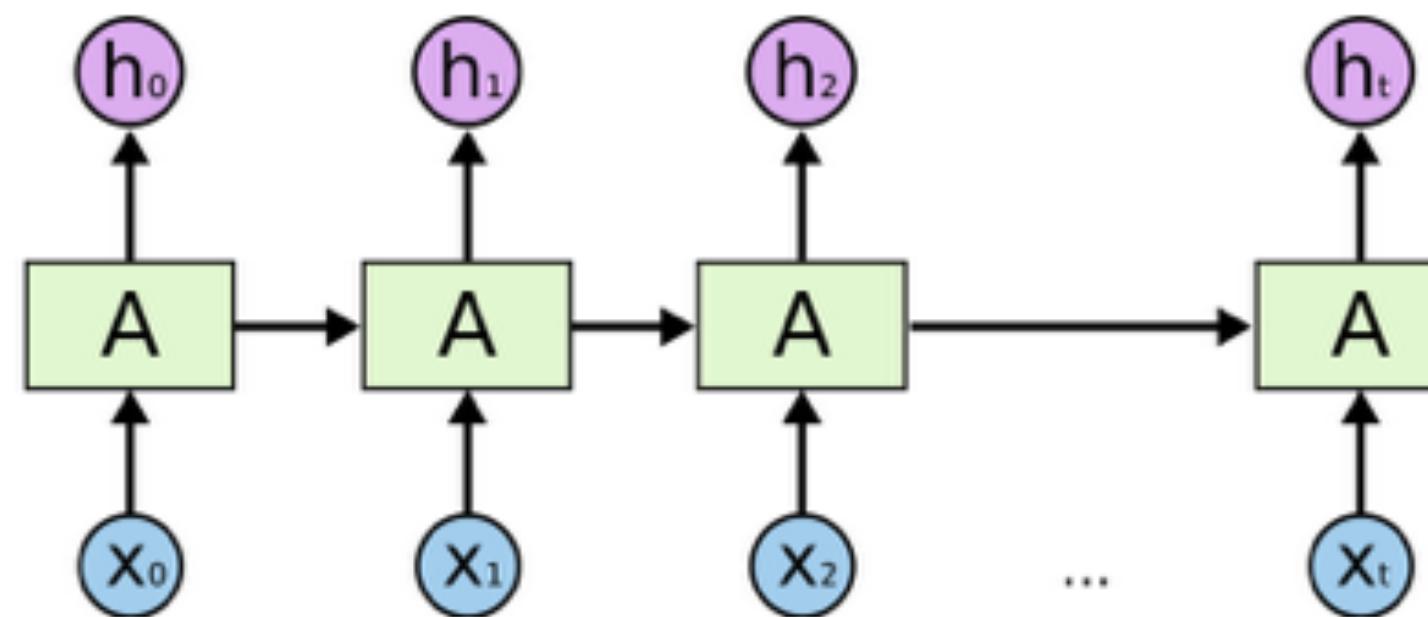
```
outputs, state = rnn.rnn(rnn_cell, X_split, state)
```



```
[<tf.Tensor 'split:0' shape=(1, 4) dtype=float32>,
 <tf.Tensor 'split:1' shape=(1, 4) dtype=float32>,
 <tf.Tensor 'split:2' shape=(1, 4) dtype=float32>,
 <tf.Tensor 'split:3' shape=(1, 4) dtype=float32>]
```

```
outputs, state = rnn.rnn(rnn_cell, X_split, state)
```

```
[<tf.Tensor 'RNN/BasicRNNCell/Tanh:0' shape=(1, 4) dtype=float32>,
 <tf.Tensor 'RNN/BasicRNNCell_1/Tanh:0' shape=(1, 4) dtype=float32>,
 <tf.Tensor 'RNN/BasicRNNCell_2/Tanh:0' shape=(1, 4) dtype=float32>,
 <tf.Tensor 'RNN/BasicRNNCell_3/Tanh:0' shape=(1, 4) dtype=float32>]
```



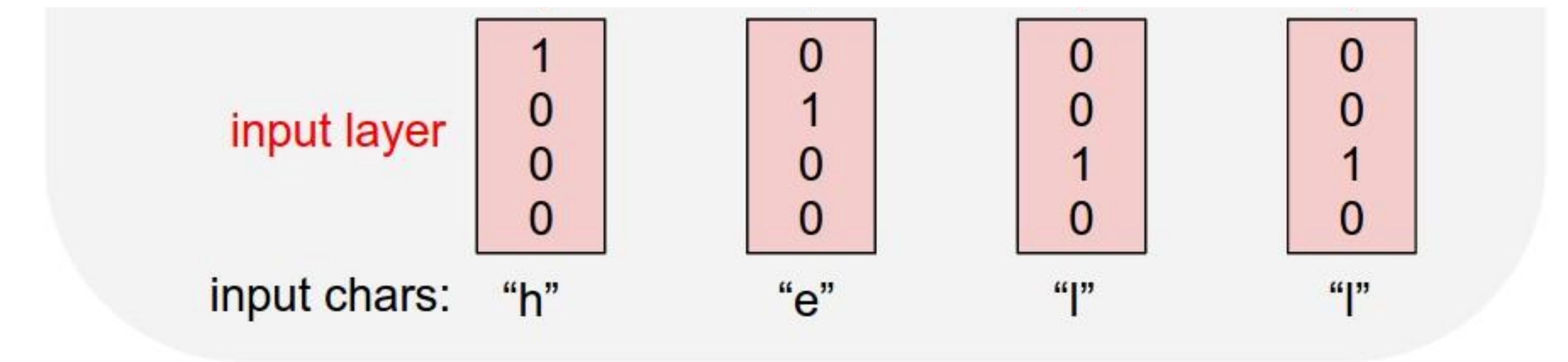
```
[<tf.Tensor 'split:0' shape=(1, 4) dtype=float32>,
 <tf.Tensor 'split:1' shape=(1, 4) dtype=float32>,
 <tf.Tensor 'split:2' shape=(1, 4) dtype=float32>,
 <tf.Tensor 'split:3' shape=(1, 4) dtype=float32>]
```

Character-level language model example

Vocabulary:
[h,e,l,o]

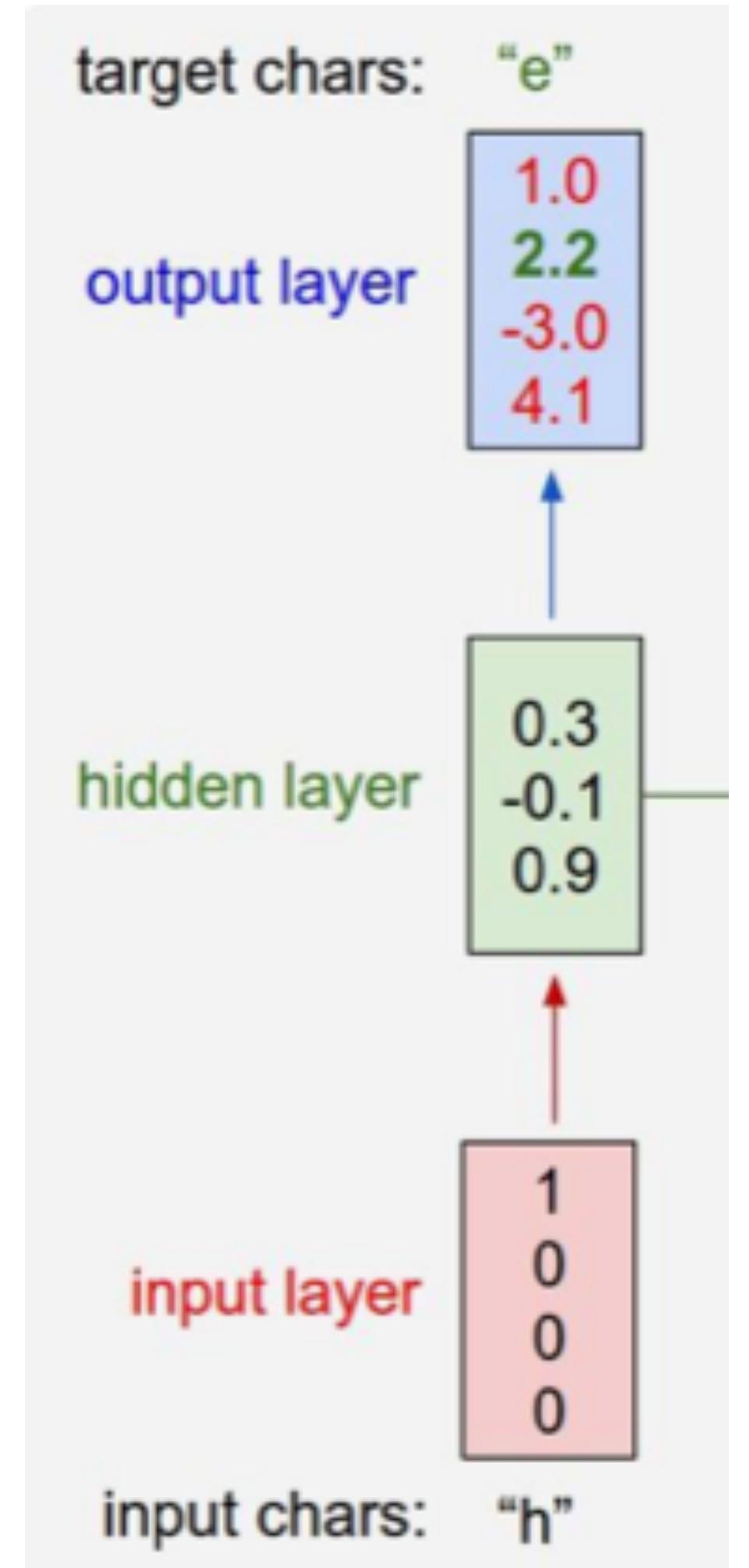
Example training
sequence:
“hello”

```
x_data = np.array([ [1,0,0,0], # h  
[0,1,0,0], # e  
[0,0,1,0], # l  
[0,0,1,0]], # l  
dtype='f')
```



RNN in TensorFlow

```
# RNN model  
rnn_cell = rnn_cell.BasicRNNCell(rnn_size)  
state = tf.zeros([batch_size, rnn_cell.state_size])  
X_split = tf.split(0, time_step_size, x_data)  
outputs, state = rnn.rnn(rnn_cell, X_split, state)
```



Cost

```
# logits: list of 2D Tensors of shape [batch_size x num_decoder_symbols].  
# targets: list of 1D batch-sized int32 Tensors of the same length as logits.  
# weights: list of 1D batch-sized float-Tensors of the same length as logits.  
logits = tf.reshape(tf.concat(1, outputs), [-1, rnn_size])  
targets = tf.reshape(sample[1:], [-1])  
weights = tf.ones([time_step_size * batch_size])  
  
loss = tf.nn.seq2seq.sequence_loss_by_example([logits], [targets], [weights])  
cost = tf.reduce_sum(loss) / batch_size  
train_op = tf.train.RMSPropOptimizer(0.01, 0.9).minimize(cost)
```

