PURDUE
ENGINEERING

TensorFlow Tutorial

Diyu Yang (PhD, Year 1)

Slides are originally prepared by Bharath Ramsundar (Stanford)

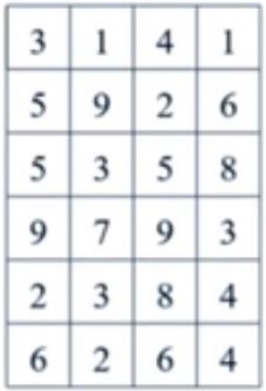
What is TensorFlow?

- TensorFlow is a deep learning library recently open-sourced by Google.
- But what does it actually do?
  - TensorFlow provides primitives for defining functions on tensors and automatically computing their derivatives.

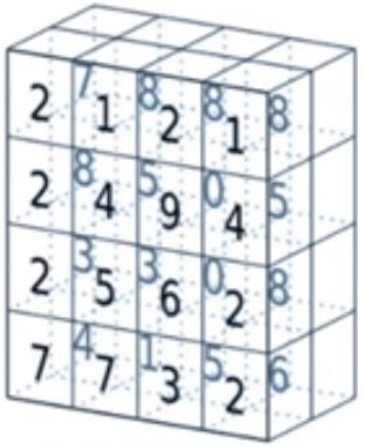
But what's a Tensor?

- Tensors are the standard way of representing data in Tensorflow (deep learning)
- Tensors are multidimensional arrays, an extension of matrices to data with higher dimensions

Tensor of dimension[1]



Tensor of dimensions[2]



Tensor of dimensions[3]

TensorFlow vs. Numpy

- Few people make this comparison, but TensorFlow and Numpy are quite similar. (Both are N-d array libraries!)
- Numpy has Ndarray support, but doesn't offer methods to create tensor functions and automatically compute derivatives (+ no GPU support).

VS

Simple Numpy Recap

```
In [23]: import numpy as np

In [24]: a = np.zeros((2,2)); b = np.ones((2,2))

In [25]: np.sum(b, axis=1)
Out[25]: array([ 2.,  2.])

In [26]: a.shape
Out[26]: (2, 2)

In [27]: np.reshape(a, (1,4))
Out[27]: array([[ 0.,  0.,  0.,  0.]])
```

Repeat in TensorFlow

```
In [31]: import tensorflow as tf

In [32]: tf.InteractiveSession()

In [33]: a = tf.zeros((2,2)); b = tf.ones((2,2))

In [34]: tf.reduce_sum(b, reduction_indices=1).eval()
Out[34]: array([ 2.,  2.], dtype=float32)

In [35]: a.get_shape()
Out[35]: TensorShape([Dimension(2), Dimension(2)])

In [36]: tf.reshape(a, (1, 4)).eval()
Out[36]: array([[ 0.,  0.,  0.,  0.]], dtype=float32)
```

More on `Session` soon

More on `.eval()` in a few slides

`TensorShape` behaves like a python tuple.

Numpy to TensorFlow Dictionary

Numpy	TensorFlow
`a = np.zeros((2,2)); b = np.ones((2,2))`	`a = tf.zeros((2,2)), b = tf.ones((2,2))`
`np.sum(b, axis=1)`	`tf.reduce_sum(a,reduction_indices=[1])`
`a.shape`	`a.get_shape()`
`np.reshape(a, (1,4))`	`tf.reshape(a, (1,4))`
`b * 5 + 1`	`b * 5 + 1`
`np.dot(a,b)`	`tf.matmul(a, b)`
`a[0,0], a[:,0], a[0,:]`	`a[0,0], a[:,0], a[0,:]`

TensorFlow requires explicit evaluation!

```
In [37]: a = np.zeros((2,2))

In [38]: ta = tf.zeros((2,2))

In [39]: print(a)
[[ 0.  0.]
 [ 0.  0.]]

In [40]: print(ta)
Tensor("zeros_1:0", shape=(2, 2), dtype=float32)

In [41]: print(ta.eval())
[[ 0.  0.]
 [ 0.  0.]]
```

*TensorFlow computations define a **computation graph** that has no numerical value until evaluated!*

TensorFlow Session Object (1)

- “A Session object encapsulates the environment in which Tensor objects are evaluated” - TensorFlow Docs

```
In [20]: a = tf.constant(5.0)

In [21]: b = tf.constant(6.0)

In [22]: c = a * b

In [23]: with tf.Session() as sess:
   ....:     print(sess.run(c))
   ....:     print(c.eval())
   ....:
30.0
30.0
```

`c.eval()` is just syntactic sugar for `sess.run(c)` in the currently active session!

