# TENSORFLOW

Introduction Session

# Choice of Toolkit

- Model specification:
  - Configuration file (e.g. Caffe, DistBelief, Microsoft CNTK) versus programmatic generation (e.g. Torch, Theano, Tensorflow)
- Choice of high-level language:
  - Lua (Torch) vs. Python (Theano, Tensorflow) vs C++(Caffe/Tensorflow)
- People work with python because of rich community and library infrastructure
  - Shift of Torch to PyTorch, caffe libraries in python

# Why Tensorflow

- Visualization (TensorBoard is da bomb)
- Checkpoints (for managing experiments)
- Developed and maintained by Google Brain
- Auto-differentiation autodiff (no more taking derivatives by hand. Yay)
- Used by Google, OpenAI, DeepMind, Snapchat, Uber, Airbus, eBay, Dropbox, etc

# How Tensorflow

- “Tensors”: multi dimensional arrays
  - They flow thru a “computational graph”. Duh, that’s why the name
- 2 stages of writing TF code
  - Building the graph
  - Running the graph

# How Tensorflow

- “Tensors”: multi dimensional arrays
  - They flow thru a “computational graph”. Duh, that’s why the name
- 2 stages of writing TF code
  - Building the graph
  - Running the graph
- Most Important Point when trying first time: Above Point

# How Tensorflow

- “Tensors”: multi dimensional arrays
  - They flow thru a “computational graph”. Duh, that’s why the name
- 2 stages of writing TF code
  - Building the graph
  - Running the graph
- Most Important Point when trying first time: Above Point
- Most Important Point every other time: Two points above

# What is TensorFlow ?

TensorFlow is a way of representing computation without actually performing it until asked

- Avoids doing a lot of unnecessary computation
- Allows good parallelization of computation
- Handles on-line data (non stop) pretty well
- Executes only relevant sub-graphs (auto-generated) based on need

# Numpy vs TensorFlow

(Source: Stanford Course)

# 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

More on `Session` soon

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

*`TensorShape` behaves like a python tuple.*

```
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)
```

# Common Numpy Functions

```
np.zeros()
np.ones()
np.linspace()
np.range()
np.mul()
np.mean()
np.sum()
```

# Common Numpy Functions

And their TF equivalents

```
np.zeros()      ⇒tf.zeros()
np.ones()       ⇒tf.ones()
np.linspace()   ⇒tf.linspace()
np.range()      ⇒tf.range()
np.mul()        ⇒tf.mul()
np.mean()       ⇒tf.reduce_mean()
np.sum()        ⇒tf.reduce_sum()
```

# Common Numpy Functions

And their TF equivalents

More or less have same syntax

```
np.zeros()      ⇒tf.zeros()
np.ones()       ⇒tf.ones()
np.linspace()   ⇒tf.linspace()
np.range()      ⇒tf.range()
np.mul()        ⇒tf.mul()
np.mean()       ⇒tf.reduce_mean()
np.sum()        ⇒tf.reduce_sum()
```

# TF vs np

- TensorFlow integrates seamlessly with NumPy
    - tf.int32 == np.int32 # True
- Can pass numpy types to TensorFlow ops
    - tf.ones([2, 2], np.float32) # ⇒ [[1.0 1.0], [1.0 1.0]]
    - Exception: np.float128 doesn’t exist in tensorflow, as people normally use lower precision floats for speed up
- For tf.Session.run(fetches): If the requested fetch is a Tensor, then the output of will be a NumPy ndarray.

# Data Structures in TensorFlow

# Data structures in TensorFlow

- tf.constant: Inputs as pre-defined constants
- tf.Placeholder: Inputs of variable size
- tf.Variable: Variables of fixed size

# Data structures in TensorFlow

- tf.constant: Inputs as pre-defined constants
- tf.Placeholder: Inputs of variable size
- tf.Variable: Variables of fixed size

Often, you will feed in the dataset finally when executing the graph. The dataset can contain inputs of a previously undefined size. A placeholder is a promise to provide a value as an input later.