Train & Prediction

```
# Launch the graph in a session  
with tf.Session() as sess:  
    # you need to initialize all variables  
    tf.initialize_all_variables().run()  
    for i in range(100):  
        sess.run(train_op)  
        result = sess.run(tf.argmax(logits, 1))  
        print (result, [char_rdic[t] for t in result])
```

```

import tensorflow as tf
from tensorflow.models.rnn import rnn, rnn_cell
import numpy as np

char_rdic = ['h','e','l','o'] # id -> char
char_dic = {w: i for i, w in enumerate(char_rdic)} # char -> id
x_data = np.array([ [1,0,0,0], # h
                    [0,1,0,0], # e
                    [0,0,1,0], # l
                    [0,0,1,0]], # l
                  dtype='f')

sample = [char_dic[c] for c in "hello"] # to index

# Configuration
char_vocab_size = len(char_dic)
rnn_size = char_vocab_size # 1 hot coding (one of 4)
time_step_size = 4 # 'hell' -> predict 'ello'
batch_size = 1 # one sample

# RNN model
rnn_cell = rnn_cell.BasicRNNCell(rnn_size)
state = tf.zeros([batch_size, rnn_cell.state_size])
X_split = tf.split(0, time_step_size, x_data)
outputs, state = rnn.rnn(rnn_cell, X_split, state)

# logits: list of 2D Tensors of shape [batch_size x num_decoder_symbols].
# targets: list of 1D batch-sized int32 Tensors of the same length as logits.
# weights: list of 1D batch-sized float-Tensors of the same length as logits.
logits = tf.reshape(tf.concat(1, outputs), [-1, rnn_size])
targets = tf.reshape(sample[1:], [-1])
weights = tf.ones([time_step_size * batch_size])

loss = tf.nn.seq2seq.sequence_loss_by_example([logits], [targets], [weights])
cost = tf.reduce_sum(loss) / batch_size
train_op = tf.train.RMSPropOptimizer(0.01, 0.9).minimize(cost)

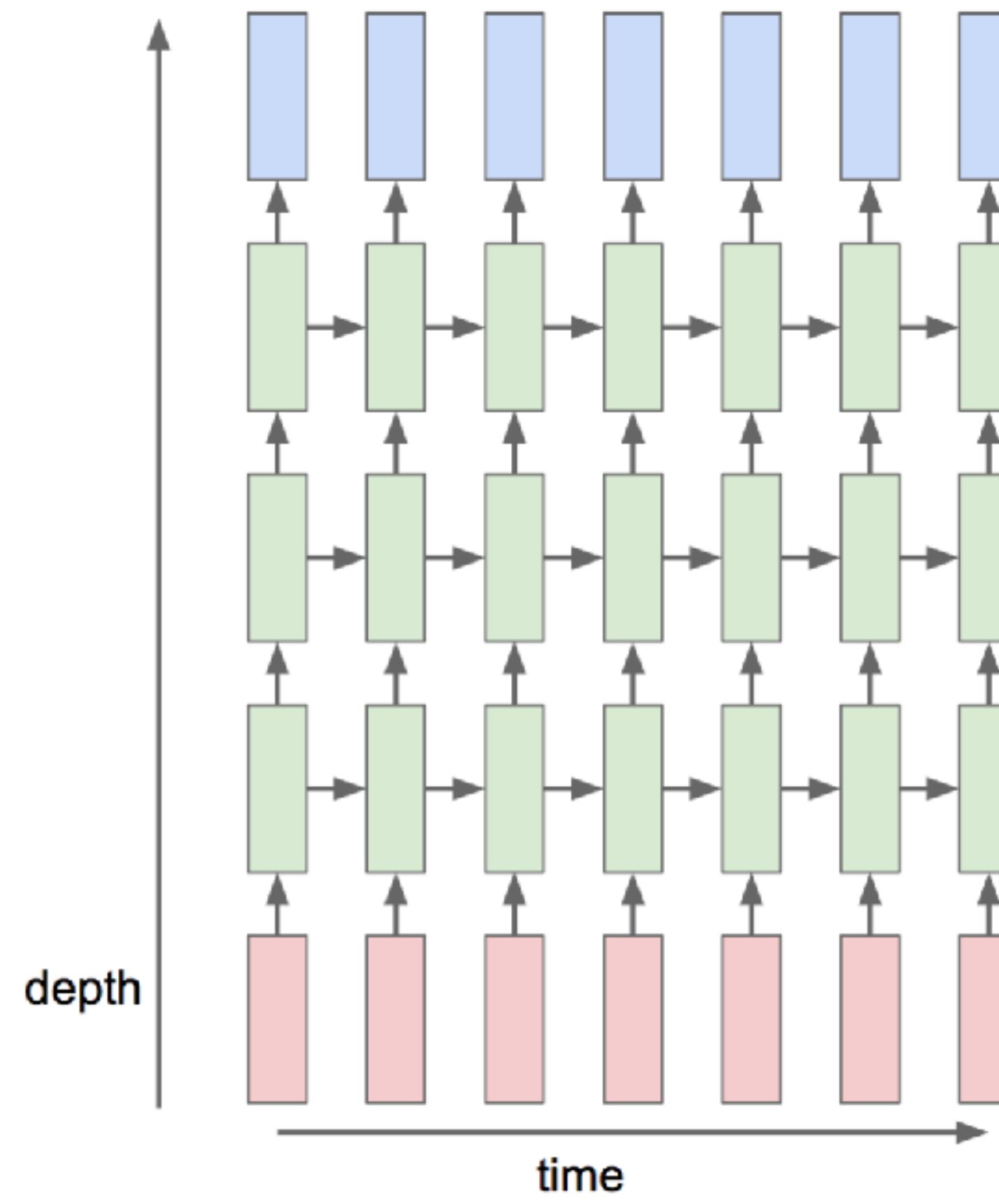
# Launch the graph in a session
with tf.Session() as sess:
    # you need to initialize all variables
    tf.initialize_all_variables().run()
    for i in range(100):
        sess.run(train_op)
        result = sess.run(tf.argmax(logits, 1))
        print(result, [char_rdic[t] for t in result])

```

Output

```
result = sess.run(tf.argmax(logits, 1))
print(result, [char_rdic[t] for t in result])
```

```
(array([2, 0, 2, 1]), ['l', 'h', 't', 'e'])
(array([2, 2, 2, 3]), ['l', 'l', 'l', 'o'])
(array([2, 2, 2, 3]), ['l', 'l', 'l', 'o'])
(array([1, 2, 2, 3]), ['e', 'l', 't', 'o'])
(array([1, 2, 2, 3]), ['e', 'l', 't', 'o'])
(array([1, 2, 2, 3]), ['e', 'l', 't', 'o'])
```



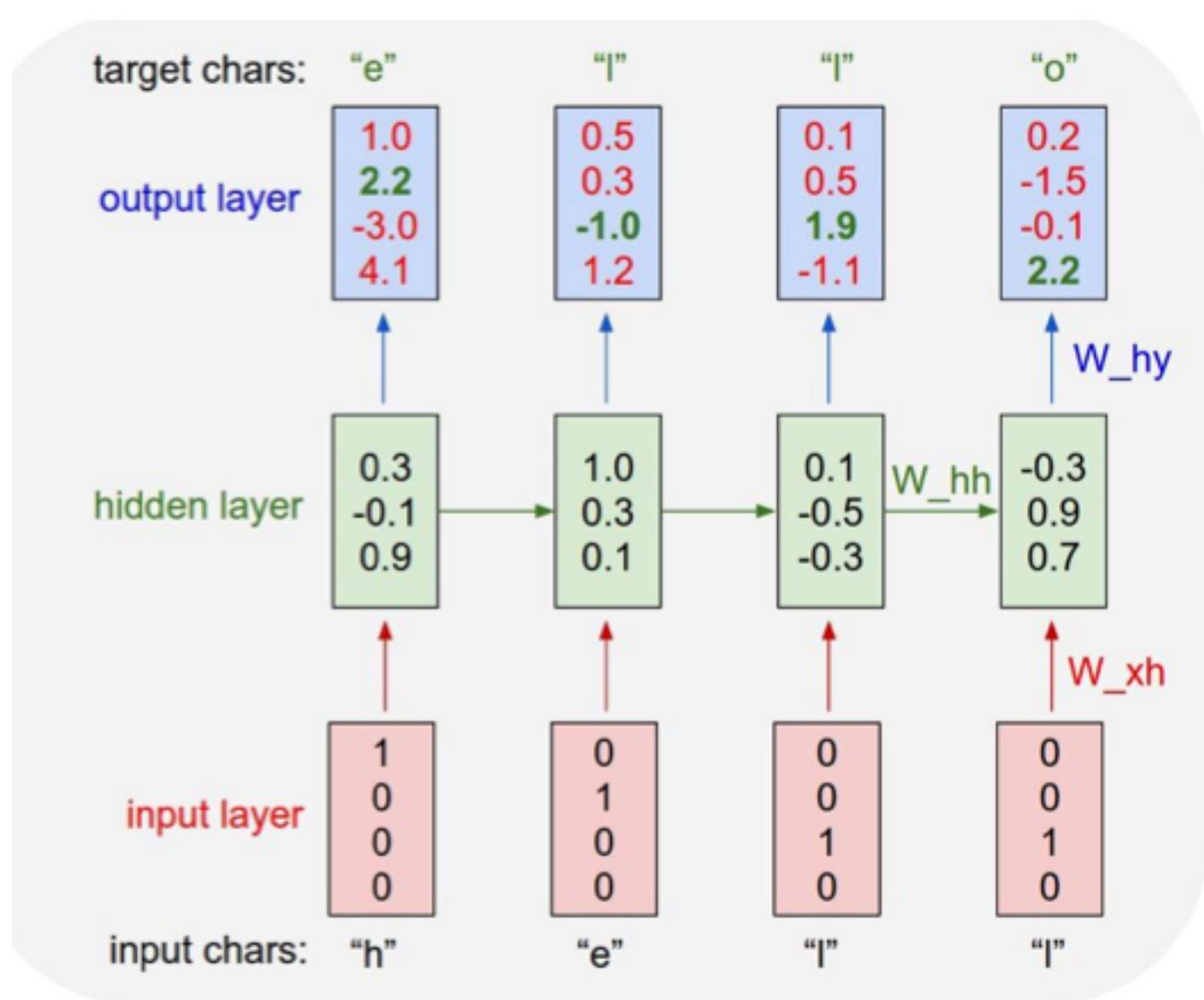
```
one_cell = rnn_cell.BasicRNNCell(rnn_size)
```

```
rnn_cell = rnn_cell.MultiRNNCell([one_cell] * depth)
```

char-rnn

Shakespeare

It looks like we can learn to spell English words. But how about if there is more structure and style in the data? To examine this I downloaded all the works of Shakespeare and concatenated them into a single (4.4MB) file. We can now afford to train a larger network, in this case lets try a 3-layer RNN with 512 hidden nodes on each layer. After we train the network for a few hours we obtain samples such as:



PANDARUS:

Alas, I think he shall be come approached and the day
When little strain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:

They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:

Well, your wit is in the care of side and that.

Second Lord:

They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:

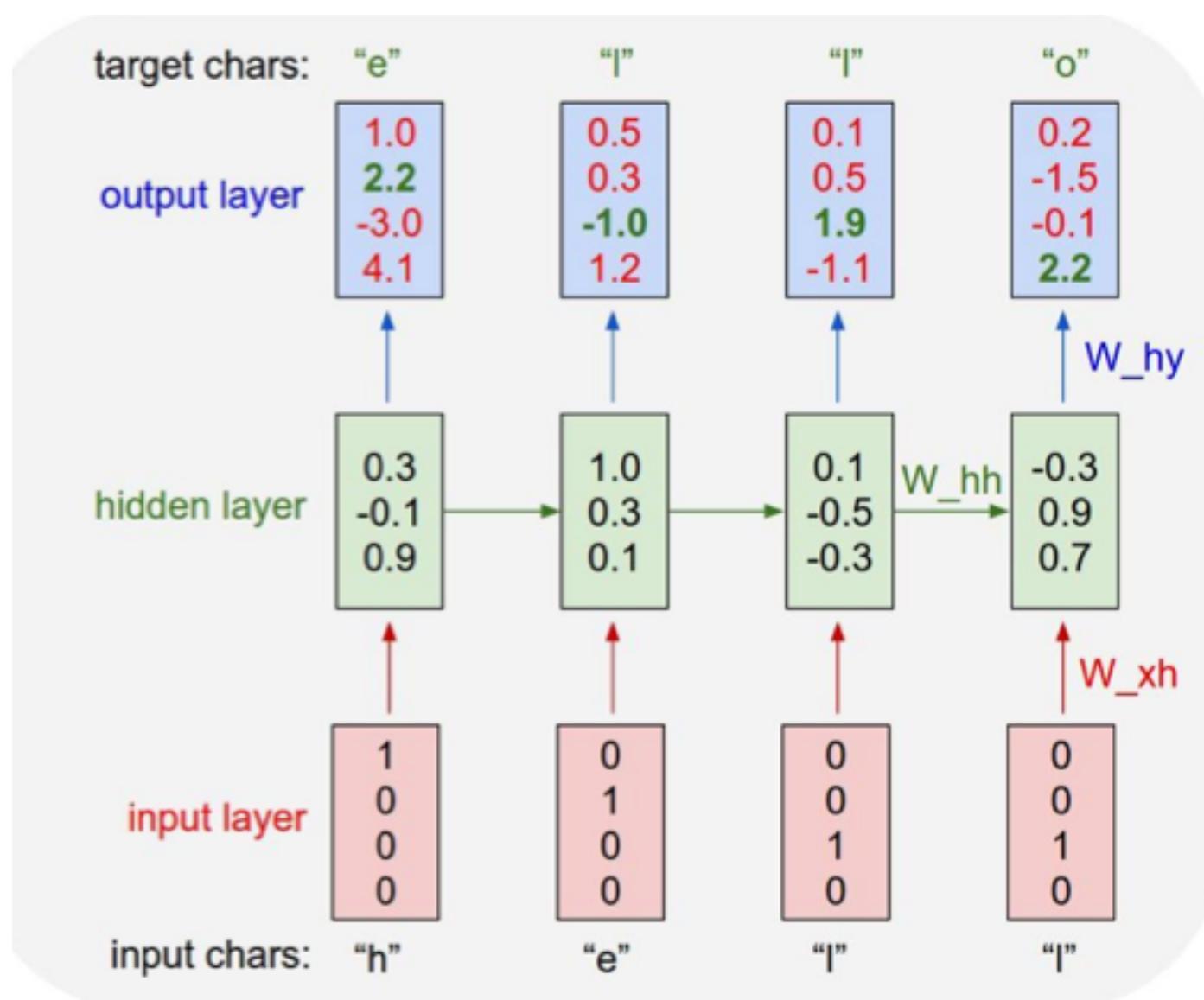
Come, sir, I will make did behold your worship.

VIOLA:

I'll drink it.