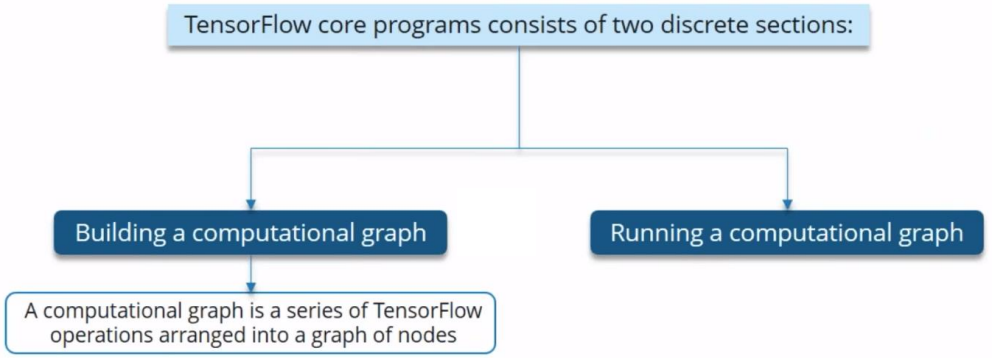
TensorFlow Session Object (2)

- `tf.InteractiveSession()` is just convenient syntactic sugar for keeping a default session open in ipython.
- `sess.run(c)` is an example of a TensorFlow *Fetch*. Will say more on this soon.

Tensorflow Computation Graph

- “TensorFlow programs are usually structured into a construction phase, that assembles a graph, and an execution phase that uses a session to execute ops in the graph.” - TensorFlow docs
- All computations add nodes to global default graph (docs)

Tensorflow Computation Graph



Tensorflow Structure

```
import tensorflow as tf

a = tf.constant(5.0,tf.float32)
b = tf.constant(6.0)
c = a*b
```

Build a computational graph

```
print(sess.run(c))
sess.close()
```

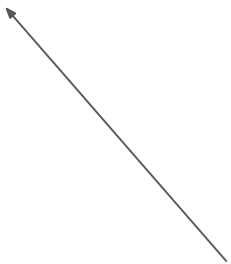
Run the computational graph

TensorFlow Variables (1)

- “When you train a model you use variables to hold and update parameters. Variables are in-memory buffers containing tensors” - TensorFlow Docs.
- All tensors we’ve used previously have been *constant* tensors, not variables.

TensorFlow Variables (2)

```
In [32]: W1 = tf.ones((2,2))

In [33]: W2 = tf.Variable(tf.zeros((2,2)), name="weights")

In [34]: with tf.Session() as sess:
             print(sess.run(W1))
             sess.run(tf.global_variables_initializer())
             print(sess.run(W2))
   ....:
[[ 1.  1.]
 [ 1.  1.]]
[[ 0.  0.]
 [ 0.  0.]]
```



Note the initialization step
`tf.global_variables_initializer()`

TensorFlow Variables (3)

- TensorFlow variables must be initialized before they have values! Contrast with constant tensors.

```
In [38]: W = tf.Variable(tf.zeros((2,2)), name="weights")

In [39]: R = tf.Variable(tf.random_normal((2,2)), name="random_weights")

In [40]: with tf.Session() as sess:
    ....:     sess.run(tf.global_variables_initializer())
    ....:     print(sess.run(W))
    ....:     print(sess.run(R))
    ....:
```

`Variable` objects can be initialized from constants or random values

Initializes all variables with specified values.