# Data structures in TensorFlow

- tf.constant: Inputs as pre-defined constants
- tf.Placeholder: Inputs of variable size
- tf.Variable: Variables of fixed size

To make the model trainable, we need to be able to modify the graph to get new outputs with the same input. Variables allow us to add trainable parameters to a graph.
Ex: Weights and Biases of NN

# Data structures in TensorFlow

- tf.constant: Inputs as pre-defined constants
- tf.Placeholder: Inputs of variable size
- tf.Variable: Variables of fixed size

Constants: initialized in definition (a = tf.constant())

Variables: Not initialized when you call tf.Variable.
To initialize all the variables, run

```
>> init = tf.global_variables_initializer()
>> sess.run(init)
```

# Example Code: (run it)

```
import tensorflow as tf

x1 = tf.constant(35)
x2 = tf.constant(5)
y = tf.add(x1,x2)

print(y)
```

# Example Code:
# (run it)

## How do we get the output then?

```
import tensorflow as tf

x1 = tf.constant(35)
x2 = tf.constant(5)
y = tf.add(x1,x2)

print(y)
```

# How do we get the output then?

We only built the graph.
We didn't execute it

```
import tensorflow as tf

x1 = tf.constant(35)
x2 = tf.constant(5)
y = tf.add(x1,x2)

model =
tf.global_variables_initializer()

with tf.Session() as sess:
    sess.run(model)
    print(sess.run(y))
```

# Running a graph

- Two general methods:
  - sess.run()
  - <variable_name>.eval()

- Both do the same thing. Reason for using run() over eval: You can evaluate multiple variables at the same time (Do sess.run([a,b,c,..])

# tf.Variable.assign()

```
W = tf.Variable(10)
W.assign(100)
with tf.Session() as sess:
     sess.run(W.initializer)
     print W.eval() # >> ????
```

What will the output be?

# tf.Variable.assign()

```
W = tf.Variable(10)
W.assign(100)
with tf.Session() as sess:
     sess.run(W.initializer)
     print W.eval() # >> 10
```

Woah. What the ...?

# tf.Variable.assign()

```
W = tf.Variable(10)
W.assign(100)
with tf.Session() as sess:
     sess.run(W.initializer)
     print W.eval() # >> 10
```

W.assign(100) doesn’t assign the value 100 to W. It creates an assign op, and that op needs to be run to take effect.

# tf.Variable.assign()

```
W = tf.Variable(10)
W.assign(100)
with tf.Session() as sess:
     sess.run(W.initializer)
     print W.eval() # >> 10

------------------------------

W = tf.Variable(10)
assign_op = W.assign(100)
with tf.Session() as sess:
     sess.run(W.initializer)
     sess.run(assign_op)
     print W.eval() # >> 100
```

# tf.Variable.assign()

```
W = tf.Variable(10)
W.assign(100)
with tf.Session() as sess:
     sess.run(W.initializer)
     print W.eval() # >> 10
```

You don’t need to initialize variable because assign_op does it for you

```
------------------------------

W = tf.Variable(10)                  |
assign_op = W.assign(100)            |
with tf.Session() as sess:           |
     sess.run(W.initializer)         |
     sess.run(assign_op)             |
     print W.eval() # >> 100         |
```

```
W = tf.Variable(10)
assign_op = W.assign(100)
with tf.Session() as sess:
     sess.run(assign_op)
     print W.eval() # >> 100
```