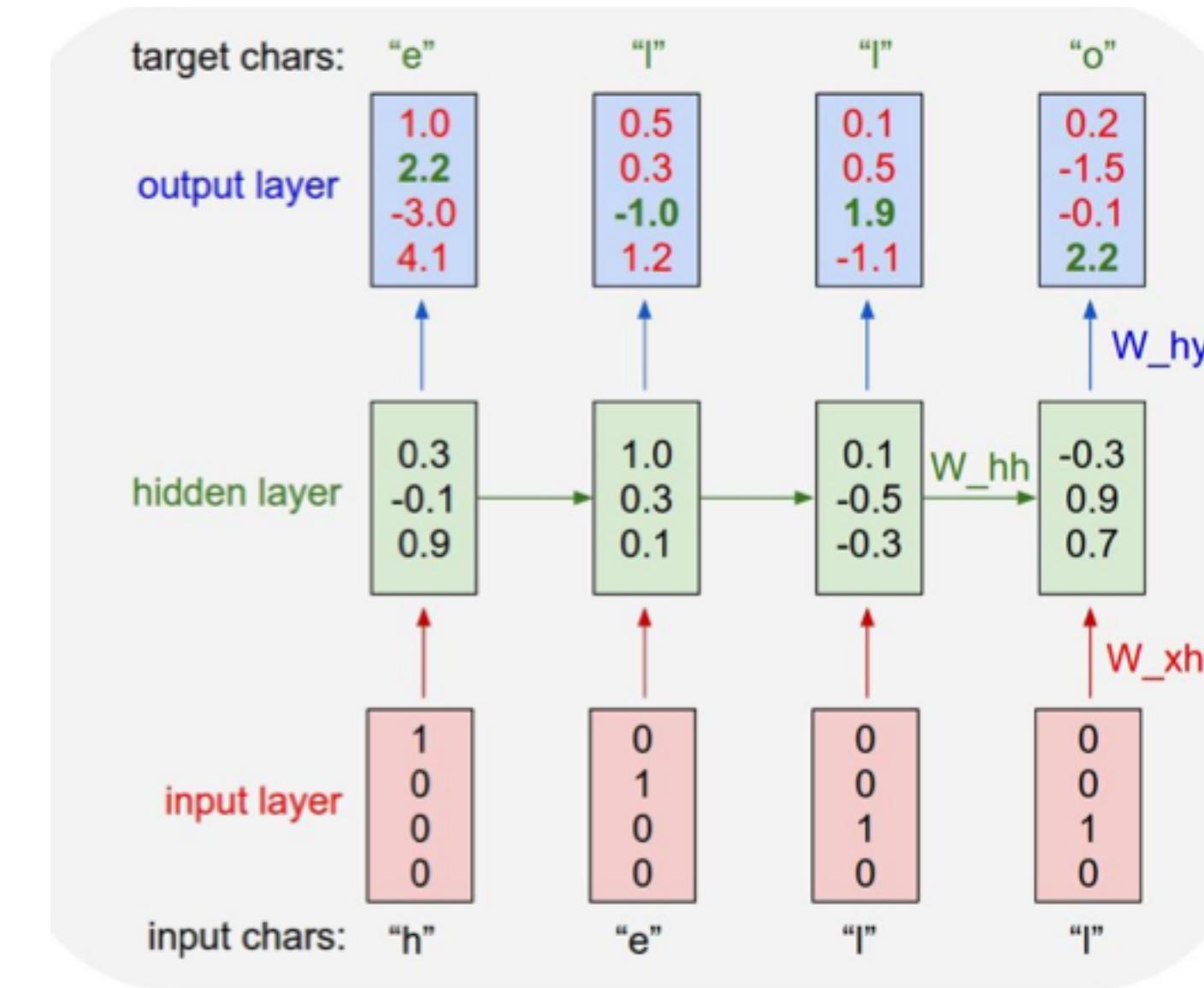
Linux Source Code

I wanted to push structured data to its limit, so for the final challenge I decided to use code. In particular, I took all the source and header files found in the [Linux repo on Github](#), concatenated all of them in a single giant file (474MB of C code) (I was originally going to train only on the kernel but that by itself is only ~16MB). Then I trained several as-large-as-fits-on-my-GPU 3-layer LSTMs over a period of a few days. These models have about 10 million parameters, which is still on the lower end for RNN models. The results are superfun:



```
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
    int error;
    if (fd == MARN_EPT) {
        /*
         * The kernel blank will coeld it to userspace.
         */
        if (ss->segment < mem_total)
            unblock_graph_and_set_blocked();
        else
            ret = 1;
        goto bail;
    }
    segaddr = in_SB(in.addr);
    selector = seg / 16;
    setup_works = true;
    for (i = 0; i < blocks; i++) {
        seq = buf[i++];
        bpf = bd->bd.next + i * search;
        if (fd) {
            current = blocked;
        }
    }
    rw->name = "Getjbbregs";
    bprm_self_clearl(&iv->version);
    regs->new = blocks[(BPF_STATS << info->historidac)] | PFMR_CLOBATHINC_SECONDS << 12;
    return segtable;
}
```

char/word rnn (char/word level n to n model)



<https://github.com/sherjilozair/char-rnn-tensorflow>

<https://github.com/hunkim/word-rnn-tensorflow>

bot.wpoem.com

신춘문예 2017 후보 시봇 (v0.003) 다시쓰기

괜찮은 행이 있으면 선택해주세요. 선택된 행들은 자동저장후 학습됩니다.

- 앞서 돌아보면 까치는 하늘 끝에서
- 초승달되도 자꾸만 안고 동안고지
- 글고양이 또통할 때마다
- 달빛 주립지
- 풀별빛 대하여 씨고 바라보아드는

이 봇은 무엇인가?

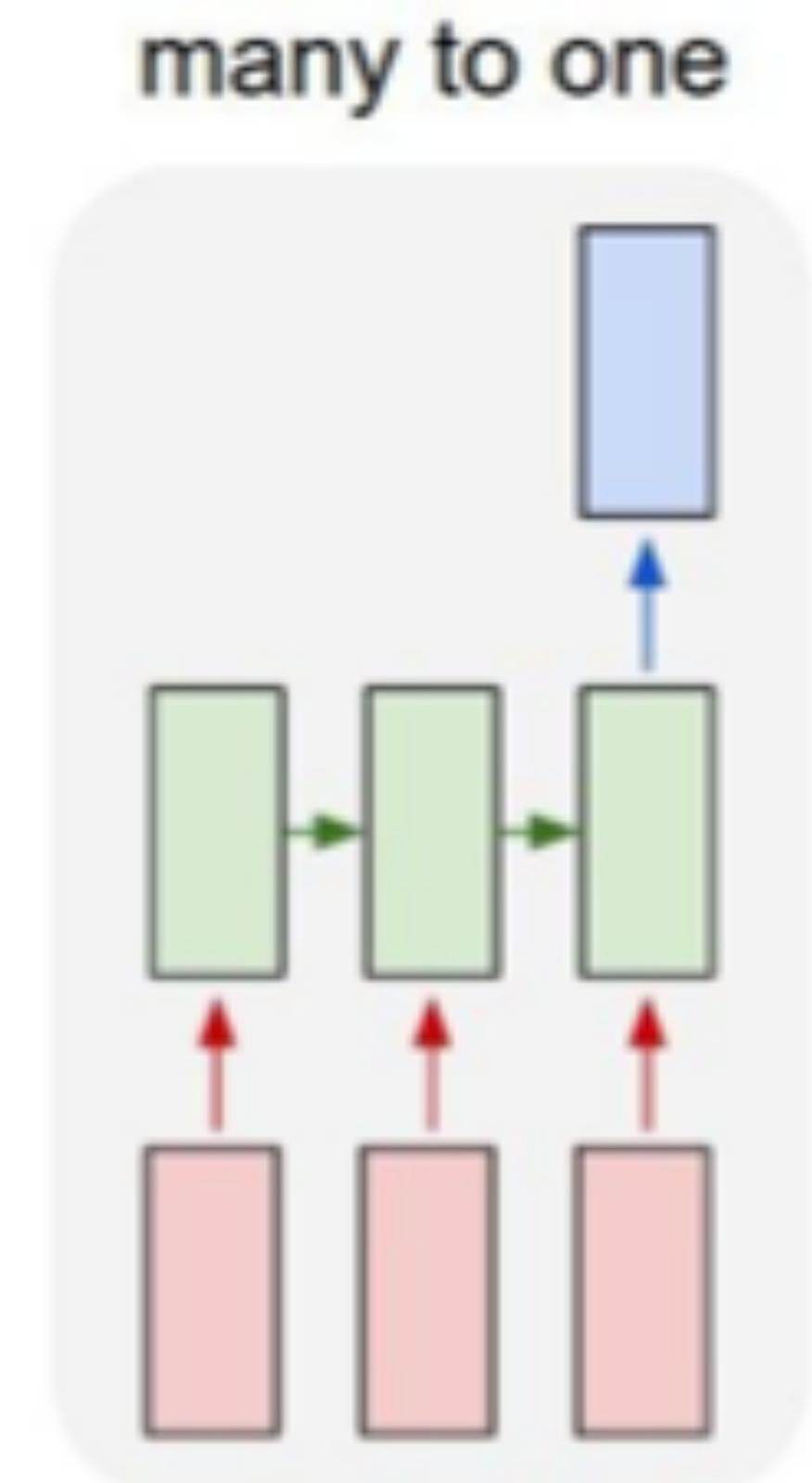
이 봇은 딥러닝을 기반으로 기존의 시에서 배워 새로운 시를 만들어 내는 프로그램입니다. 지금은 매우 엉성하지만 일년간 학습하여 2017년 신춘문예 작품을 제출하는 것을 목표로 하고 있습니다.

참여하기

- 우선 시를 보고 마음에 드는 행이 있으면 체크박스를 선택해 주시면 이 행을 추가학습에 사용합니다.
- 여러분들의 시를 아래 "시 알려주기" 입력을 통해 로봇에게 학습시켜주세요.
- 시봇 알고리즘에 기여하고 싶으시면 <https://github.com/DeepLearningProjects/poem-bot>에 참여하시면 됩니다.
 - 지금은 매우 단순한 문자 RNN 학습방법입니다. 추후 단어 레벨이나 attention 등을 추가 할 예정입니다.
- 딥러닝에 대해 자세히 알고 싶으시면 <https://hunkim.github.io/ml/> 을 참고 하시면 됩니다.

(10자 이상) 시 알려주기 이거 학습해봐!

Many to one



https://github.com/nlintz/TensorFlow-Tutorials/blob/master/7_lstm.py

RNN applications

- Language Modeling
- Speech Recognition
- Machine Translation
- Conversation Modeling/Question Answering
- Image/Video Captioning
- Image/Music/Dance Generation

TensorFlow GPU

@AWS

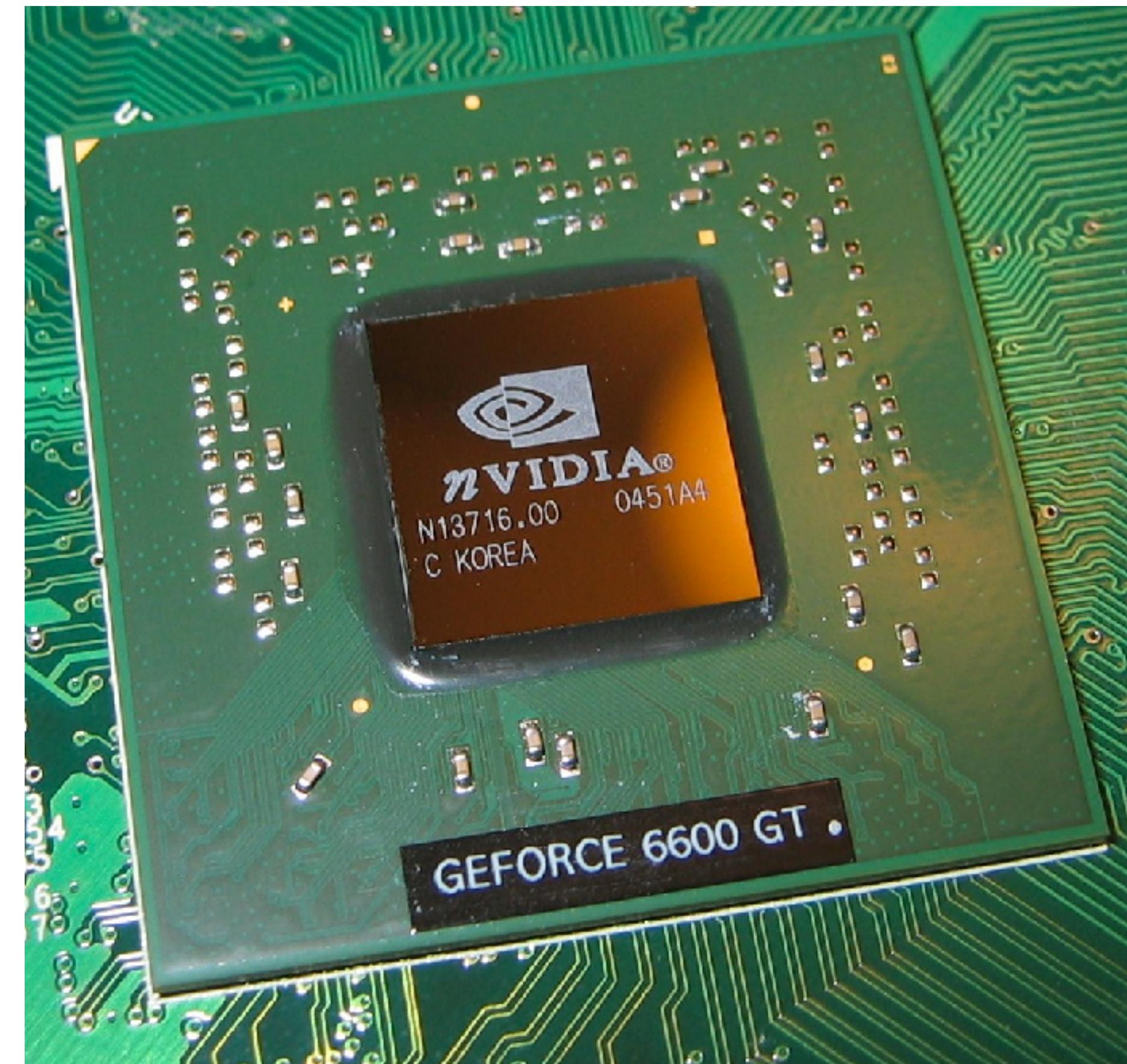
Sung Kim <hunkim+ml@gmail.com>
<http://hunkim.github.io/ml/>

Deep Network

- Takes a long time for training
 - Many forward/backward propagation and weight updates
 - Many metrics multiplications
- Very quick for testing and use in practice
 - One simple forward propagation

GPU

- A graphics processing unit (GPU), also occasionally called visual processing unit (VPU), is a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display.



GPU version

```
# Ubuntu/Linux 64-bit, GPU enabled, Python 2.7
# Requires CUDA toolkit 7.5 and CuDNN v4. For other versions, see "Install from
sources" below.
$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/linux/gpu/
tensorflow-0.9.0rc0-cp27-none-linux_x86_64.whl

# Python 2
$ sudo pip install --upgrade $TF_BINARY_URL
```





AWS GPU price in Oregon

GPU Instances - Current Generation

g2.2xlarge	8	26	15	60 SSD	\$0.65 per Hour
g2.8xlarge	32	104	60	2 x 120 SSD	\$2.6 per Hour

EC2 Console: Oregon

AWS Services Edit Sung Kim | Oregon | 

EC2 Dashboard

- Events
- Tags
- Reports
- Limits

INSTANCES

- Instances
- Spot Requests
- Reserved Instances
- Scheduled Instances
- Commands
- Dedicated Hosts

IMAGES

- AMIs
- Bundle Tasks

ELASTIC BLOCK STORE

- Volumes
- Snapshots

Resources

You are using the following Amazon EC2 resources in the US West (Oregon) region:

0 Running Instances	0 Elastic IPs
0 Dedicated Hosts	1 Snapshots
1 Volumes	0 Load Balancers
2 Key Pairs	13 Security Groups
0 Placement Groups	

Easily deploy Ruby, PHP, Java, .NET, Python, Node.js & Docker applications with [Elastic Beanstalk](#).