Updating Variable State

```
In [63]: state = tf.Variable(0, name="counter")

In [64]: new_value = tf.add(state, tf.constant(1))

In [65]: update = tf.assign(state, new_value)

In [66]: with tf.Session() as sess:
    ....:     sess.run(tf.global_variables_initializer())
    ....:     print(sess.run(state))
    ....:     for _ in range(3):
    ....:         sess.run(update)
    ....:         print(sess.run(state))
    ....:
0
1
2
3
```

Roughly `new_value = state + 1`

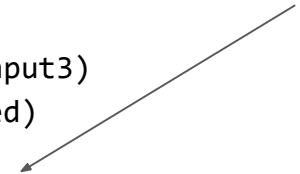
Roughly `state = new_value`

Roughly

```
state = 0
print(state)
for _ in range(3):
  state = state + 1
  print(state)
```

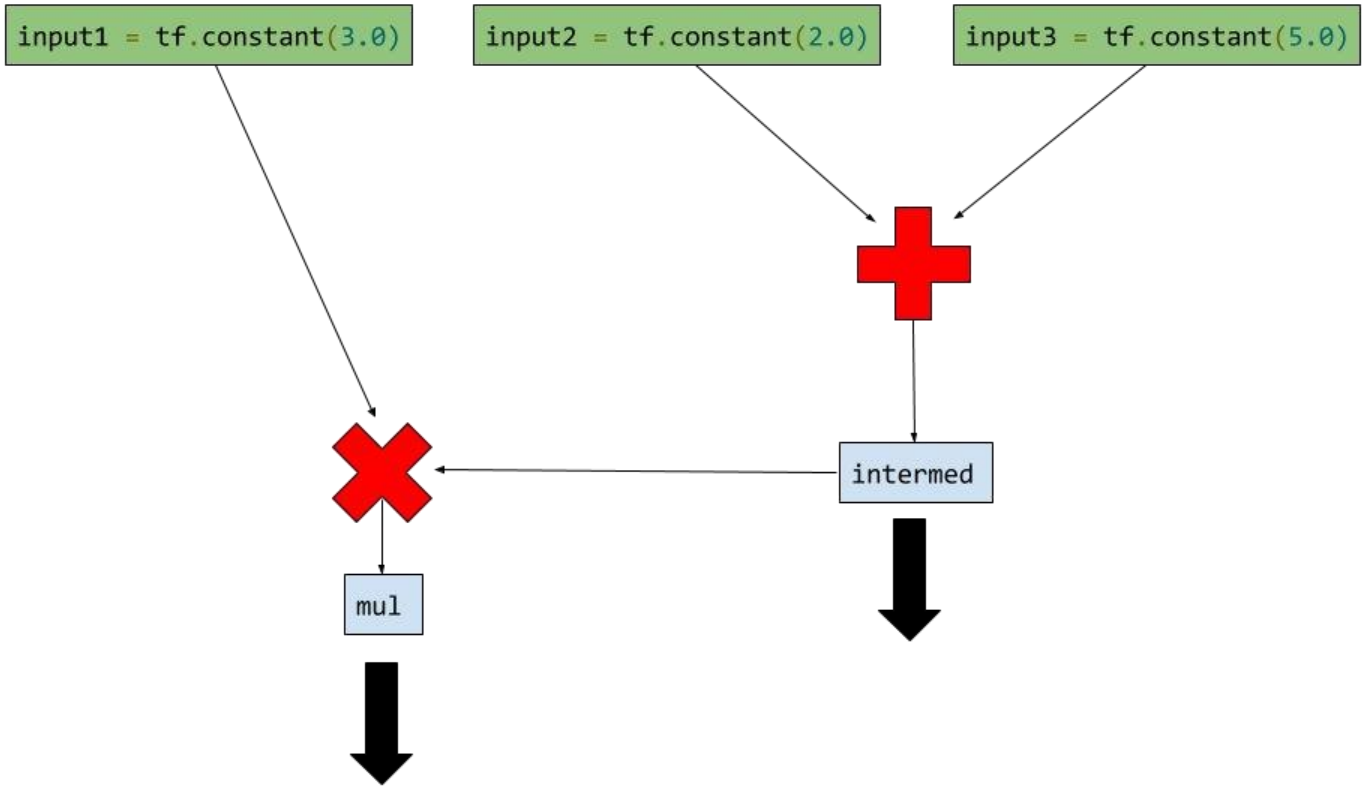
Fetching Variable State (1)

```
In [82]: input1 = tf.constant(3.0)
In [83]: input2 = tf.constant(2.0)
In [84]: input3 = tf.constant(5.0)
In [85]: intermed = tf.add(input2, input3)
In [86]: mul = tf.mul(input1, intermed)
In [87]: with tf.Session() as sess:
   ....:       result = sess.run([mul, intermed])
   ....:       print(result)
   ....:
[21.0, 7.0]
```