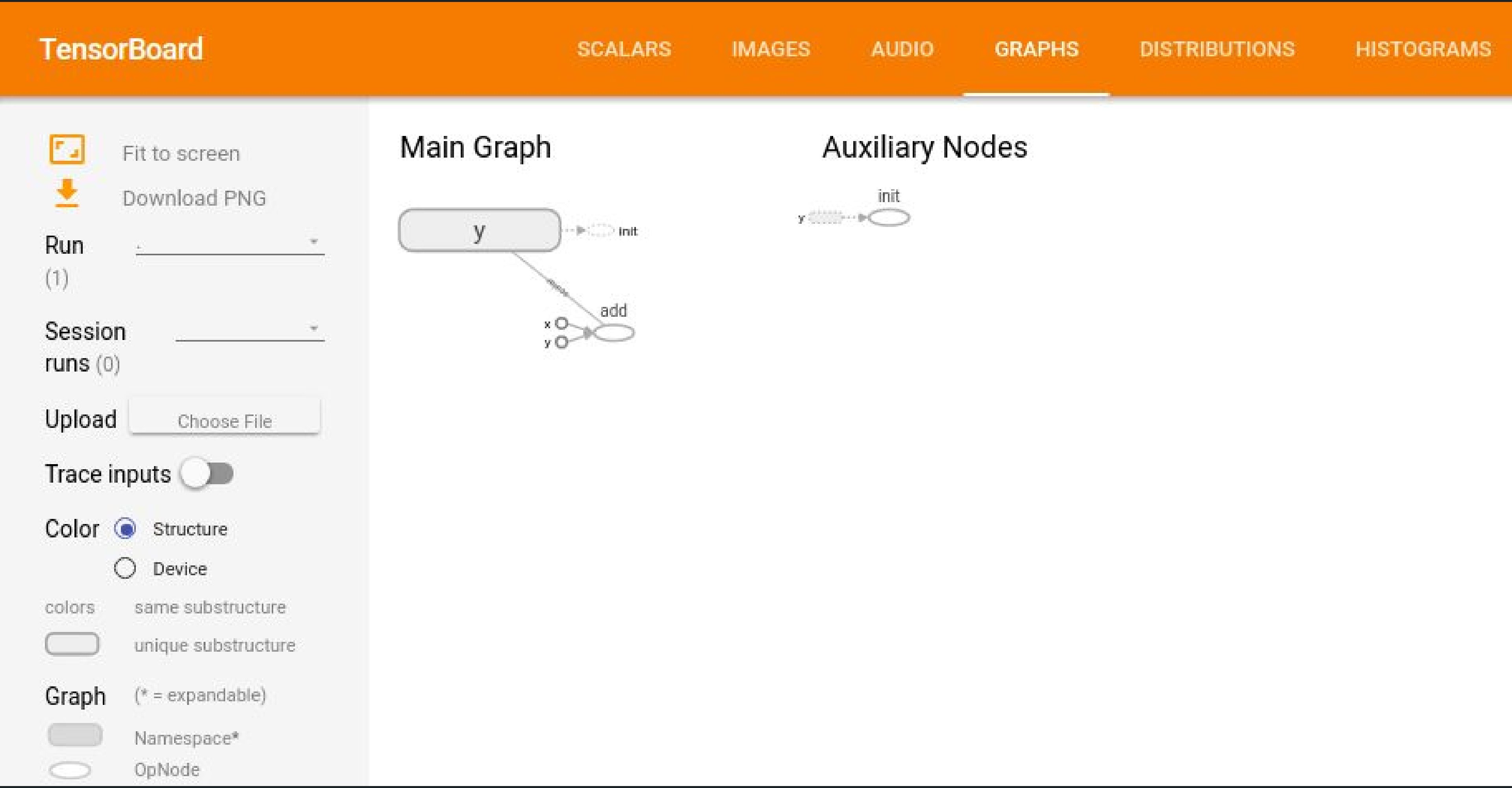
# Visualising Graph

```
import tensorflow as tf

x = tf.constant(35, name='x')
y = tf.Variable(x + 5, name='y')
model = tf.global_variables_initializer()

with tf.Session() as sess:
     sess.run(model)
     print(sess.run(y))
     writer = tf.summary.FileWriter('/tmp/tfGraphs', sess.graph)
writer.close() # close the writer when you’re done using it
```

# Visualising Graph

```
import tensorflow as tf

x = tf.constant(35, name='x')
y = tf.Variable(x + 5, name='y')
model = tf.global_variables_initializer()

with tf.Session() as sess:
     sess.run(model)
     print(sess.run(y))
     writer = tf.summary.FileWriter('/tmp/tfGraphs', sess.graph)
writer.close() # close the writer when you’re done using it
```

Run with
tensorboard --logdir=/tmp/tfGraphs


TensorBoard
SCALARS
IMAGES
AUDIO
GRAPHS
DISTRIBUTIONS
HISTOGRAMS
Fit to screen
Download PNG
Run
(1)
Session
runs (0)
Upload
Choose File
Trace inputs
Color
Structure
Device
colors
same substructure
unique substructure
Graph
(* = expandable)
Namespace*
OpNode
Main Graph
y
init
add
x
y
Auxiliary Nodes
init
y

# Data Structures

## Placeholder

- Commonly used for input to graph
- Requires feeding of value for the variable at graph run time, rather than variable declaration time.
- Data fed in using feed_dict in graph run command

# Placeholders

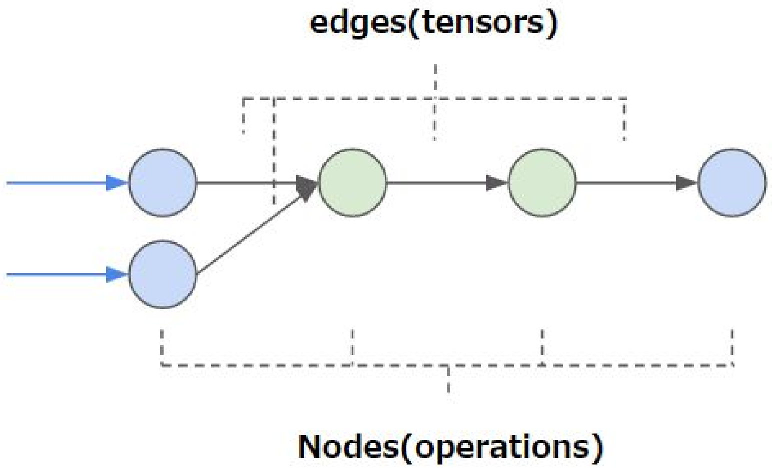
Note the usage of ‘None’ to specify undefined size for the input

```
import tensorflow as tf

x = tf.placeholder("float", [None, 3])
y = x * 2

with tf.Session() as session:
    x_data = [[1, 2, 3],
              [4, 5, 6],]
    result = session.run(y, 
feed_dict={x: x_data})
    print(result)
```