Create Instance

To start using Amazon EC2 you will want to launch a virtual server, known as an Amazon EC2 instance.

Launch Instance

Note: Your instances will launch in the US West (Oregon) region

Service Health

Scheduled Events

Account Attributes

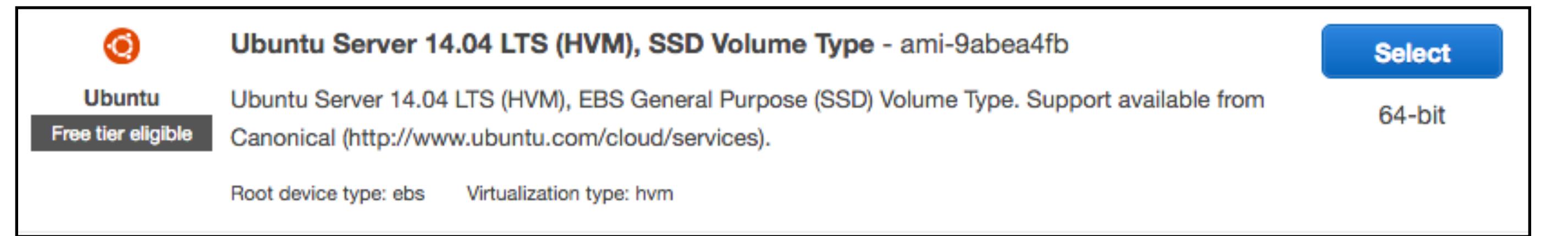
- Supported Platforms
- VPC
- Default VPC
- vpc-9b61f2fe
- Resource ID length management

Additional Information

- Getting Started Guide
- Documentation
- All EC2 Resources
- Forums
- Pricing
- Contact Us

AWS Marketplace

Find free software trial products



1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Tag Instance 6. Configure Security Group 7. Review

Step 2: Choose an Instance Type

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. [Learn more](#) about instance types and how they can meet your computing needs.

Filter by: GPU instances ▾ Current generation ▾ Show/Hide Columns

Currently selected: g2.2xlarge (26 ECUs, 8 vCPUs, 2.6 GHz, Intel Xeon E5-2670, 15 GiB memory, 1 x 60 GiB Storage Capacity)

	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
<input checked="" type="checkbox"/>	GPU instances	g2.2xlarge	8	15	1 x 60 (SSD)	Yes	High
<input type="checkbox"/>	GPU instances	g2.8xlarge	32	60	2 x 120 (SSD)	-	10 Gigabit

Step 4: Add Storage

Your instance will be launched with the following storage device settings. You can attach additional EBS volumes and instance store volumes to your instance, or edit the settings of the root volume. You can also attach additional EBS volumes after launching an instance, but not instance store volumes. [Learn more](#) about storage options in Amazon EC2.

Volume Type	Device	Snapshot	Size (GiB)	Volume Type	IOPS	Delete on Termination	Encrypted
Root	/dev/sda1	snap-306df873	12	General Purpose SSD	36 / 3000	<input checked="" type="checkbox"/>	Not Encrypted
Instance Store 0	/dev/sdb	N/A	N/A	N/A	N/A	<input type="checkbox"/>	Not Encrypted

Add New Volume

ubuntu,
GPU,
12G or more

key to access the server

Select an existing key pair or create a new key pair X

A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance.

Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about [removing existing key pairs from a public AMI](#).

Select a key pair

I acknowledge that I have access to the selected private key file (hunkim-oregon.pem), and that without this file, I won't be able to log into my instance.

EC2: Create an instance

The screenshot shows the AWS EC2 Instances page. The top navigation bar includes the AWS logo, Services dropdown, Edit dropdown, and user information (Sung Kim, Oregon, Support). The left sidebar has a tree view with 'Instances' selected under 'INSTANCES'. The main content area shows a table with one row:

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status
i-4ad0c48d	i-4ad0c48d	g2.2xlarge	us-west-2b	shutting-do...	None	

Below the table, a detailed view for instance i-4ad0c48d shows the following information:

Instance: i-4ad0c48d Public DNS: -

Description Status Checks Monitoring Tags

Instance ID	Public DNS	Public IP	Elastic IP
i-4ad0c48d	-		
Instance state	shutting-down		
Instance type	g2.2xlarge		
Private DNS	-		
Availability zone	us-west-2b		

It's ready to ssh!

The screenshot shows the AWS EC2 Instance Details page for an instance named i-068c65db. The instance is running in the g2.2xlarge instance type, located in the us-west-2c availability zone, with a public DNS of ec2-54-186-153-9.us-west-2.compute.amazonaws.com. The instance state is running, and it has status checks and alarm status both set to None. The Public IP is 54.186.153.9. The private DNS is ip-172-31-15-155.us-west-2.compute.internal.

Instance Details							
Description		Status Checks		Monitoring		Tags	
Attribute	Value	Attribute	Value	Attribute	Value	Attribute	Value
Instance ID	i-068c65db	Public DNS	ec2-54-186-153-9.us-west-2.compute.amazonaws.com	Public IP	54.186.153.9	Elastic IP	-
Instance state	running	Availability zone	us-west-2c	Private DNS	ip-172-31-15-155.us-west-2.compute.internal	Region	US West (Oregon)
Instance type	g2.2xlarge	Network interface	eni-00000000	Subnet	subnet-00000000	Security group	sg-00000000

Requires CUDA and CuDNN

```
# Ubuntu/Linux 64-bit, GPU enabled, Python 2.7
# Requires CUDA toolkit 7.5 and CuDNN v4. For other versions, see "Install from
sources" below.
$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/linux/gpu/
tensorflow-0.9.0rc0-cp27-none-linux_x86_64.whl

# Python 2
$ sudo pip install --upgrade $TF_BINARY_URL
```

CUDA

Seven Story Rabbit Hole

Sometimes awesome things happen in deep rabbit holes. Or not.



Blog | Archives

Search

NOV 22ND, 2015

CUDA 7.5 on AWS GPU Instance Running Ubuntu 14.04

Launch stock Ubuntu AMI

- Launch **ami-d05e75b8**
- Choose a GPU instance type: **g2.2xlarge** or **g2.8xlarge**
- Increase the size of the storage (this depends on what else you plan to install, I'd suggest at least 20 GB)

SSH in

```
1 $ ssh ubuntu@<instance ip>
```

About Me

I'm a software engineer at Couchbase, working on [Couchbase Mobile](#) -- an open source mobile NoSQL database with built-in sync capabilities.

In my spare time I am working on [ElasticThought](#) -- a scalable containerized REST API wrapper for the Caffe Deep Learning toolkit.

Follow me on twitter: [@tleydn](#)

Recent Posts

[Setting Up a Self-hosted drone.io CI Server](#)

[Adding Vendoring to a Go Project](#)

[Configure Emacs as a Go Editor From Scratch Part 3](#)

[Octopress Under Docker](#)

[Setting Up Unisock With APNS](#)

cuDNN

Optional: cuDNN

One can apply for the developer program here

<https://developer.nvidia.com/cudnn>. When approved, download cuDNN for Linux (either v4 RC or v3 is fine), upload the cuDNN package from the local computer to the instance, and install cuDNN:

```
tar -zxf cudnn-7.0-linux-x64-v4.0-rc.tgz #or cudnn-7.0-linux-x64-v3.0
cd cuda
sudo cp lib64/* /usr/local/cuda/lib64/
sudo cp include/cudnn.h /usr/local/cuda/include/
```

16 commands

- 1 wget http://developer.download.nvidia.com/.../cuda-repo-ubuntu1404...
- 2 sudo dpkg -i cuda-repo-ubuntu1404_7.5-18_amd64.deb
- 3 sudo apt-get update
- 4 sudo apt-get upgrade -y
- 5 sudo apt-get install -y opencl-headers build-essential protobuf-compiler libprotobuf-dev libboost-all-dev libleveldb-dev hdf5-tools libhdf5-serial-dev libopencv-core-dev libopencv-highgui-dev libsnavy-dev libsnavy1 libatlas-base-dev cmake libstdc++6-4.8-dbg libgoogle-glog0 libgoogle-glog-dev libgflags-dev liblmdb-dev git python-pip gfortran
- 6 sudo apt-get clean
- 7 sudo apt-get install -y linux-image-extra-`uname -r` linux-headers-`uname -r` linux-image-`uname -r`
- 8 sudo apt-get install -y cuda
- 9 nvidia-smi
- 10 sudo apt-get install python-pip python-dev
- 11 sudo pip install --upgrade https://storage.googleapis.com/.../tensorflow-0.8.0rc0-cp27-n...
- 12 git clone https://github.com/nlintz/TensorFlow-Tutorials
- 13 cd TensorFlow-Tutorials/
- 14 vi ~/.profile # add PATH, LD PATH
- 15 source ~/.profile
- 16 python 06_autoencoder.py

Add Path

- `export PATH=/usr/local/cuda/bin:$PATH`
- `export LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH`

Requires CUDA and CuDNN

```
# Ubuntu/Linux 64-bit, GPU enabled, Python 2.7
# Requires CUDA toolkit 7.5 and CuDNN v4. For other versions, see "Install from
sources" below.
$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/linux/gpu/
tensorflow-0.9.0rc0-cp27-none-linux_x86_64.whl

# Python 2
$ sudo pip install --upgrade $TF_BINARY_URL
```

Reuse ami-9e39dcf3 (N.Virginia) ami-38f60658 (oregon)

The screenshot shows the AWS EC2 Dashboard with the 'AMIs' tab selected. The main area displays a search results table for AMIs owned by the user. A search bar at the top shows the query 'search : ami-9e39dcf3'. The table has columns for Name, AMI Name, AMI ID, Source, Owner, Visibility, Status, and Creation Date. One row is visible, corresponding to the search term, with the name 'TF0.8_cuda7.4...' and the AMI ID 'ami-9e39dcf3'. The status is listed as 'available'.

Name	AMI Name	AMI ID	Source	Owner	Visibility	Status	Creation Date
TF0.8_cuda7.4...	ami-9e39dcf3	705119112097/T...	705119112097	Public	available	April 27, 2016 at 8:42:10 PM ...	

Perhaps, it will be unavailable when later CUDA versions are out.

Creating TensorFlow device (/gpu:0)

```
05] successfully opened CUDA library libcublas.so locally
9] Couldn't open CUDA library libcudnn.so. LD_LIBRARY_PATH: /usr/local/cuda/lib64:
c:1562] Unable to load cuDNN DSO
05] successfully opened CUDA library libcufft.so locally
05] successfully opened CUDA library libcuda.so.1 locally
05] successfully opened CUDA library libcurand.so locally
-ubyte.gz')
cationWarning: converting an array with ndim > 0 to an index will result in an error in the
e]
.py:42: VisibleDeprecationWarning: converting an array with ndim > 0 to an index will result
, 1)
-ubyte.gz')
ubyte.gz')
ubyte.gz')
xecutor.cc:900] successful NUMA node read from SysFS had negative value (-1), but there must
t.cc:102] Found device 0 with properties:
```