Calling `sess.run(var)` on a `tf.Session()` object retrieves its value. Can retrieve multiple variables simultaneously with `sess.run([var1, var2])` (See *Fetches* in TF docs)

Fetching Variable State (2)



Exercise 1: Gradient Descent with TensorFlow

Minimizing the following quadratic function:

$$f(x) = \frac{1}{2}x^T Q x + x^T b$$

$$\text{where } Q = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \text{ is positive definite, and } b = \begin{bmatrix} 3 \\ 3 \end{bmatrix}.$$

$$\text{Analytical solution: } x* = -Q^{-1}b = \begin{bmatrix} -1 \\ -1 \end{bmatrix}.$$

Let's try to verify it using gradient descent in tensorflow:

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(f)
```

This function, when evaluated, will automatically update the variable state for you.

Skeleton code is in exp1.py. Let's get our hands dirty and start programming :)

Inputting Data

- All previous examples have manually defined tensors. How can we input external data into TensorFlow?
- Simple solution: Import from Numpy:

```
In [93]: a = np.zeros((3,3))
In [94]: ta = tf.convert_to_tensor(a)
In [95]: with tf.Session() as sess:
   ....:     print(sess.run(ta))
   ....:
[[ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]]
```

Placeholders and Feed Dictionaries (1)

- Inputting data with `tf.convert_to_tensor()` is convenient, but doesn't scale.
- Use `tf.placeholder` variables (dummy nodes that provide entry points for data to computational graph).
- A `feed_dict` is a python dictionary mapping from `tf.placeholder` vars (or their names) to data (numpy arrays, lists, etc.).

Placeholders and Feed Dictionaries (2)

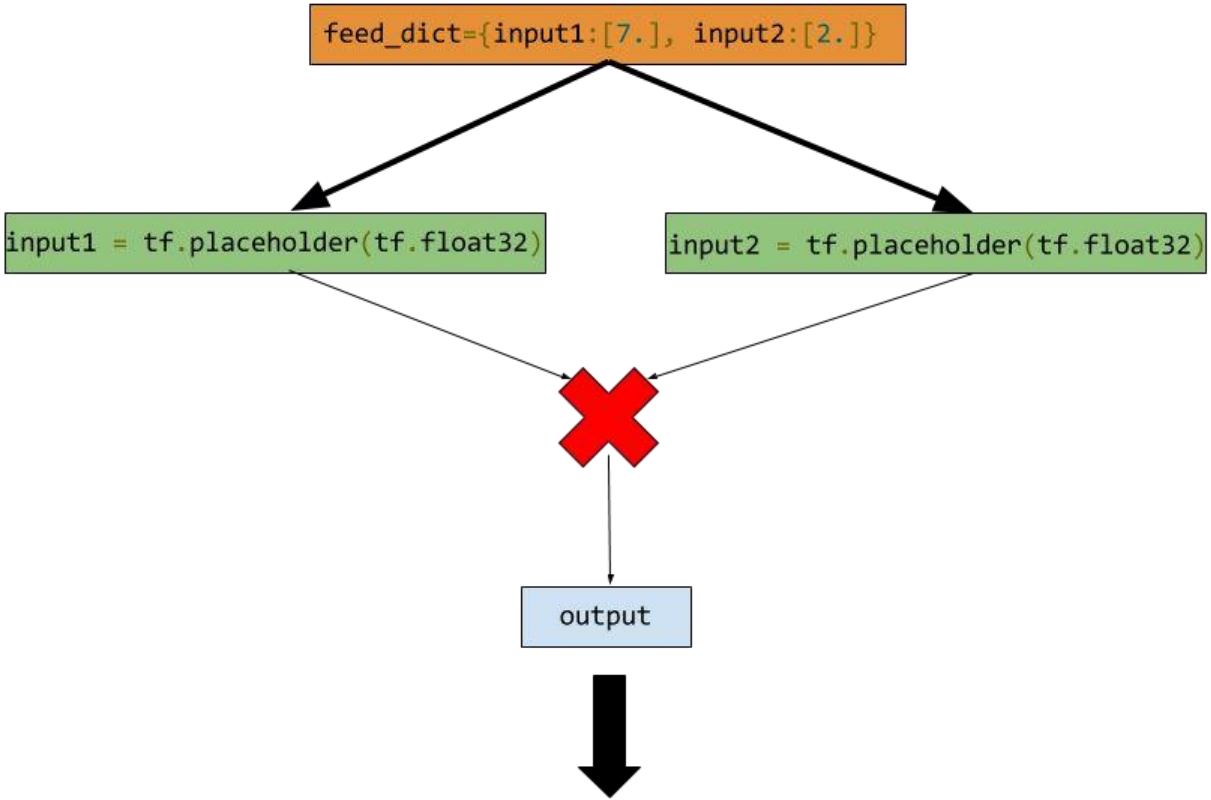
```
In [96]: input1 = tf.placeholder(tf.float32)