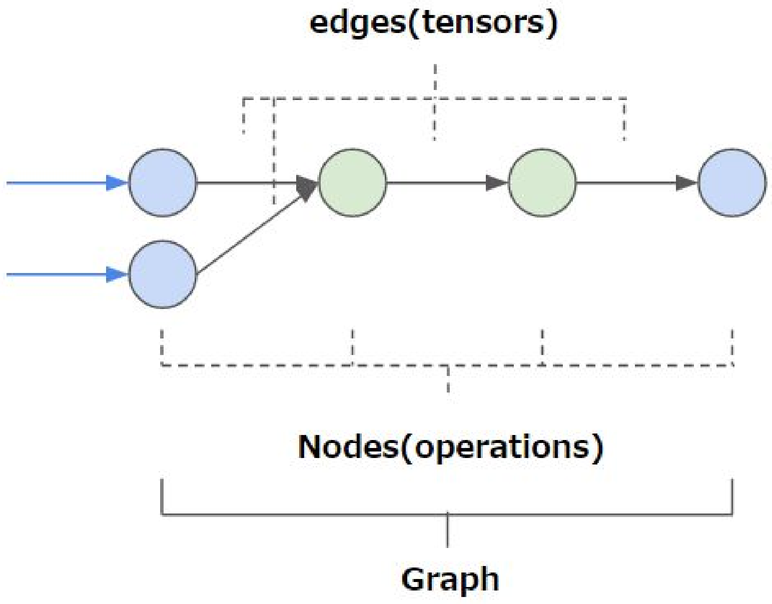
# Getting Stuff Done In TF

(Taken from NLintz' Tutorials)

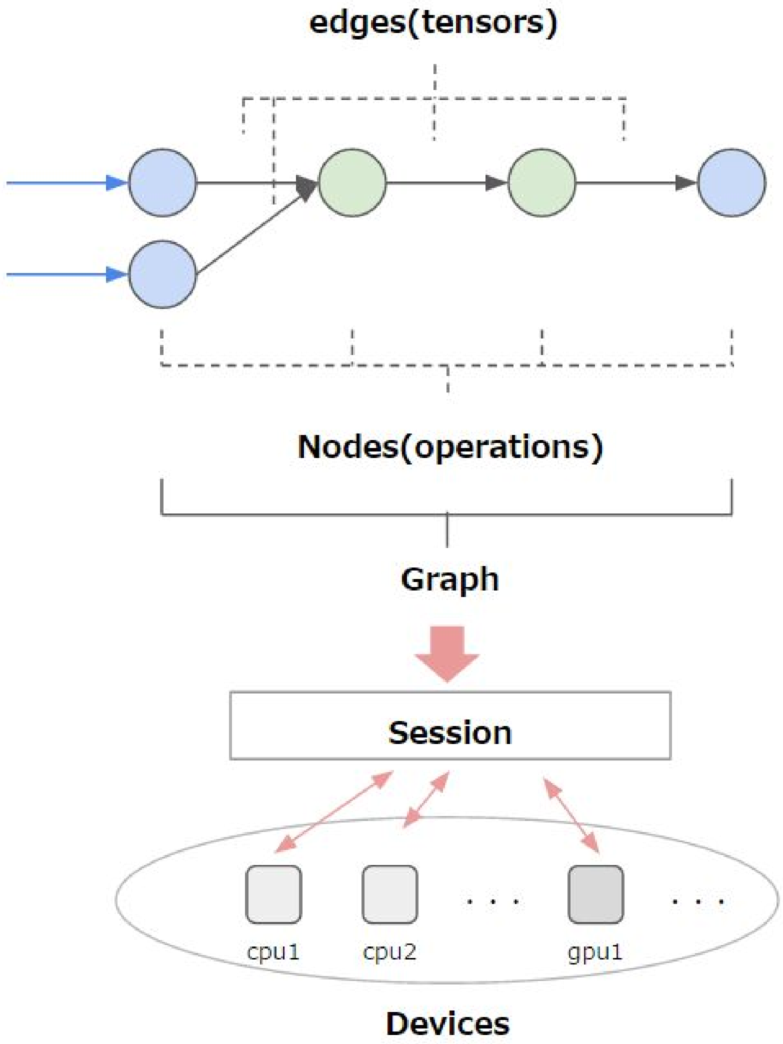
# Structuring TF Code 101



# Structuring TF Code 101



# Structuring TF Code 101



# Getting Stuff done in Tensorflow

Break it down into five steps

- Take in Inputs
- Define Network / Model
- Define Cost
- Define Optimization technique
- Train