7

```
t.cc:126] DMA: 0
t.cc:136] 0:  Y
ice.cc:755] Creating TensorFlow device (/gpu:0) -> (device: 0, name: GRID K520, pci bus id:
locator.cc:244] PoolAllocator: After 1704 get requests, put_count=1321 evicted_count=1000 ev
locator.cc:256] Raising pool_size_limit_ from 100 to 110
locator.cc:244] PoolAllocator: After 1704 get requests, put_count=1812 evicted_count=1000 ev
locator.cc:256] Raising pool_size_limit_ from 256 to 281
```

```
482/6750 (epoch 3), train_loss = 7.503, time/batch = 0.237
483/6750 (epoch 3), train_loss = 7.612, time/batch = 0.239
484/6750 (epoch 3), train_loss = 7.473, time/batch = 0.248
485/6750 (epoch 3), train_loss = 7.603, time/batch = 0.241
486/6750 (epoch 3), train_loss = 7.519, time/batch = 0.248
487/6750 (epoch 3), train_loss = 7.526, time/batch = 0.238
488/6750 (epoch 3), train_loss = 7.356, time/batch = 0.243
489/6750 (epoch 3), train_loss = 7.550, time/batch = 0.239
490/6750 (epoch 3), train_loss = 7.440, time/batch = 0.243
491/6750 (epoch 3), train_loss = 7.514, time/batch = 0.240
492/6750 (epoch 3), train_loss = 7.514, time/batch = 0.242
493/6750 (epoch 3), train_loss = 7.467, time/batch = 0.245
494/6750 (epoch 3), train_loss = 7.351, time/batch = 0.239
495/6750 (epoch 3), train_loss = 7.553, time/batch = 0.245
496/6750 (epoch 3), train_loss = 7.373, time/batch = 0.240
497/6750 (epoch 3), train_loss = 7.493, time/batch = 0.242
498/6750 (epoch 3), train_loss = 7.445, time/batch = 0.243
499/6750 (epoch 3), train_loss = 7.432, time/batch = 0.240
500/6750 (epoch 3), train_loss = 7.476, time/batch = 0.246
501/6750 (epoch 3), train_loss = 7.463, time/batch = 0.238
502/6750 (epoch 3), train_loss = 7.477, time/batch = 0.241
503/6750 (epoch 3), train_loss = 7.495, time/batch = 0.244
504/6750 (epoch 3), train_loss = 7.543, time/batch = 0.240
505/6750 (epoch 3), train_loss = 7.550, time/batch = 0.241
506/6750 (epoch 3), train_loss = 7.567, time/batch = 0.242
507/6750 (epoch 3), train_loss = 7.415, time/batch = 0.247
508/6750 (epoch 3), train_loss = 7.414, time/batch = 0.236
509/6750 (epoch 3), train_loss = 7.540, time/batch = 0.242
510/6750 (epoch 3), train_loss = 7.402, time/batch = 0.245
511/6750 (epoch 3), train_loss = 7.548, time/batch = 0.241
512/6750 (epoch 3), train_loss = 7.385, time/batch = 0.242
513/6750 (epoch 3), train_loss = 7.563, time/batch = 0.238
514/6750 (epoch 3), train_loss = 7.488, time/batch = 0.241
515/6750 (epoch 3), train_loss = 7.504, time/batch = 0.245
516/6750 (epoch 3), train_loss = 7.546, time/batch = 0.243
517/6750 (epoch 3), train_loss = 7.521, time/batch = 0.244
518/6750 (epoch 3), train_loss = 7.384, time/batch = 0.246
519/6750 (epoch 3), train_loss = 7.402, time/batch = 0.242
520/6750 (epoch 3), train_loss = 7.552, time/batch = 0.244
521/6750 (epoch 3), train_loss = 7.499, time/batch = 0.250
522/6750 (epoch 3), train_loss = 7.342, time/batch = 0.239
523/6750 (epoch 3), train_loss = 7.378, time/batch = 0.237
524/6750 (epoch 3), train_loss = 7.391, time/batch = 0.242
525/6750 (epoch 3), train_loss = 7.348, time/batch = 0.242
526/6750 (epoch 3), train_loss = 7.270, time/batch = 0.242
527/6750 (epoch 3), train_loss = 7.381, time/batch = 0.244
```

```
model saved to save/model.ckpt
1/6750 (epoch 0), train_loss = 11.127, time/batch = 4.697
2/6750 (epoch 0), train_loss = 11.176, time/batch = 4.430
3/6750 (epoch 0), train_loss = 10.946, time/batch = 4.604
4/6750 (epoch 0), train_loss = 10.616, time/batch = 4.483
5/6750 (epoch 0), train_loss = 10.207, time/batch = 5.354
6/6750 (epoch 0), train_loss = 9.723, time/batch = 5.674
7/6750 (epoch 0), train_loss = 9.449, time/batch = 5.972
8/6750 (epoch 0), train_loss = 9.089, time/batch = 5.785
9/6750 (epoch 0), train_loss = 8.861, time/batch = 5.518
10/6750 (epoch 0), train_loss = 8.797, time/batch = 5.856
11/6750 (epoch 0), train_loss = 8.688, time/batch = 5.795
12/6750 (epoch 0), train_loss = 8.814, time/batch = 5.803
13/6750 (epoch 0), train_loss = 8.728, time/batch = 5.595
14/6750 (epoch 0), train_loss = 8.938, time/batch = 5.586
15/6750 (epoch 0), train_loss = 8.657, time/batch = 5.529
16/6750 (epoch 0), train_loss = 8.736, time/batch = 5.536
17/6750 (epoch 0), train_loss = 8.643, time/batch = 5.690
18/6750 (epoch 0), train_loss = 8.985, time/batch = 5.828
19/6750 (epoch 0), train_loss = 8.792, time/batch = 5.504
20/6750 (epoch 0), train_loss = 8.882, time/batch = 5.555
21/6750 (epoch 0), train_loss = 8.750, time/batch = 5.437
22/6750 (epoch 0), train_loss = 8.872, time/batch = 5.597
[]
```



OS X El Capitan

Version 10.11.3

MacBook Pro (Retina, 13-inch, Early 2015)

Processor 3.1 GHz Intel Core i7

Memory 16 GB 1867 MHz DDR3

AWS GPU price in Oregon

GPU Instances - Current Generation					
g2.2xlarge	8	26	15	60 SSD	\$0.65 per Hour
g2.8xlarge	32	104	60	2 x 120 SSD	\$2.6 per Hour

Amazon Elastic Compute Cloud running Linux/UNIX			
\$0.650 per On Demand Linux g2.2xlarge Instance Hour		17 Hrs	\$11.05
\$2.6 per On Demand Linux g2.8xlarge Instance Hour		1 Hrs	\$2.60
Total:			\$13.65

$$2.6 * 24 * 30 = 1,872 \text{ USD}$$

Spot instances

The screenshot shows the AWS EC2 Dashboard with the 'Services' dropdown open. The 'Spot Requests' tab is selected in the left sidebar. The main area displays the 'Request Spot Instances' button, which is highlighted with a blue border. Other buttons like 'Cancel' and 'Pricing History' are also visible. A search bar and a message indicating no spot instance requests are present in the region are also shown.

AWS Services Edit Sung Kim N. Virginia

EC2 Dashboard Events Tags Reports Limits

INSTANCES Instances

Spot Requests

Reserved Instances

Request Spot Instances Cancel Pricing History

Filter by tags and attributes or search by keyword

You do not have any Spot instance requests in this region.

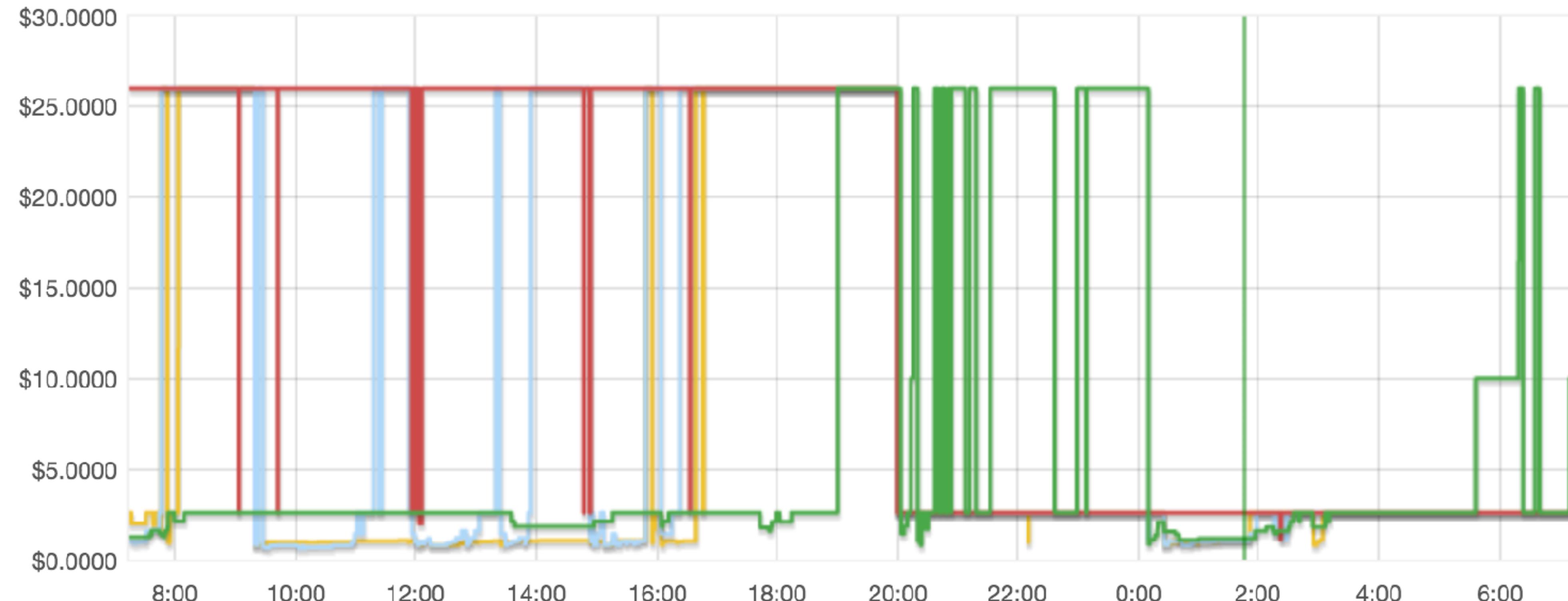
Click on the "Request Spot Instances" button to request your first Spot instance.

Request Spot Instances

Spot Instance Pricing History

X

Product : **Linux/UNIX** ▾ Instance type: **g2.8xlarge** ▾ Date range : **1 day** ▾ Availability zone: **All zones** ▾



Availability zone	Price
us-east-1a	\$1.0485
us-east-1c	\$1.0200
us-east-1d	\$2.6001

N. Virginia ▾



None found

Price bidding

Step 3: Configure Instance Details

Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot instances to take advantage of the lower pricing, assign an access management role to the instance, and more.

Number of instances	i	<input type="text" value="1"/>	Launch into Auto Scaling Group i
Purchasing option	i	<input checked="" type="checkbox"/> Request Spot instances	
Current price	i	us-east-1a	0.2428
		us-east-1c	0.1442
		us-east-1d	0.153
		us-east-1e	6.500
Maximum price	i	\$ <small>(e.g. 0.045 = 4.5 cents/hour)</small>	<input type="text"/>
Launch group	i	(Optional)	

bill, bill, bill!

AWS Services Edit Global Support

Dashboard Bills Cost Explorer Budgets Payment Methods Payment History Consolidated Billing Reports Preferences Credits Tax Settings DevPay

Billing & Cost Management Dashboard

Spend Summary

Welcome to the AWS Account Billing console. Your last month, month-to-date, and month-end forecasted costs appear below.

Current month-to-date balance for August 2015

\$6,485.39

Last Month (July 2015) Month-to-Date (August 2015) Forecast (August 2015)

\$0.6 \$6.49K \$6.59K

Important Information about these Costs Include Subscription Charges

Alerts & Notifications

Your account is enabled for monitoring estimated charges. Set your first billing alarm to receive an e-mail when charges reach a threshold you define.

IAM access to your account's billing information is not enabled. You can enable it on the Account Information page.

Month-to-Date Spend by Service

The chart below shows the proportion of costs spent for each service you use.

Service	Amount
EC2	\$6,484.99
SES	\$0.38
DataTransfer	\$0.01
Glacier	\$0.01
Other Services	\$0.00
Tax	\$0.00
Total	\$6,485.39

Bill Details

Check, stop, and terminate

The screenshot shows the AWS EC2 Dashboard. On the left, there's a sidebar with links like EC2 Dashboard, Events, Tags, Reports, Limits, INSTANCES (with sub-links Instances and Spot Requests), and a navigation bar with AWS, Services, Edit, Sung Kim, and Oregon.

The main area is titled "Resources" and displays the following statistics:

Resource Type	Count
Running Instances	0
Dedicated Hosts	0
Volumes	0
Key Pairs	2
Placement Groups	0
Elastic IPs	0
Snapshots	1
Load Balancers	0
Security Groups	14

To the right, under "Account Attributes", it shows:

- Supported Platforms: VPC
- Default VPC: Default VPC
- VPC ID: vpc-9b61f2fe
- Resource ID length management: Resource ID length management

The screenshot shows the AWS EC2 Instances page. At the top, there are buttons for Launch Instance, Connect, and Actions. A dropdown menu is open under Actions, showing options: Connect, Get Windows Password, Launch More Like This, Instance State, Instance Settings, Image, Networking, and CloudWatch Monitoring. The Instance State option is highlighted in yellow.

The main content area says "You do not have any running instances in this region." and "Want to learn more about using EC2? Check out the Getting Started Guide." Below this, there are three small circular icons.

At the bottom left, it says "Select an instance above".

CloudWatch

The screenshot shows the AWS EC2 Instances page. On the left, there's a sidebar with links like EC2 Dashboard, Events, Tags, Reports, Limits, Instances, Spot Requests, Reserved Instances, and AMIs. The Instances link is highlighted. The main area shows a table of instances. One instance, 'kbill' (i-7858ecdf), is selected. A context menu is open over this instance, with 'CloudWatch Monitoring' expanded. The options under 'CloudWatch Monitoring' are: Enable Detailed Monitoring, Disable Detailed Monitoring, and Add/Edit Alarms.

This is a modal window titled 'Alarm Details for i-7858ecdf (kbill)'. It contains the following text: 'Below are your CloudWatch alarms for the selected resources. Click on an alarm to edit it or click 'view' to see additional options and details in Amazon CloudWatch.' Below this, a table header is shown with columns for State and Name, and a 'More Options' button. The message 'There are no alarms for this instance' is displayed. At the bottom are 'Create Alarm' and 'Close' buttons.

Stop when CPU utilization ≤ 0.3

Create Alarm

You can use CloudWatch alarms to be notified automatically whenever metric data reaches a level you define.

To edit an alarm, first choose whom to notify and then define when the notification should be sent.