In [97]: input2 = tf.placeholder(tf.float32)

In [98]: output = tf.mul(input1, input2)

In [99]: with tf.Session() as sess:
   ....:       print(sess.run([output], feed_dict={input1:[7.], input2:[2.]}))
   ....:

[array([ 14.], dtype=float32)]
```

Define `tf.placeholder` *objects for data entry.*

Fetch value of output from computation graph.

Feed data into computation graph.

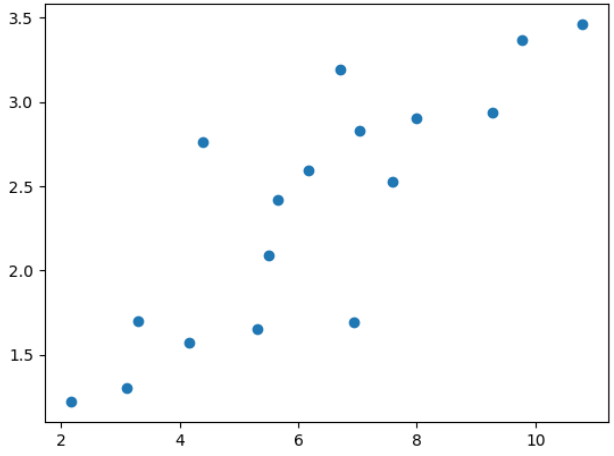
Placeholders and Feed Dictionaries (3)



Ex: Linear Regression in TensorFlow (1)

Data Preprocessing

```
# Import libraries
import tensorflow as tf
import numpy
import matplotlib.pyplot as plt
rng = numpy.random

# Training Data
train_X = numpy.array([3.3,4.4,5.5,6.71,6.93,4.168,9.779,6.182,7.59,2.167,
                       7.042,10.791,5.313,7.997,5.654,9.27,3.1])
train_Y = numpy.array([1.7,2.76,2.09,3.19,1.694,1.573,3.366,2.596,2.53,1.221,
                       2.827,3.465,1.65,2.904,2.42,2.94,1.3])
n_samples = train_X.shape[0]
print('length of input data: ',n_samples)


# Plot training data
plt.scatter(train_X, train_Y)
plt.show()
```



Ex: Linear Regression in TensorFlow (2)

Build a computation graph

```
# Parameters
learning_rate = 0.01
training_epochs = 1000
display_step = 50

# placeholder for Input
X = tf.placeholder("float")
Y = tf.placeholder("float")

# Set model weights as variables
W = tf.Variable(rng.randn(), name="weight")
b = tf.Variable(rng.randn(), name="bias")

# Construct a linear model
pred = tf.add(tf.multiply(X, W), b)
```

Ex: Linear Regression in TensorFlow (3)

Build a computation graph: Loss function, Optimizer

```
# Mean squared error
cost = tf.reduce_sum(tf.pow(pred-Y, 2))/n_samples
# Gradient descent
#  Note, minimize() knows to modify W and b because Variable objects are
trainable=True by default
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)