# Try with MNIST example

Placeholders

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import
input_data
mnist = input_data.read_data_sets('MNIST_data',
                                  one_hot=True)

X = tf.placeholder(tf.float32, [128, 784])
Y_true = tf.placeholder(tf.float32, [128, 10])
```

Parameters and Operations

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)

X = tf.placeholder(tf.float32, [128, 784])
Y_true = tf.placeholder(tf.float32, [128, 10])
m = tf.get_variable('m', [784, 10])
b = tf.get_variable('b', [10])
Y_pred = tf.nn.xw_plus_b(X, m, b)
```

```
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)

X = tf.placeholder(tf.float32, [128, 784])
Y_true = tf.placeholder(tf.float32, [128, 10])
m = tf.get_variable('m', [784, 10])
b = tf.get_variable('b', [10])
Y_pred = tf.nn.xw_plus_b(X, m, b)
```

Cost

```
cost =
tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
                                          Y_pred, Y_true))
```

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)

X = tf.placeholder(tf.float32, [128, 784])
Y_true = tf.placeholder(tf.float32, [128, 10])
m = tf.get_variable('m', [784, 10])
b = tf.get_variable('b', [10])
Y_pred = tf.nn.xw_plus_b(X, m, b)

cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(Y_pred,
                                                               Y_true))
```

Optimizer

```
optimzer =
tf.train.GradientDescentOptimizer(learning_rate=0.5)
.minimize(cost)
```

```
X = tf.placeholder(tf.float32, [128, 784])
Y_true = tf.placeholder(tf.float32, [128, 10])
m = tf.get_variable('m', [784, 10])
b = tf.get_variable('b', [10])
Y_pred = tf.nn.xw_plus_b(X, m, b)

cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(Y_pred,
                                                                Y_true))
optimzer =
tf.train.GradientDescentOptimizer(learning_rate=0.5) .minimize(cost)
```

Train Code

```
sess = tf.Session()
sess.run(tf.initialize_all_variables())