Send a notification to: No SNS topics found... ▾

Take the action: Recover this instance ⓘ
 Stop this instance ⓘ
 Terminate this instance ⓘ
 Reboot this instance ⓘ

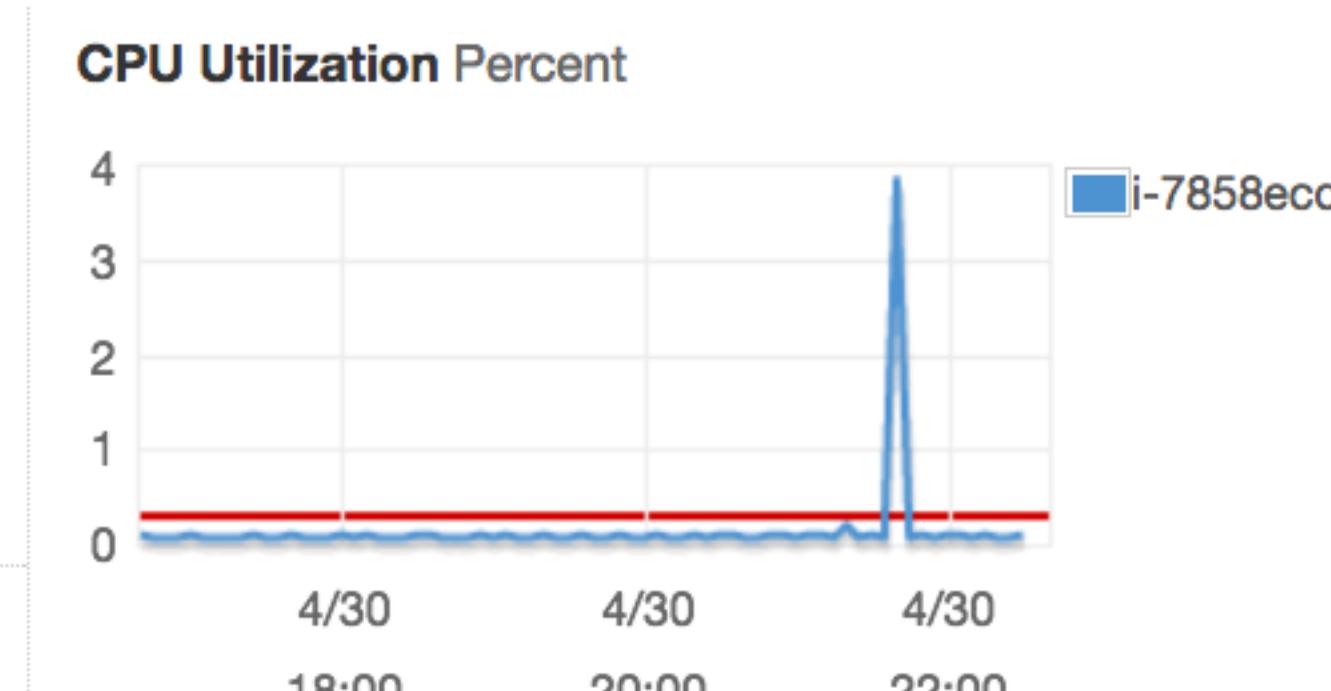
Whenever: Average of CPU Utilization ▾

Is: ≤ 0.3 Percent

For at least: 1 consecutive period(s) of 5 Minutes ▾

Name of alarm: awsec2-i-7858ecdf-High-CPU-Utilization

[Cancel](#) [Create Alarm](#)



Shutdown after training

```
$ screen
```

```
$ sudo -i
```

```
# python train.py; shutdown -h now
```

Deep learning for Everyone

Season 2 coming soon!

Sung Kim <hunkim+ml@gmail.com>
<http://hunkim.github.io/ml/>



TensorFlow GPU @AWS Spot Instances without losing data + *100 USD credit*

Sung Kim <hunkim+ml@gmail.com>
<http://hunkim.github.io/ml/>

Consulted Channy Yun and DoHyun Jung

GPU

- A graphics processing unit (GPU), also occasionally called visual processing unit (VPU), is a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display.

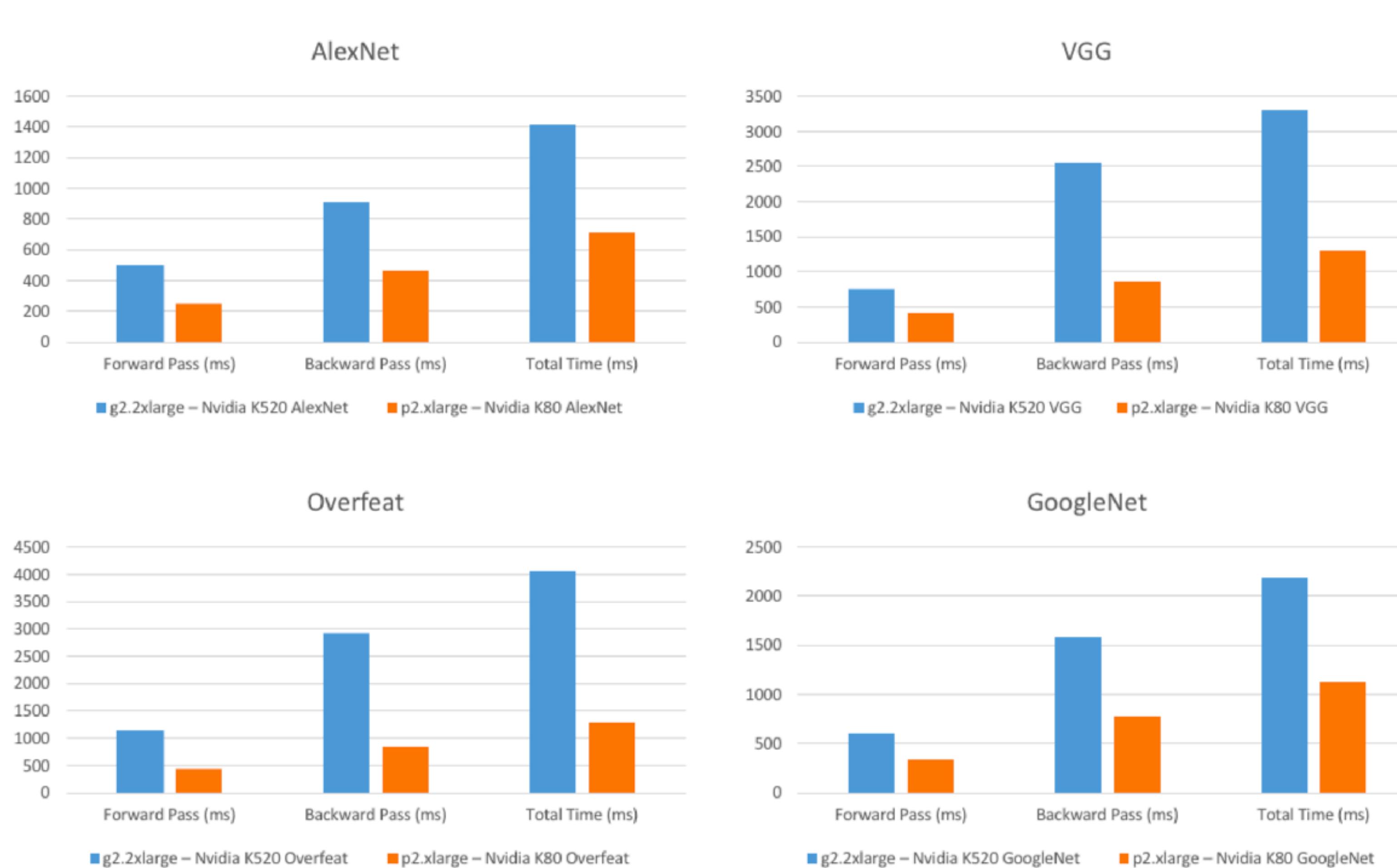


AWS G2 and P2 Instances

Model	GPUs	vCPU	Mem (GiB)	SSD Storage (GB)	GPU Memory
g2.2xlarge	1	8	15	1 x 60	4G (8G)
g2.8xlarge	4	32	60	2 x 120	4G (32G)

Instance Name	GPU Count	vCPU Count	Memory	Parallel Processing Cores	GPU Memory	Network Performance
p2.xlarge	1	4	61 GiB	2,496	12 GiB	High
p2.8xlarge	8	32	488 GiB	19,968	96 GiB	10 Gigabit
p2.16xlarge	16	64	732 GiB	39,936	192 GiB	20 Gigabit

AWS G2 and P2 Instances



TensorFlow GPU Performance: AlexNet, VGG, Overfeat, GoogleNet

<http://www.bitfusion.io/2016/11/03/quick-comparison-of-tensorflow-gpu-performance-on-aws-p2-and-g2-instances/>

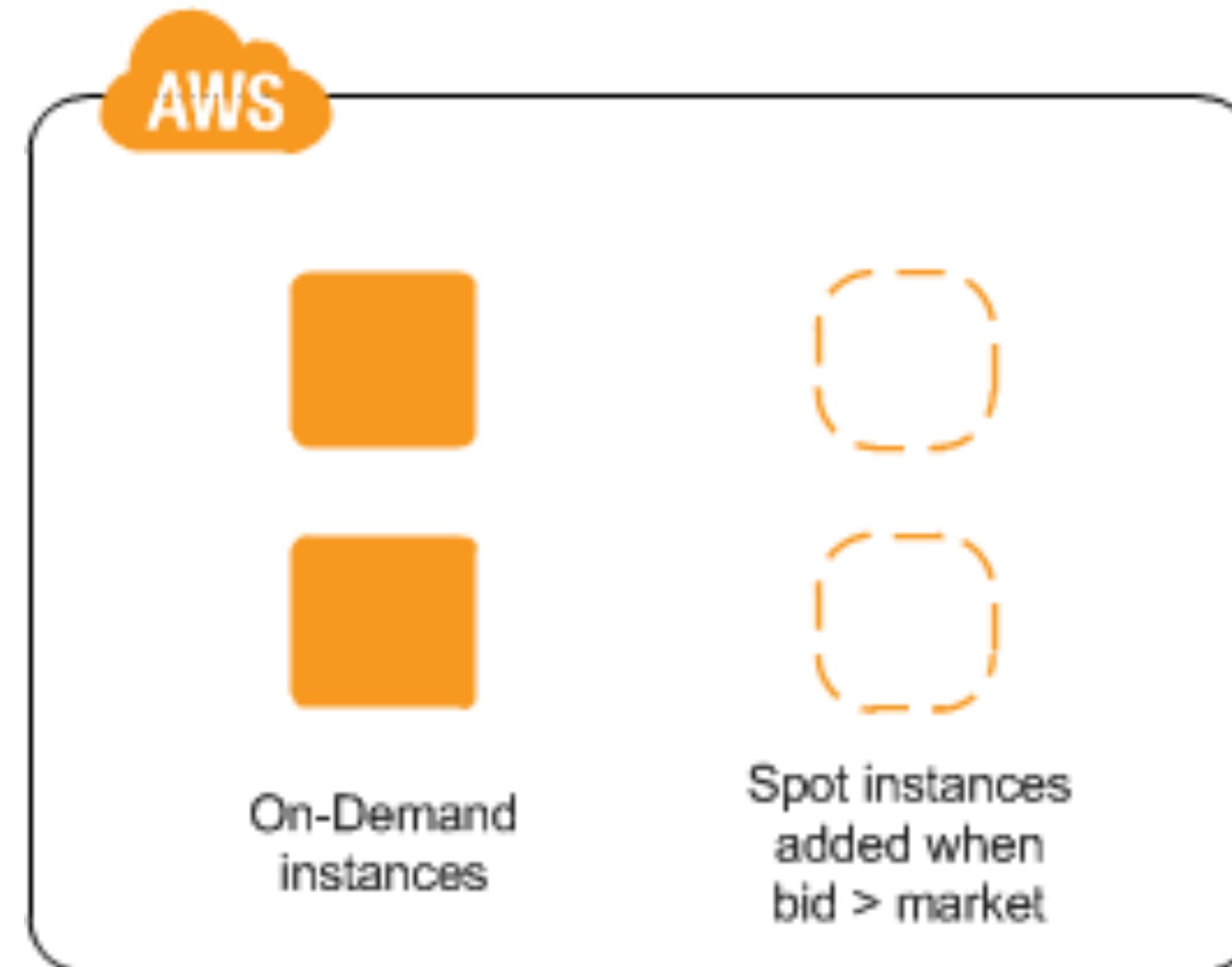
US EAST (N.Virginia) On-Demand Price

vCPU	ECU	Memory (GiB)	Instance Storage (GB)	Linux/UNIX Usage
GPU Instances - Current Generation				
p2.xlarge	4	12	61	EBS Only \$0.9 per Hour
p2.8xlarge	32	94	488	EBS Only \$7.2 per Hour
p2.16xlarge	64	188	732	EBS Only \$14.4 per Hour
g2.2xlarge	8	26	15	60 SSD \$0.65 per Hour
g2.8xlarge	32	104	60	2 x 120 SSD \$2.6 per Hour

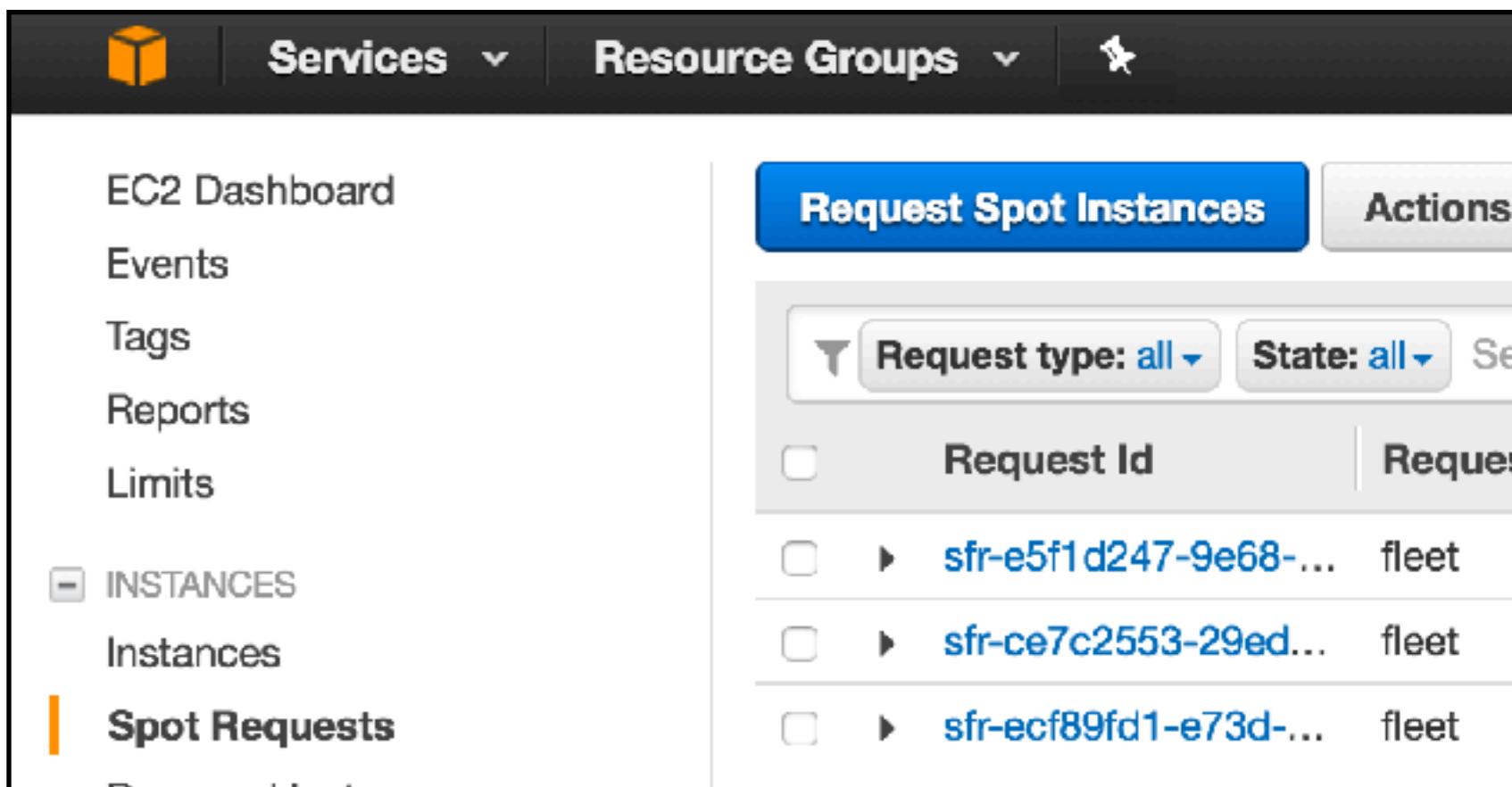
US EAST (N.Virginia) Spot Instance Price

g2.2xlarge	\$0.1564 per Hour	VS 0.6
g2.8xlarge	\$1.966 per Hour	VS 2.8
p2.xlarge	\$0.243 per Hour	VS 0.9
p2.8xlarge	\$3.583 per Hour	VS 7.2

On-Demand VS Spot Instances



Request spot instances and save \$



Select request type

Request type Request Submit a one-time Spot instance request

Request and Maintain Request a fleet of Spot instances to maintain your target capacity

Reserve for duration Request a Spot instance with no interruption for 1 to 6 hours (a Spot block)

Target capacity instances ▾

AMI Select

Instance type(s) Select multiple instance types to find the lowest priced instances available Select

Request spot instances and save \$

Select instance types x

Supports dedicated tenancy **Pricing History** **Spot Bid Advisor**

Instance type ▾	vCPUs	Memory (GiB)	Storage (GB)	Network ▾	Current spot price ▾	Spot savings ▾
GPU compute	1	(Any)	(Any)	Any network		
<input type="checkbox"/> p2.xlarge	4	61	EBS only	High	\$0.1162	87%
<input type="checkbox"/> p2.8xlarge	32	488	EBS only	10 Gigabit	\$1.3821	81%
<input type="checkbox"/> p2.16xlarge	64	732	EBS only	20 Gigabit	\$4.5648	68%

Request spot instances ready (takes 2~3 min)

Screenshot of the AWS EC2 Instances page showing two instances listed:

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv4 Public IP
	i-013b972e59561e7b3	g2.2xlarge	us-east-1a	pending	Initializing	None	ec2-52-90-17-8.compute-1.amazonaws.com	52.90.17.8
	i-0480cf73c8a91222e	g2.2xlarge	us-east-1d	terminated		None		-

Details for Instance i-013b972e59561e7b3:

Description	Value
Instance ID	i-013b972e59561e7b3
Instance state	pending
Instance type	g2.2xlarge
Elastic IPs	
Availability zone	us-east-1a
Security groups	default , view inbound rules
Scheduled events	-
AMI ID	TF0.12_CUDA8 (ami-eb31c2fd)
Public DNS (IPv4)	ec2-52-90-17-8.compute-1.amazonaws.com
IPv4 Public IP	52.90.17.8
IPv6 IPs	-
Private DNS	ip-172-31-18-62.ec2.internal
Private IPs	172.31.18.62
Secondary private IPs	
VPC ID	vpc-3e020a5b
Subnet ID	subnet-02bdccf5b

Work with screen

- ssh
- screen: open a new screen
 - python train.py; ..; echo “Done” | mail -s “Finished” hunkim@gmail.com
 - Ctrl-a d (to exit screen)
- screen -r: attach the screen

batch.ly

AWS COST
REDUCTION.
AUTOMATED.

AWS Spot Instance

**TERMINATION
NOTICES**



But spot instances can be terminated by AWS!

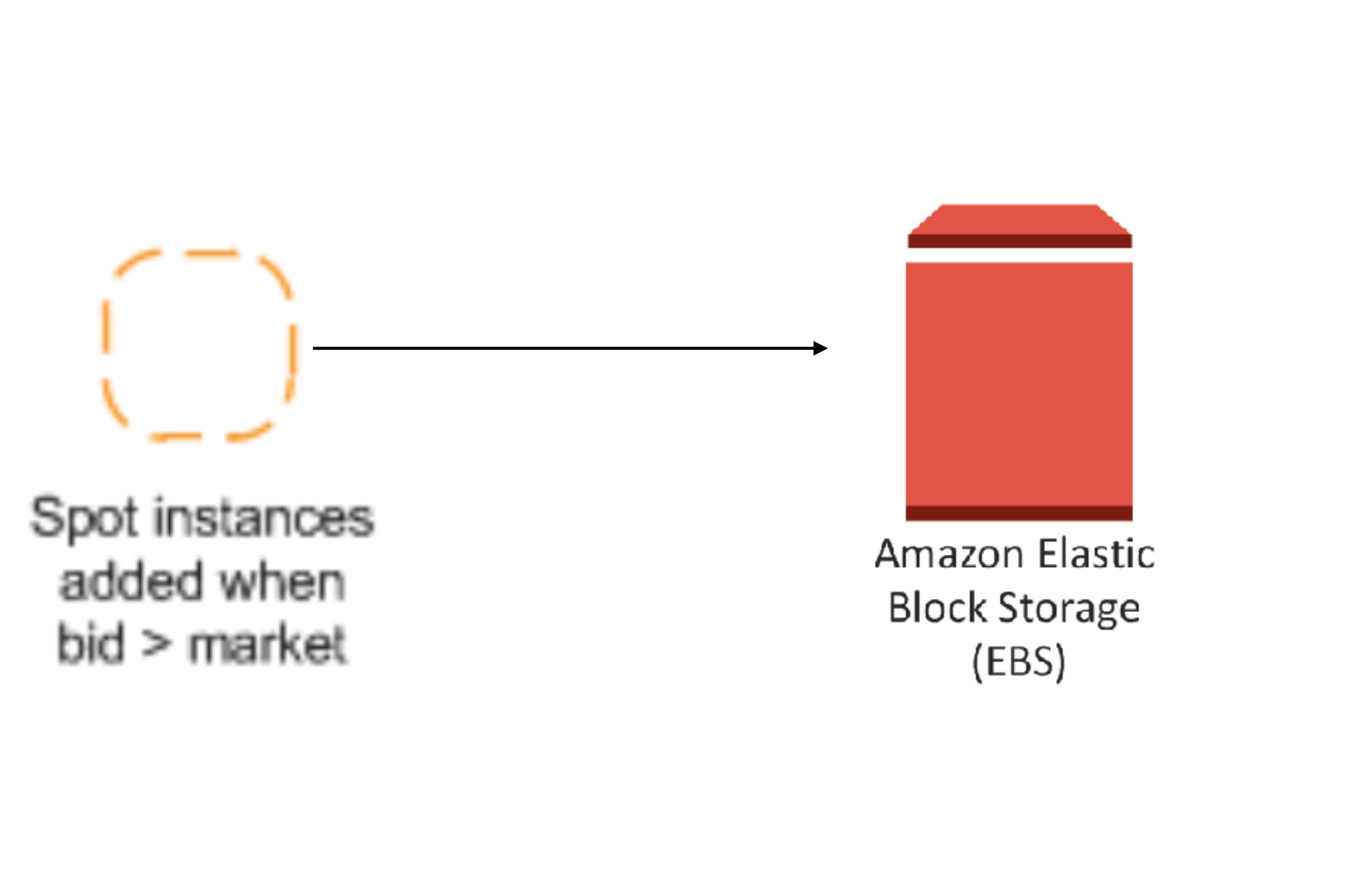
Solution: Spot Instance + EBS Volume



Spot instances
added when
 $\text{bid} > \text{market}$

Automatic termination after finishing job! (save \$)

Solution: Spot Instance + EBS Volume (do not delete on termination)



Automatic termination after finishing job! (save \$)

Don't delete the volume!

Spot instance launch wizard

[Step 1: Find instance types](#)

[Step 2: Configure](#)

[Step 3: Review](#)

Configure storage

Instance store i

Attach at launch

EBS volumes i

Device <small>i</small>	Snapshot	Size (GiB)	Volume Type	IOPS	Delete <small>i</small>
Root: /dev/xvda	snap-fe8a3c04	8	General Purpose (SSD)		<input type="checkbox"/>

No additional EBS volumes configured

[Add new volume](#)

EBS-optimized i

Launch EBS-optimized instances

Workflow

- Create a spot instance (from an AMI)
 - ***Do not delete the main disk on termination***
- Run TF tasks; sudo shutdown now
 - run (in screen) and save results on the volume
 - terminate the instance (save \$)
- Create a (1) snapshot and (2) AMI from the leftover volume
- Create a new spot using the AMI
 - Delete old AMIs

0. Create AMI - TF/Cuda

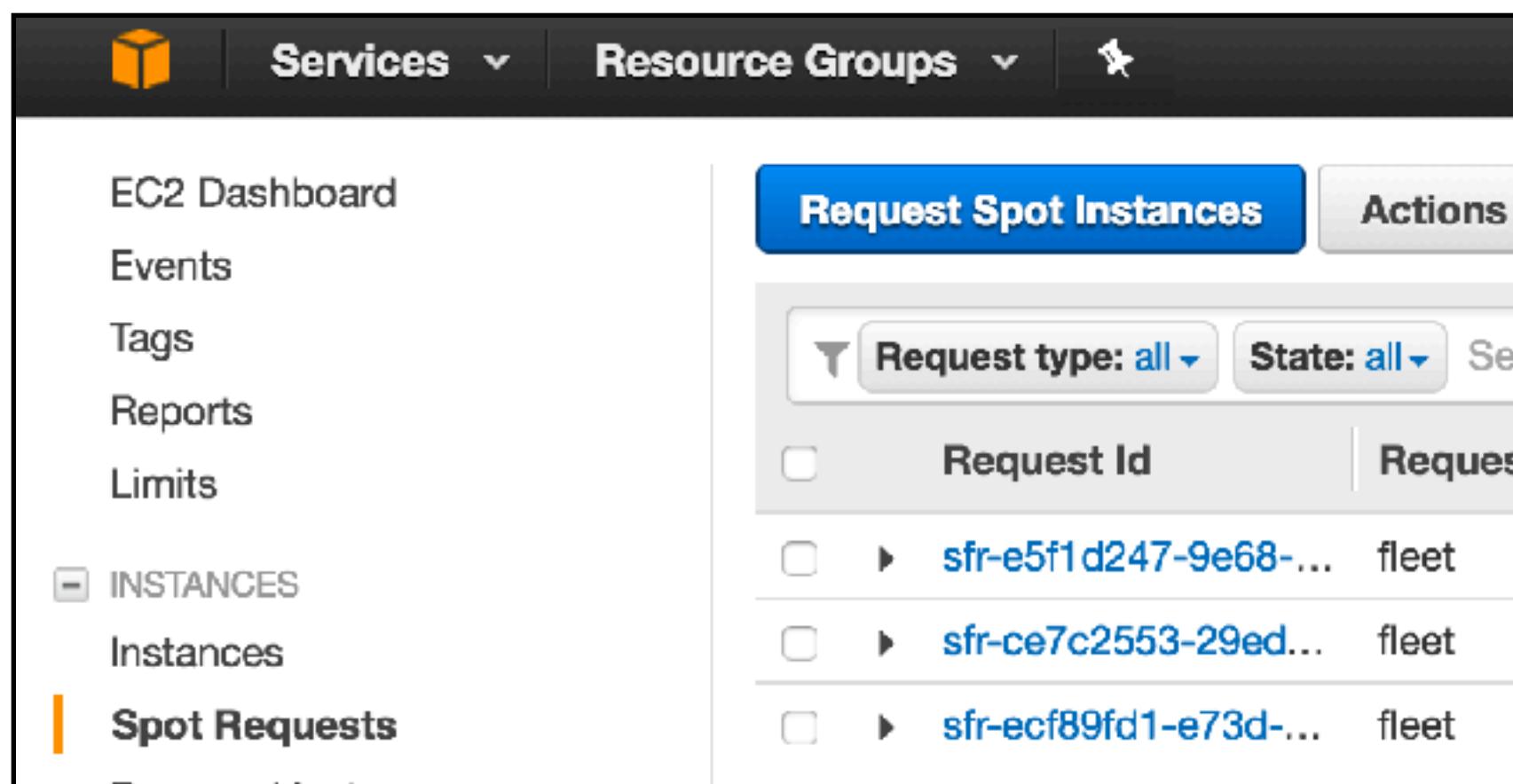
The screenshot shows the AWS EC2 Instances page. On the left, there's a sidebar with links like EC2 Dashboard, Events, Tags, Reports, Limits, INSTANCES, Instances, Spot Requests, Reserved Instances, and Dedicated Hosts. The Instances link is highlighted with an orange border. In the main area, there's a navigation bar with 'Launch Instance', 'Connect', and 'Actions'. A dropdown menu is open under 'Actions', showing options: Connect, Get Windows Password, Launch More Like This, Instance State, Instance Settings, Image (which is highlighted in orange), Networking, and CloudWatch Monitoring. To the right of the dropdown, there's a table showing instance details: Name (fgsight), Instance ID (i-0202900a...), Availability Zone (ap-northeast-2a), and Instance State (running). There's also a 'Create Image' button.

N. Virginia: ami-eb31c2fd (maybe outdated soon)

<https://github.com/ritchien/tensorflow-aws-ami>

<http://expressionflow.com/2016/10/09/installing-tensorflow-on-an-aws-ec2-p2-gpu-instance>

I. Request spot instances using AMI



Select request type

Request type **Request**
Submit a one-time Spot instance request

Request and Maintain
Request a fleet of Spot instances to maintain your target capacity

Reserve for duration
Request a Spot instance with no interruption for 1 to 6 hours (a Spot block)

Target capacity **1** instances

AMI **Amazon Linux AMI 2016.09.0.20161028 x86_64**

Instance type(s) **Select multiple instance types to find the lowest priced instances available**

I. Request spot instances using AMI

Choose an Amazon Machine Image (AMI) x

ami-eb31c2fd Community AMIs « < 1 to 1 of 1 AMIs > »

	TF0.12_CUDA8 - ami-eb31c2fd	Select
	Root device type: ebs Virtualization type: hvm Owner: 705119112097	64-bit

I. Request spot instances: check price

Select instance types x

Supports dedicated tenancy **Pricing History** **Spot Bid Advisor**

Instance type	vCPUs	Memory (GiB)	Storage (GB)	Network	Current spot price	Spot savings
GPU compute	1	(Any)	(Any)	Any network		
<input type="checkbox"/> p2.xlarge	4	61	EBS only	High	\$0.1162	87%
<input type="checkbox"/> p2.8xlarge	32	488	EBS only	10 Gigabit	\$1.3821	81%
<input type="checkbox"/> p2.16xlarge	64	732	EBS only	20 Gigabit	\$4.5648	68%

I. Request spot instances: Don't delete the volume!

Spot instance launch wizard

[Step 1: Find instance types](#)
[Step 2: Configure](#)
[Step 3: Review](#)

Configure storage

Instance store i

Attach at launch

EBS volumes i

Device <small>i</small>	Snapshot	Size (GiB)	Volume Type	IOPS	Delete <small>i</small>
Root: /dev/xvda	snap-fe8a3c04	8	General Purpose (SSD)		<input type="checkbox"/>

No additional EBS volumes configured

[Add new volume](#)

EBS-optimized i

Launch EBS-optimized instances

I. Request spot instances: Success

Network [i](#) vpc-3e020a5b (172.31.0.0/16) (default)

Availability Zones / Subnets [i](#) No preference (launch in cheapest Availability Zone)

Security groups [i](#) sg-c6c631a1

Auto-assign IPv4 Public IP [i](#) Use subnet setting (Enable)

▼ Instance details Success

Spot request with id: sfr-2224b5ea-2ff1-470f-9454-5a877ba574b7 successfully created.

[OK](#)

Root volume [i](#) /dev/sda1: 20GiB (gp2), from snapshot snap-0d3380be

EBS-optimized [i](#) no

Monitoring [i](#) no

Tenancy [i](#) default

Request valid from [i](#) Now

Request valid until [i](#) 1 year from now

Terminate instances at expiration [i](#) yes

[Cancel](#) [Previous](#) [Launch](#)

2. Run TF (with screen) and enjoy!

- screen
 - Start job: `python3py; sudo shutdown now`
 - `python3 main.py --maxLength=80 | tee out.log; sudo shutdown now`
 - Ctrl-a d (exit the screen, but Job is going)
- Checking progress: `screen -r`
 - Ctrl-a d (exit the screen)
- When the job is done, the instances will be terminated (save \$)
- But the results will be in the volume

3. Spot Instance termination + EBS Volume



Automatic termination after finishing job! (save \$)

4. Create snapshot from the volume

Screenshot of the AWS CloudFormation console showing the creation of a snapshot from a volume.

The main interface shows a list of volumes:

Name	Volume ID	Size	Volume Type	IOPS	Snapshot	Created	Availability Zone	State	Alarm Status
vol-0c74cb5...	20 GiB	gp2	100 / 3000	snap-c515822f	January 30, 2017 at...	us-east-1d	available	None	
vol-0aeebea...	20 GiB	gp2	100 / 3000	snap-c515822f	January 30, 2017 at...	us-east-1c	in-use	None	
forspot	vol...	gp2	100 / 3000	snap-c515822f	January 30, 2017 at...	us-east-1a	available	None	

A modal dialog titled "Create Snapshot" is open, showing the configuration for the new snapshot:

- Volume: vol-0c74cb5896968b1b8
- Name: snap2017-1-30
- Description: (empty)
- Encrypted: No

Buttons at the bottom of the modal are "Cancel" and "Create".

At the bottom of the page, there are navigation links for "Description" and "Status".

5.Create AMI from the snapshot

The screenshot shows the AWS EC2 Dashboard with the 'Create Snapshot' button highlighted. A table lists a single EBS snapshot:

Name	Snapshot ID	Size	Description	Status	Started	Pro
AMI-2017-1-20	snap-02ea135683e1fb71d	20 GiB		completed	January 30, 2017 at 7:48:29 ...	ava

Create Image from EBS Snapshot

Name AMI-2017-1-20 **Description**
Architecture x86_64 **Virtualization type** Paravirtual
Root device name /dev/sda1 **Kernel ID** Use default
RAM disk ID Use default

Block Device Mappings

Volume Type	Device	Snapshot	Size (GiB)	Volume Type	IOPS	Throughput (MB/s)	Delete on Termination	Encrypted
Root	/dev/sda1	snapshot-02ea135683e1fb71d	20	General Purpose SSD (GP2)	100 / 3000	N/A	<input checked="" type="checkbox"/>	Not Encrypted

Add New Volume **Create** **Cancel**

6. Create new spot instance using the AMI

The screenshot shows the AWS Management Console interface for launching a spot instance. The top navigation bar includes 'Services' (selected), 'Resource Groups', and user information for 'Sung Kim'. A modal window titled 'Choose an Amazon Machine Image (AMI)' is open, showing a list of available AMIs. The first item listed is 'AMI-2017-1-20 - ami-199a650f', which is a 64-bit paravirtualized Linux AMI. The 'Select' button is highlighted in blue at the bottom right of the modal.

Spot instance launch wizard

Step 1: Find instance types

Step 2: Configure

Step 3: Review

Choose an Amazon Machine Image (AMI)

Search by AMI ID or keyword

My AMIs

AMI-2017-1-20 - ami-199a650f

Select

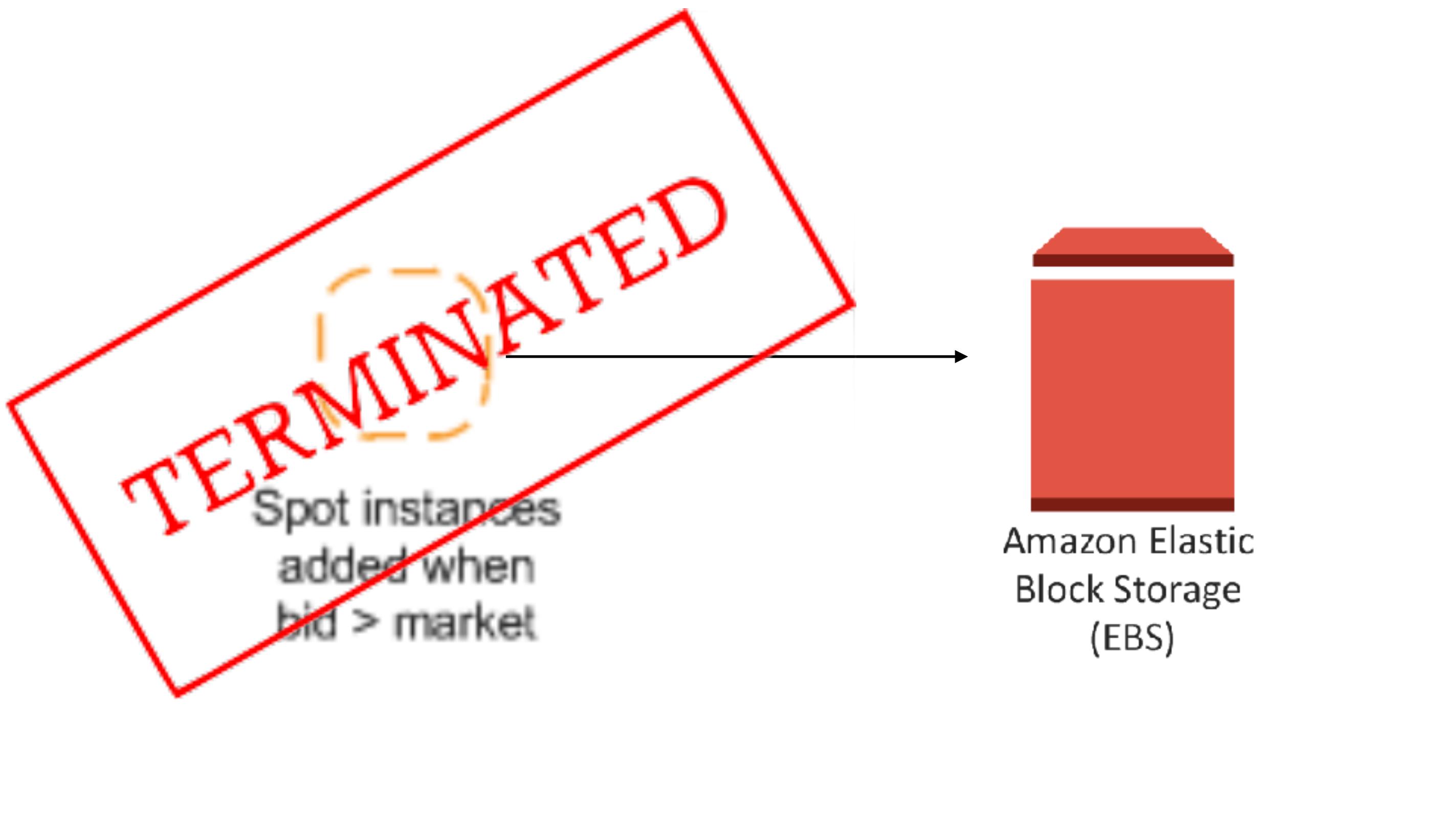
64-bit

Root device type: ebs Virtualization type: paravirtual Owner: 705119112097

GPU, GPU, GPU

```
Total memory: 11.17GiB
Free memory: 11.11GiB
I tensorflow/core/common_runtime/gpu/gpu_device.cc:906] DMA: 0 1 2 3 4 5 6 7
I tensorflow/core/common_runtime/gpu/gpu_device.cc:916] 0: Y Y Y Y Y Y Y Y
I tensorflow/core/common_runtime/gpu/gpu_device.cc:916] 1: Y Y Y Y Y Y Y Y
I tensorflow/core/common_runtime/gpu/gpu_device.cc:916] 2: Y Y Y Y Y Y Y Y
I tensorflow/core/common_runtime/gpu/gpu_device.cc:916] 3: Y Y Y Y Y Y Y Y
I tensorflow/core/common_runtime/gpu/gpu_device.cc:916] 4: Y Y Y Y Y Y Y Y
I tensorflow/core/common_runtime/gpu/gpu_device.cc:916] 5: Y Y Y Y Y Y Y Y
I tensorflow/core/common_runtime/gpu/gpu_device.cc:916] 6: Y Y Y Y Y Y Y Y
I tensorflow/core/common_runtime/gpu/gpu_device.cc:916] 7: Y Y Y Y Y Y Y Y
I tensorflow/core/common_runtime/gpu/gpu_device.cc:975] Creating TensorFlow device (/gpu:0) -> (device: 0, name: Tesla K80, pci bus id: 0000:00:17.0)
I tensorflow/core/common_runtime/gpu/gpu_device.cc:975] Creating TensorFlow device (/gpu:1) -> (device: 1, name: Tesla K80, pci bus id: 0000:00:18.0)
I tensorflow/core/common_runtime/gpu/gpu_device.cc:975] Creating TensorFlow device (/gpu:2) -> (device: 2, name: Tesla K80, pci bus id: 0000:00:19.0)
I tensorflow/core/common_runtime/gpu/gpu_device.cc:975] Creating TensorFlow device (/gpu:3) -> (device: 3, name: Tesla K80, pci bus id: 0000:00:1a.0)
I tensorflow/core/common_runtime/gpu/gpu_device.cc:975] Creating TensorFlow device (/gpu:4) -> (device: 4, name: Tesla K80, pci bus id: 0000:00:1b.0)
I tensorflow/core/common_runtime/gpu/gpu_device.cc:975] Creating TensorFlow device (/gpu:5) -> (device: 5, name: Tesla K80, pci bus id: 0000:00:1c.0)
I tensorflow/core/common_runtime/gpu/gpu_device.cc:975] Creating TensorFlow device (/gpu:6) -> (device: 6, name: Tesla K80, pci bus id: 0000:00:1d.0)
I tensorflow/core/common_runtime/gpu/gpu_device.cc:975] Creating TensorFlow device (/gpu:7) -> (device: 7, name: Tesla K80, pci bus id: 0000:00:1e.0)
```

Solution: Spot Instance + EBS Volume



Automatic termination after finishing job! (save \$)

TensorFlow-KR 회원을 위한 AWS 크레딧 제공!



<http://bit.ly/awskr-feedback>

딥러닝 연구 및 학습
AWS 100달러 무료 크레딧 제공

등록하시면 패키지를 받으실 수 있는 URL 및 AWS 학습 정보를 이메일로 보내드립니다!