# Initialize the variables (i.e. assign their default value)
init = tf.global_variables_initializer()
```

$$J(W,b) = \frac{1}{N}\sum_{i=1}^{N}(y_i - (Wx_i + b))^2$$

Ex: Linear Regression in TensorFlow (4)

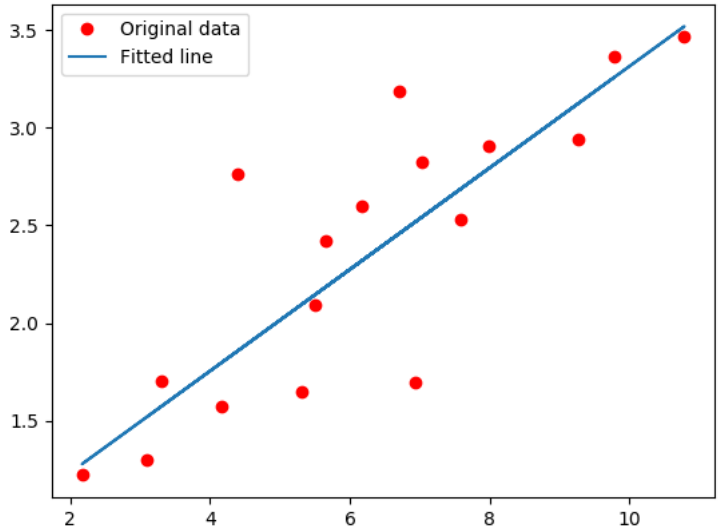
Training: Running the computation graph with user specified inputs

```
# Start training
with tf.Session() as sess:

    # Run the initializer
    sess.run(init)

    # Fit all training data
    for epoch in range(training_epochs):
        for (x, y) in zip(train_X, train_Y):
            sess.run(optimizer, feed_dict={X: x, Y: y})

        # Display logs per epoch step
        if (epoch+1) % display_step == 0:
            c = sess.run(cost, feed_dict={X: train_X, Y:train_Y})
            print("Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}".format(c), \
                "W=", sess.run(W), "b=", sess.run(b))