for i in range(2000):
    trX, trY = mnist.train.next_batch(128)
    sess.run(optimzer, feed_dict={X: trX, Y_true: trY})
```

# Final Tricks

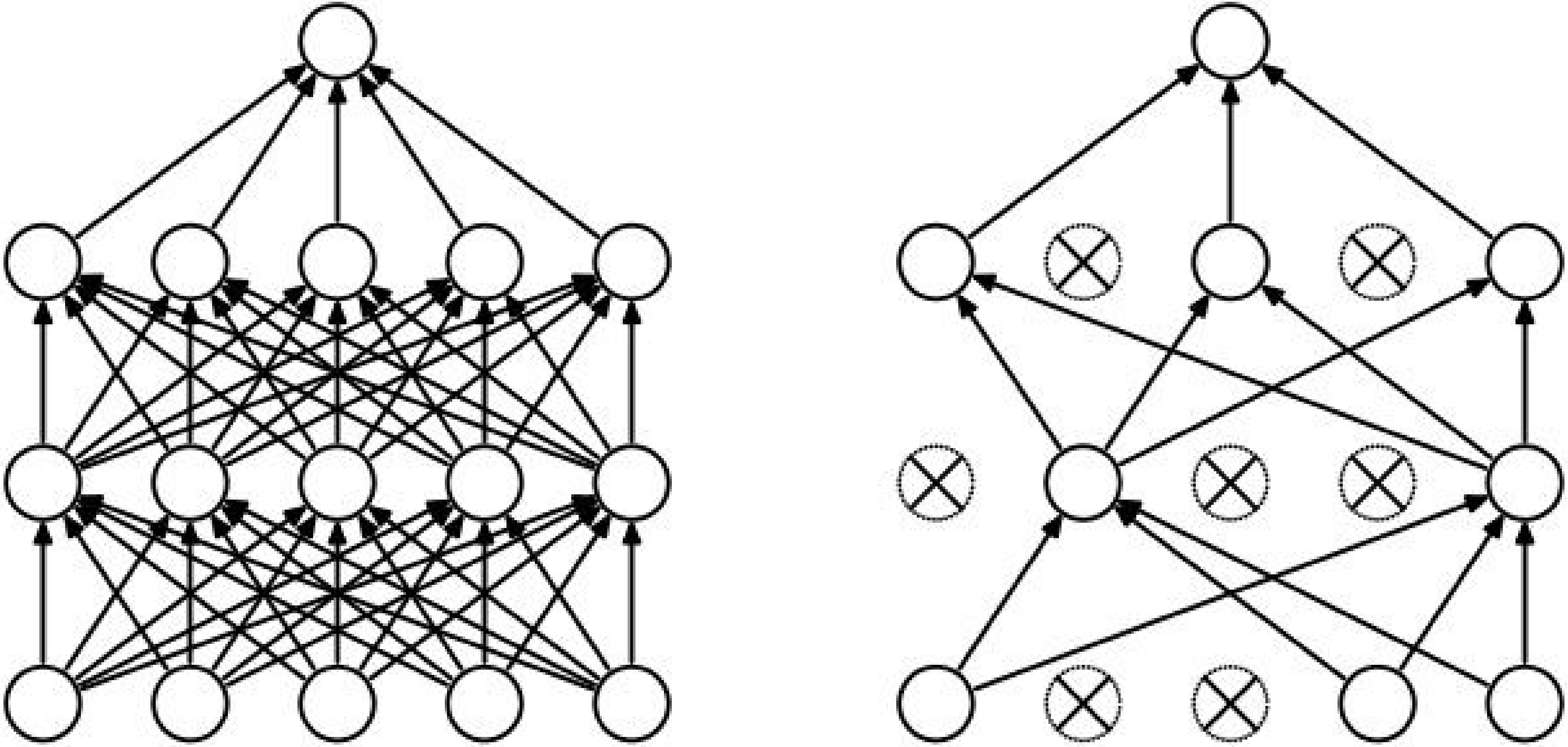
# Scaling to Optimize?

- When doing cost predictions using TF(Ex: Using softmax, similar to previous slide), pass in unscaled predictions, don’t scale them first.

- TF does it’s own optimization, and it does it brilliantly. If any scaling is to be done in order to avoid overflows/underflows, TF will take care of it

# Parameter Sharing

- In dropout and other train-time stochastic methods, make sure you use tf.get_variable() as giving different dropout keep probabilities actually creates two different models with two different variable sets. For instance, (see next few slides)

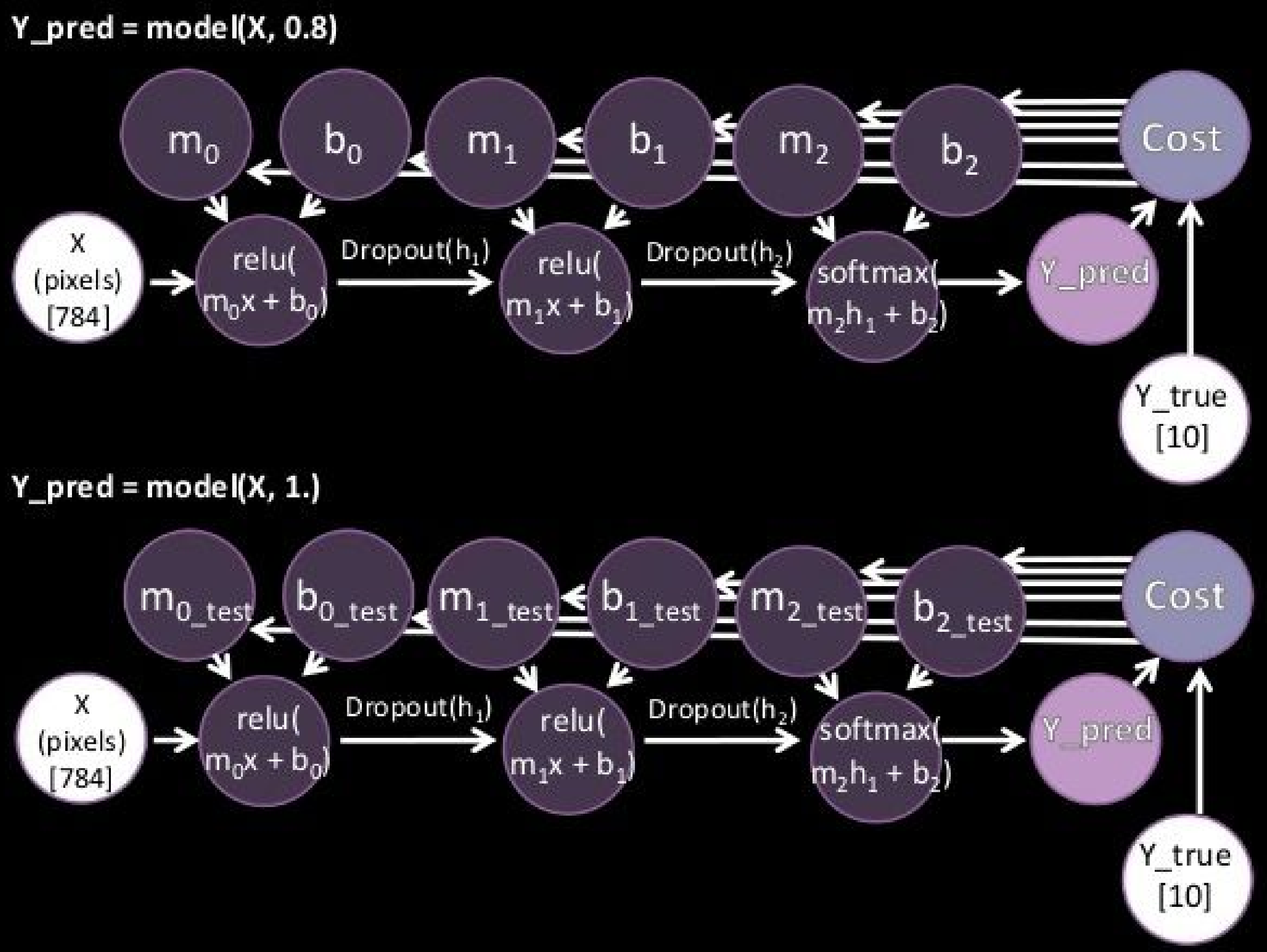


(a) Standard Neural Net

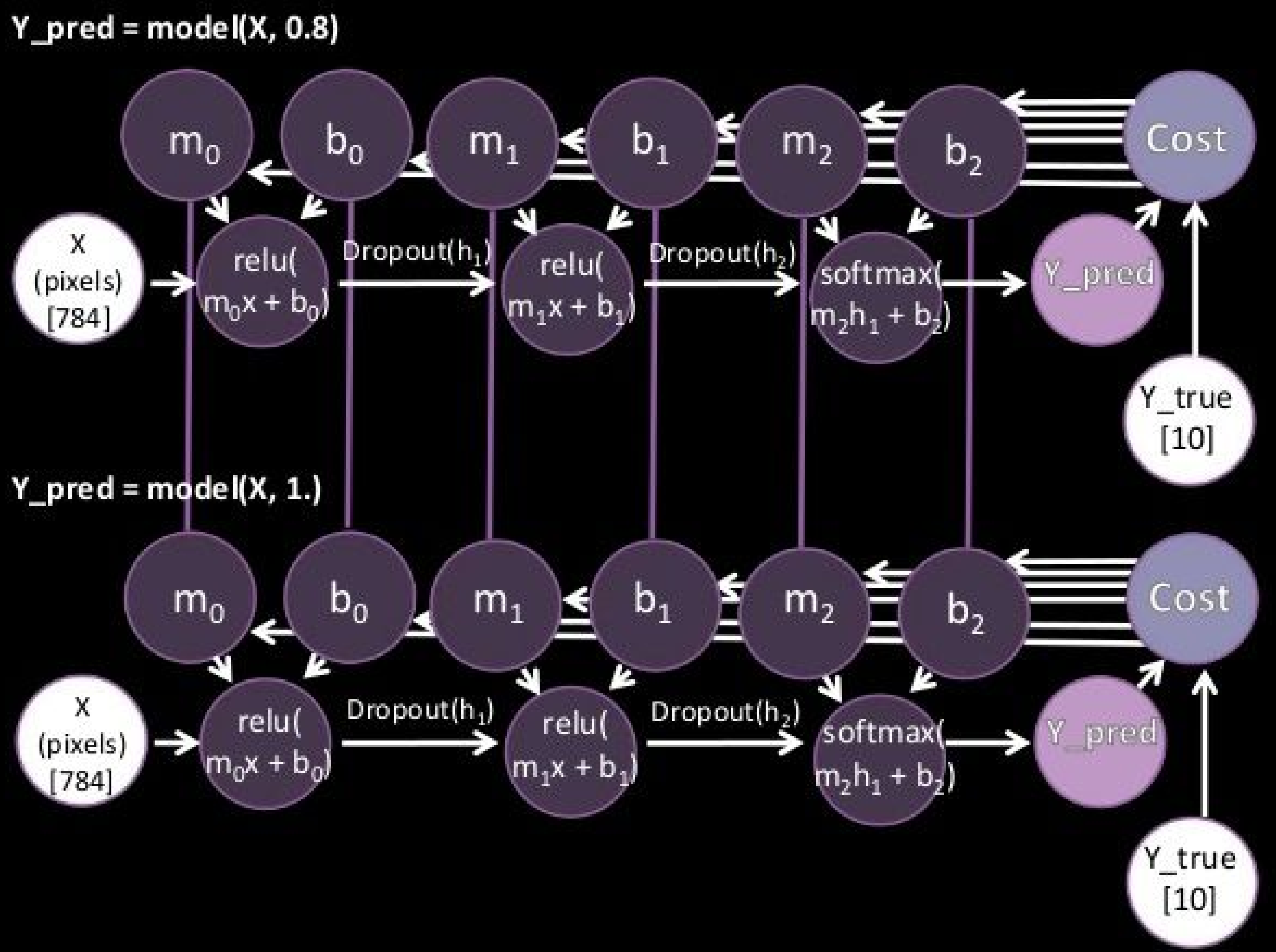
(b) After applying dropout.

Illustration borrowed from CS231n notes

# Without Weight Sharing



# With Weight Sharing



# Parameter Sharing

```
def model(X, p_keep):
    m0 = tf.get_variable('m0', [784, 256])
    b0 = tf.get_variable('b0', [256],   initializer=tf.constant_initializer(0.))
    m1 = tf.get_variable('m1', [256, 256])
    b1 = tf.get_variable('b1', [256],   initializer=tf.constant_initializer(0.))
    m2 = tf.get_variable('m2', [256, 10])
    b2 = tf.get_variable('b2', [10],    initializer=tf.constant_initializer(0.))
    h1 = tf.nn.relu(tf.nn.xw_plus_b(X, m0, b0))
    h1 = tf.nn.dropout(h1, p_keep)
    h2 = tf.nn.relu(tf.nn.xw_plus_b(h1, m1, b1))
    h2 = tf.nn.dropout(h2, p_keep)
    output = tf.nn.xw_plus_b(h2, m2, b2)
    return output

Y_pred = model(X, 0.8)
Y_pred_test = model(X, 1.)
```

# Parameter Sharing (correct)

```
def model(X, p_keep):
    m0 = tf.get_variable('m0', [784, 256])
    b0 = tf.get_variable('b0', [256],         initializer=tf.constant_initializer(0.))
    m1 = tf.get_variable('m1', [256, 256])
    b1 = tf.get_variable('b1', [256],         initializer=tf.constant_initializer(0.))
    m2 = tf.get_variable('m2', [256, 10])
    b2 = tf.get_variable('b2', [10],          initializer=tf.constant_initializer(0.))
    h1 = tf.nn.relu(tf.nn.xw_plus_b(X, m0, b0))
    h1 = tf.nn.dropout(h1, p_keep)
    h2 = tf.nn.relu(tf.nn.xw_plus_b(h1, m1, b1))
    h2 = tf.nn.dropout(h2, p_keep)
    output = tf.nn.xw_plus_b(h2, m2, b2)
    return output
with tf.variable_scope("model") as scope:
    Y_pred = model(X, 0.8)
    scope.reuse_variables()
    Y_pred_test = model(X, 1.)
```

# Collections

- Are a bucket which can contain parameters, operations, model predictions, etc for later reference more conveniently

  P.S.: TF automatically creates some collections

# Collections

```
def model(X):
    m0 = tf.get_variable('m0', [784, 256])
    b0 = tf.get_variable('b0', [256], initializer=tf.constant_initializer(0.))
    m1 = tf.get_variable('m1', [256, 256])
    b1 = tf.get_variable('b1', [256], initializer=tf.constant_initializer(0.))
    m2 = tf.get_variable('m2', [256, 10])
    b2 = tf.get_variable('b2', [10], initializer=tf.constant_initializer(0.))
    h1 = tf.nn.relu(tf.nn.xw_plus_b(X, m0, b0))
    h2 = tf.nn.relu(tf.nn.xw_plus_b(h1, m1, b1))
    tf.add_to_collection("activations", h1)
    tf.add_to_collection("activations", h2)
    output = tf.nn.xw_plus_b(h2, m2, b2)
    return output

Y_pred = model(X)
```

# Collections

```
activations = tf.get_collection('activations')
activations_values = session.run(activations)

parameters =  tf.get_collection('trainable_parameters')
parameter_values = session.run(parameters)
```

# Stuff to read up

- Image handling using TensorFlow: https://www.tensorflow.org/versions/master/api_docs/python/image.html#encoding-and-decoding
- How to run interactive sessions:
- TF Learn, a semi-high level API to TF
- TF Records, TF's default binary storage format

# Acknowledgements

- CS230i and CS231n - Stanford Courses
- NLintz' tutorials