    print("Optimization Finished!")
    training_cost = sess.run(cost, feed_dict={X: train_X, Y: train_Y})
    print("Training cost=", training_cost, "W=", sess.run(W), "b=", sess.run(b), '\n')
```

Ex: Linear Regression in TensorFlow (5)

Training output

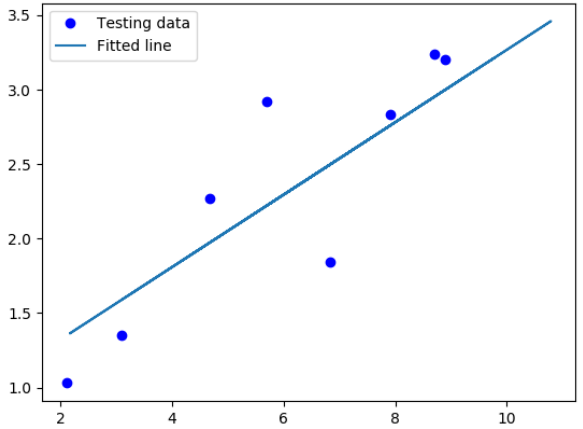
Epoch: 0050 cost= 0.264935046 W= 0.37721756 b= -0.14037561
Epoch: 0100 cost= 0.240656435 W= 0.3621354 b= -0.030448152
Epoch: 0150 cost= 0.221704662 W= 0.3488091 b= 0.06668142
Epoch: 0200 cost= 0.206911236 W= 0.3370343 b= 0.15250306
Epoch: 0250 cost= 0.195364133 W= 0.32663032 b= 0.22833313
Epoch: 0300 cost= 0.186351120 W= 0.31743762 b= 0.2953348
Epoch: 0350 cost= 0.179316282 W= 0.30931517 b= 0.35453597
Epoch: 0400 cost= 0.173825666 W= 0.3021383 b= 0.4068448
Epoch: 0450 cost= 0.169540361 W= 0.29579702 b= 0.4530639
Epoch: 0500 cost= 0.166195989 W= 0.29019395 b= 0.49390215
Epoch: 0550 cost= 0.163586035 W= 0.2852433 b= 0.5299855
Epoch: 0600 cost= 0.161549360 W= 0.28086883 b= 0.56186867
Epoch: 0650 cost= 0.159960091 W= 0.27700385 b= 0.5900391
Epoch: 0700 cost= 0.158720046 W= 0.2735886 b= 0.6149312
Epoch: 0750 cost= 0.157752573 W= 0.27057117 b= 0.6369243
Epoch: 0800 cost= 0.156997830 W= 0.2679051 b= 0.65635645
Epoch: 0850 cost= 0.156409115 W= 0.26554918 b= 0.67352724
Epoch: 0900 cost= 0.155949891 W= 0.26346764 b= 0.68869877
Epoch: 0950 cost= 0.155591801 W= 0.26162836 b= 0.70210433
Epoch: 1000 cost= 0.155312538 W= 0.26000333 b= 0.71394837
Optimization Finished!
Training cost= 0.15531254 W= 0.26000333 b= 0.71394837



Ex: Linear Regression in TensorFlow (6)

Testing on new data points

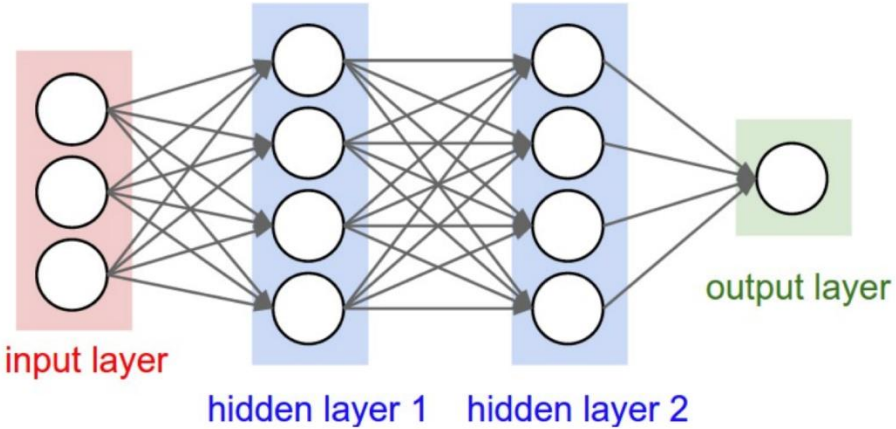
```
# Testing example (note that we are still inside tensorflow session)
test_X = numpy.asarray([6.83, 4.668, 8.9, 7.91, 5.7, 8.7, 3.1, 2.1])
test_Y = numpy.asarray([1.84, 2.273, 3.2, 2.831, 2.92, 3.24, 1.35, 1.03])


print("Testing... (Mean square loss Comparison)")
testing_cost = sess.run(
    tf.reduce_sum(tf.pow(pred - Y, 2)) / (2 * test_X.shape[0]),
    feed_dict={X: test_X, Y: test_Y})  # same function as cost above
print("Testing cost=", testing_cost)
print("Absolute mean square loss difference:", abs(
    training_cost - testing_cost))
# plot testing results
plt.plot(test_X, test_Y, 'bo', label='Testing data')
plt.plot(train_X, sess.run(W) * train_X + sess.run(b), label='Fitted line')
plt.legend()
plt.show()
```



Testing cost= 0.16192143

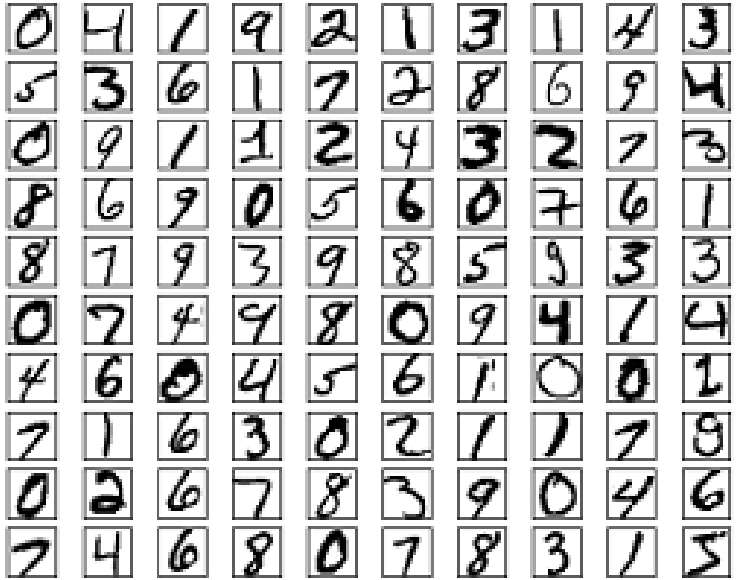
Ex: Linear Regression for Image Processing

Model

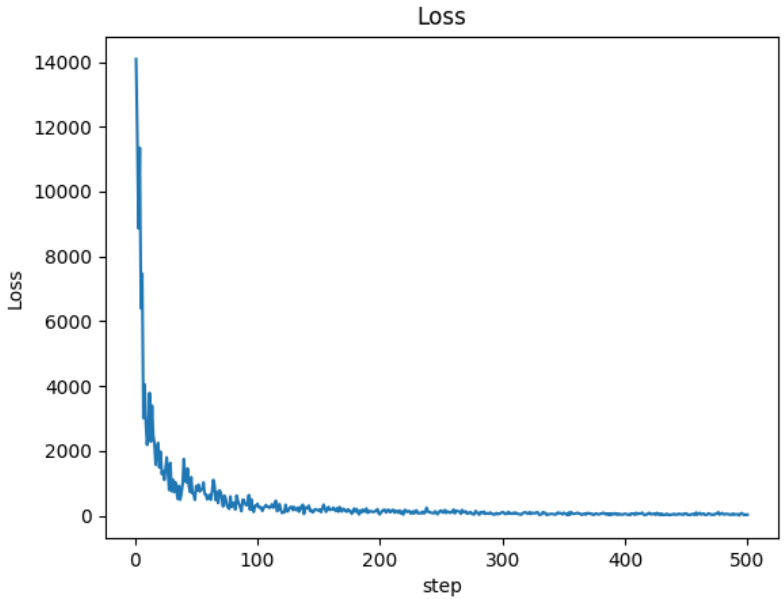


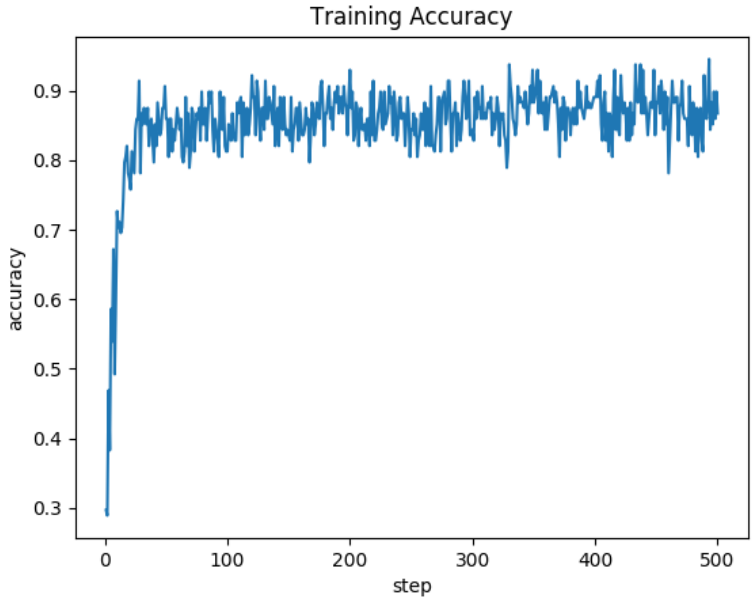
X $\quad L_1 = w_1 X + b_1 \quad L_2 = w_2 L_1 + b_1 \quad Y = w_{out} L_2 + b_{out}$

MNIST Dataset



Ex: Linear Regression for Image Processing





References

- Official python API guide for TensorFlow: https://www.tensorflow.org/api_guides/python/
- More TensorFlow examples: https://github.com/aymericdamien/TensorFlow-Examples
- This Tutorial: https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf