Intro to TensorFlow

Ben Kubit

NSC211 Spring 2017

University of California, Davis

Overview

- TF Python API
- Building a network
- XOR example
- TensorBoard

TF Python API

- Graphs and Sessions:
  - A **computational graph** is a series of TensorFlow operations arranged into a graph of nodes.
  - To actually evaluate the nodes, we must run the computational graph within a **session**.
    - Graph is run entirely outside Python. TF relies on efficient C++ backend for computations.
    - Connection to this backend is called a **session**.
  - Since TF knows the entire **computation graph**, faster computing
    - E.g., automatic differentiation to find loss gradients with respect to each variable.

TF Python API

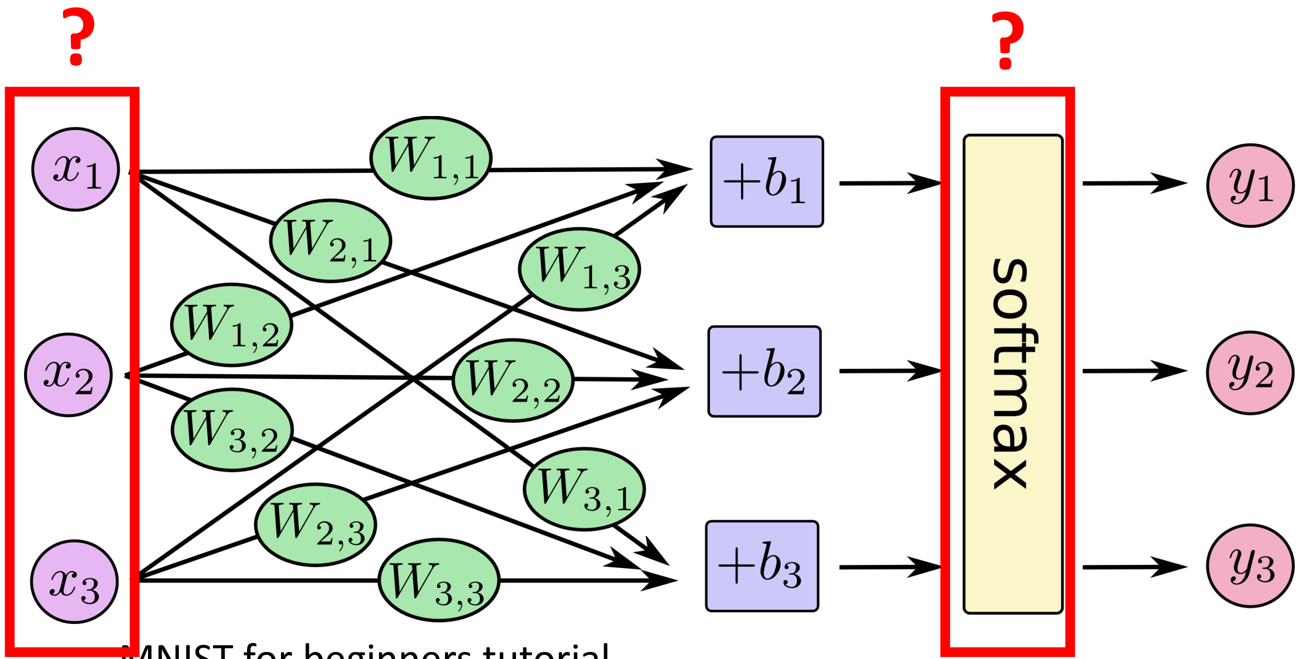
- *Tensor Nodes*: may be thought of as an operation that takes no inputs and always produces the same output corresponding to the constant/variable it represents.
  - **Constants**: hard-coded values.
  - **Placeholders**: promise to provide a value later. Specify **placeholder** values when you run a session (typically input data/predicted values).
  - **Variables**: allow us to add trainable parameters to a graph. Initialize on first run, then dynamically update based on optimization function (typically model parameters).
- *Operation Nodes*: express the combination/transformation of data flowing through the graph
  - E.g., add_node = y + x -> receives input from tensor nodes y and x
  - E.g., lm_node = tf.matmul(x, w) + b
- *Summary Nodes*: capture informaiton during training for use with TensorBoard

Building a Network

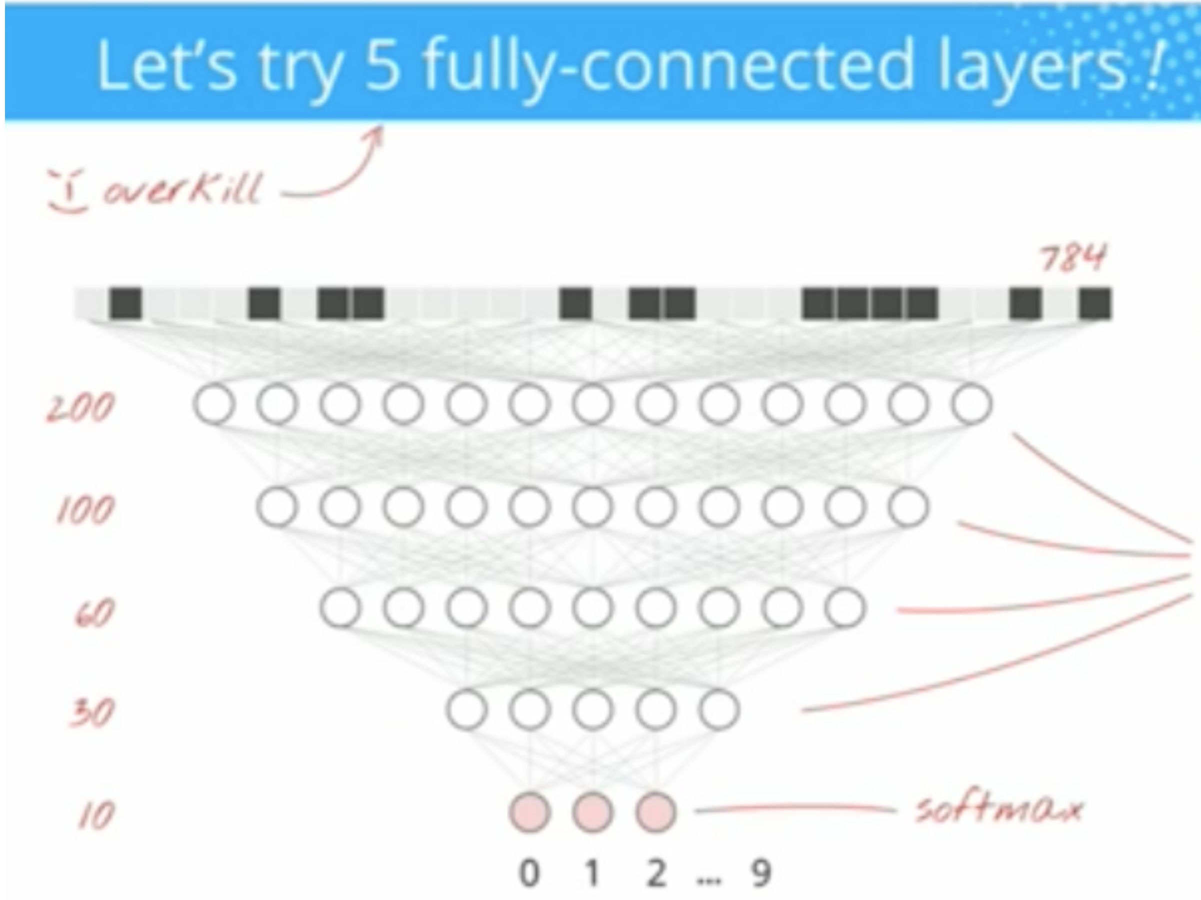
- Inference - Build the graph
  - Initialize operation nodes and tensor nodes
  - Form of output units -> functions + # of nodes
- Loss - Choose loss function
  - E.g., MSE, cross-entropy
- Training - Choose optimizer
  - E.g., Gradient descent

Building a Network

- What is a layer?!
  - Project inputs into a new space.
  - Has weights for each incoming edge and biases for each node in the layer.



MNIST for beginners tutorial
https://www.tensorflow.org/get_started/mnist/beginners



Tensorflow and deep learning - without a PhD by Martin Görner
https://www.youtube.com/watch?v=vq2nnJ4g6N0

Deep Learning 6.1 Example: Learning XOR

- The XOR function (“exclusive or”):
  - operation on two binary values, x1 and x2.
    - when only one of these values==1, the function returns 1, otherwise 0
  - 2-way classification
    - We can treat this problem as a regression problem
  - We want our network to perform correctly on the four points:
    - X = {[0,0] , [0,1] , [1,0] , and [1,1] }.
- Examine:
  - utility of hidden layer transformations
  - Influence of initial weight values
  - Low-level ML (vs tf.contrib.train high-level ML)
- Code:
  - NSC211_BKlecture_code.ipynb

Deep Learning 6.1 Example: Learning XOR

- We can minimize in closed form with respect to w and b using the normal equations.
  - After solving the normal equations, we obtain w = 0 and b = 1/2.
    - The linear 2 model simply outputs 0.5 everywhere!

```
sess = tf.Session()
#input data
XOR_X = [[0,0],[0,1],[1,0],[1,1]] #input
#XOR_Y = [[0],[1],[1],[0]] #predicted

#placeholders
x_ = tf.placeholder(tf.float32, shape=[4,2], name="x-input")
#use weights/biases from book example
w = tf.Variable(tf.zeros([2,1]), tf.float32)
b = tf.Variable([1/2.], tf.float32)

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

#operation node
linear_model = tf.matmul(x_,w) + b

#see what the predictions are
print(sess.run(linear_model, {x_: XOR_X}))
```

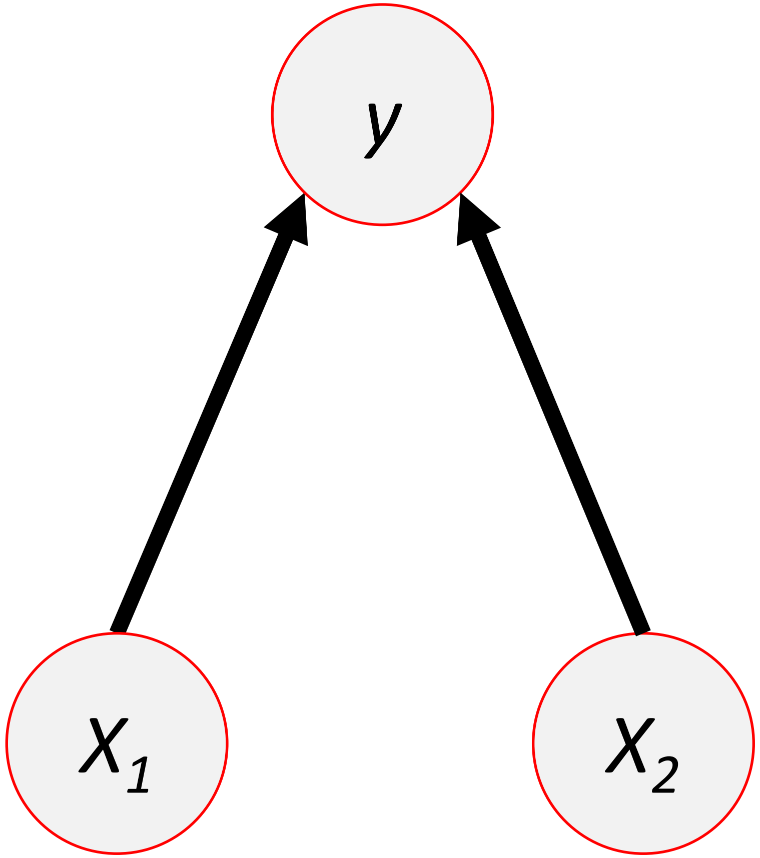
Output:

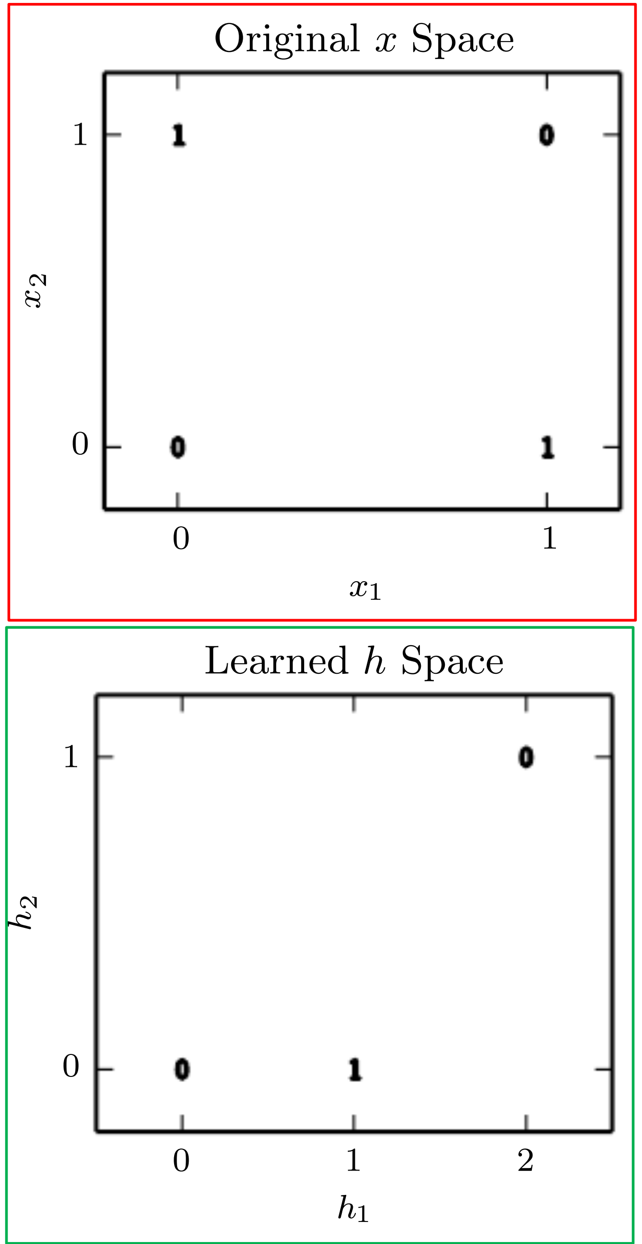
```
[[ 0.5]
 [ 0.5]
 [ 0.5]
 [ 0.5]]
```

Single fully-connected layer: 2 inputs (so 2 weights total); one bias for the one node

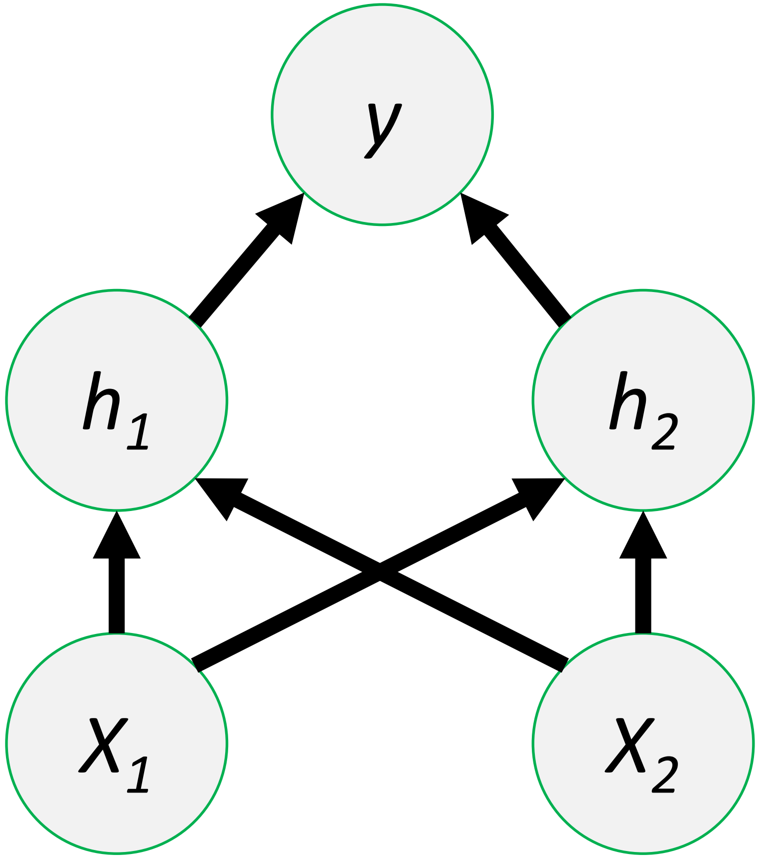
Deep Learning 6.1 Example: Learning XOR

Network 1





Network 2



Deep Learning 6.1 Example: Learning XOR

- solve using a model that learns a different feature space in which a linear model is able to represent the desired solution.
  - simple feedforward network with one hidden layer -> change what is given to output layer

```
sess2 = tf.Session()
#input data
XOR_X = [[0,0],[0,1],[1,0],[1,1]] #input
#XOR_Y = [[0],[1],[1],[0]] #predicted

#placeholders
x_ = tf.placeholder(tf.float32, shape=[4,2], name="x-input")
#use weights/biases from book example
w1 = tf.Variable(tf.ones([2,2]), tf.float32) #W
w2 = tf.Variable([[1.],[-2.]], tf.float32) #w
b1 = tf.Variable([[0.,-1.]], tf.float32) #c
b2 = tf.Variable(tf.zeros(1), tf.float32) #b

init2 = tf.global_variables_initializer()
sess2.run(init2)

#operation nodes
transformedH = tf.nn.relu(tf.matmul(x_,w1) + b1, name=None) #hidden layer with rect. linear act. func.
linear_model = tf.matmul(transformedH, w2) + b2

#see what the predictions are
print(sess2.run(linear_model, {x_: XOR_X}))
```

Output:

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

1st fully-connected layer: 4 inputs (so 4 weights total); 2 bias for the 2 nodes

2nd fully-connected layer: 2 inputs (so 2 weights total); 1 bias for the single node

Deep Learning 6.1 Example: Learning XOR

- In a real situation, there are lots of model parameters and training examples
  - we cannot guess the solution as we did before.
- now solve the same problem but let's use gradient-based optimization to find params
  - in order to do so need to measure error/cost
  - also need predicted values!

```
sess3 = tf.Session()
#input data
XOR_X = [[0,0],[0,1],[1,0],[1,1]] #input
XOR_Y = [[0],[1],[1],[0]] #predicted

#placeholders, now we need one for predicted vals too
x_ = tf.placeholder(tf.float32, shape=[4,2], name="x-input")
y_ = tf.placeholder(tf.float32, shape=[4,1], name="y-input")
#now we will init with some random values
w1 = tf.Variable(tf.random_uniform([2,2], -2, 2), tf.float32) #W
w2 = tf.Variable(tf.random_uniform([2,1], -2, 2), tf.float32) #w
b1 = tf.Variable(tf.zeros([2]), tf.float32) #c
b2 = tf.Variable(tf.zeros([1]), tf.float32) #b
```

Predicted values (labels)

Initialize variables with random values and zeros

Deep Learning 6.1 Example: Learning XOR

```
#operation nodes
transformedH = tf.nn.relu(tf.matmul(x_,w1) + b1) #hidden layer with rect. linear act. func.
linear_model = tf.matmul(transformedH, w2) + b2

#MSE
loss = tf.reduce_sum(tf.square(linear_model - y_)) #create error vector.We call tf.square to square that error.

#gradient descent
optimizer = tf.train.GradientDescentOptimizer(0.01) #0.01 is learning rate
train = optimizer.minimize(loss) #feed optimizer loss function

init3 = tf.global_variables_initializer()
sess3.run(init3)

#train it
for i in range(10000):
        sess3.run(train, {x_: XOR_X, y_: XOR_Y})

#take a look at the results
predictions = sess3.run(linear_model, {x_: XOR_X})
curr_w1, curr_w2, curr_b1, curr_b2, curr_loss  = sess3.run([w1, w2, b1, b2, loss], {x_: XOR_X, y_: XOR_Y})
hidlay  = sess3.run(transformedH, {x_: XOR_X, y_: XOR_Y})
print("predictions:\n %s\n hlayinput:\n %s\n"%(predictions,hidlay))
print("w1:\n %s \nw2:\n %s \nb1: %s \nb2: %s \nloss: %s"%(curr_w1, curr_w2, curr_b1, curr_b2, curr_loss))
```

Feed 2nd layer output into loss function

Feed loss function output into optimizer

run optimizer (updates param vars)

Output1:

```
predictions:
 [[  4.99999642e-01]
 [   9.99998569e-01]
 [   4.99999642e-01]
 [   1.46031380e-06]]
```

Output2:

```
predictions:
 [[  2.81445682e-06]
 [   9.99997795e-01]
 [   9.99997795e-01]
 [   1.61863863e-06]]
```

Output3:

```
predictions:
 [[  5.00001073e-01]
 [   5.00001073e-01]
 [   9.99999583e-01]
 [  -2.38418579e-07]]
```

Deep Learning 6.1 Example: Learning XOR

- using this approach we often find different solutions because the minima found depends on the rand. initial weights

Output1:

```
predictions:
 [[  4.99999642e-01]
 [  9.99998569e-01]
 [  4.99999642e-01]
 [  1.46031380e-06]]
 hlayinput:
 [[ 0.          0.        ]
 [ 0.81953818  1.01845491]
 [ 0.          0.        ]
 [ 0.          1.48590386]]

w1:
 [[-0.8780849   0.46744898]
 [ 1.40672278  1.49006176]]
w2:
 [[ 1.02826595]
 [-0.3364943 ]]
b1: [-0.58718461 -0.47160682]
b2: [ 0.49999964]
loss: 0.5
```

Output2:

```
predictions:
 [[  2.81445682e-06]
 [  9.99997795e-01]
 [  9.99997795e-01]
 [  1.61863863e-06]]
 hlayinput:
 [[ 0.          0.00977176]
 [ 0.          1.08689225]
 [ 0.          1.08689225]
 [ 1.17514503  2.16401291]]

w1:
 [[ 1.17514503  1.07712054]
 [ 1.17514503  1.07712054]]
w2:
 [[-1.70191026]
 [ 0.92839658]]
b1: [-1.17514503  0.00977176]
b2: [-0.00906925]
loss: 2.02685e-11
```

Output3:

```
predictions:
 [[  5.00001073e-01]
 [  5.00001073e-01]
 [  9.99999583e-01]
 [ -2.38418579e-07]]
 hlayinput:
 [[ 0.          0.        ]
 [ 0.          0.        ]
 [ 0.13460982  1.74021423]
 [ 0.38378608  0.        ]]

w1:
 [[ 1.02023137  1.80393577]
 [ 0.24917631 -1.88829768]]
w2:
 [[-1.30281258]
 [ 0.38809583]]
b1: [-0.88562155 -0.06372153]
b2: [ 0.50000107]
loss: 0.5
```

Deep Learning 6.1 Example: Learning XOR

- if we set the weights closer to the values provided in the example the output is more consistent and similar to predicted values

```
sess4 = tf.Session()
#input data
XOR_X = [[0,0],[0,1],[1,0],[1,1]] #input
XOR_Y = [[0],[1],[1],[0]] #predicted

#placeholders, now we need one for predicted vals too
x_ = tf.placeholder(tf.float32, shape=[4,2], name="x-input")
y_ = tf.placeholder(tf.float32, shape=[4,1], name="y-input")
#constrain rand. values
w1 = tf.Variable(tf.random_uniform([2,2], .7, 1.3), tf.float32) #W
w2 = tf.Variable(tf.random_uniform([2,1], -2, 1), tf.float32) #w
b1 = tf.Variable(tf.zeros([2]), tf.float32) #c
b2 = tf.Variable(tf.zeros([1]), tf.float32) #b
```

Initialize variables with random values and zeros

.
.
.

W(w1): [[1, 1]
[1, 1]]

w(w2): [[-2]
[1]]

b1(c): [-1 0]

b2(b): [[0]]

Output:

```
predictions:
 [[  3.11833674e-06]
 [  9.99997497e-01]
 [  9.99997616e-01]
 [  1.76924414e-06]]
 hlayinput:
 [[  0.00000000e+00   3.96847035e-08]
 [  0.00000000e+00   1.05011714e+00]
 [  0.00000000e+00   1.05011725e+00]
 [  1.31451106e+00   2.10023451e+00]]

w1:
 [[ 1.31451106  1.05011725]
 [ 1.31451106  1.05011714]]
w2:
 [[-1.5214709 ]
 [ 0.95226938]]
b1: [ -1.31451106e+00   3.96847035e-08]
b2: [  3.08054632e-06]
loss: 2.48056e-11
```

Deep Learning 6.1 Example: Learning XOR

- Summary
  - Utility of hidden layer transformations
  - Importance of initial parameter values
- Better way to save diagnostic info?
  - TensorBoard
    - Visualize the graph
    - Generate summary nodes

Better way to save diagnostic info?

- TensorBoard

```
linear_model = tf.matmul(transformedH, w2) + b2
tf.summary.histogram("predicted", linear_model)

#MSE
loss = tf.reduce_sum(tf.square(linear_model - y_))
tf.summary.scalar("curr_loss", loss)

# Build the summary Tensor based on the TF collection of summaries.
summary = tf.summary.merge_all()
init = tf.global_variables_initializer()
sess = tf.Session()

# Instantiate a SummaryWriter to output summaries and the Graph.
summary_writer = tf.summary.FileWriter(log_dir, sess.graph)
sess.run(init)
#train it
for i in range(10000):
    #run the session and get summary info
    _, suminfo = sess.run([train, summary], feed_dict={x_: XOR_X, y_: XOR_Y})
    # Write the summaries every 100 trials.
    if i % 100 == 0:
        summary_writer.add_summary(suminfo, i) # Update the events file.
        summary_writer.flush()
```

collect prediction values

collect loss value

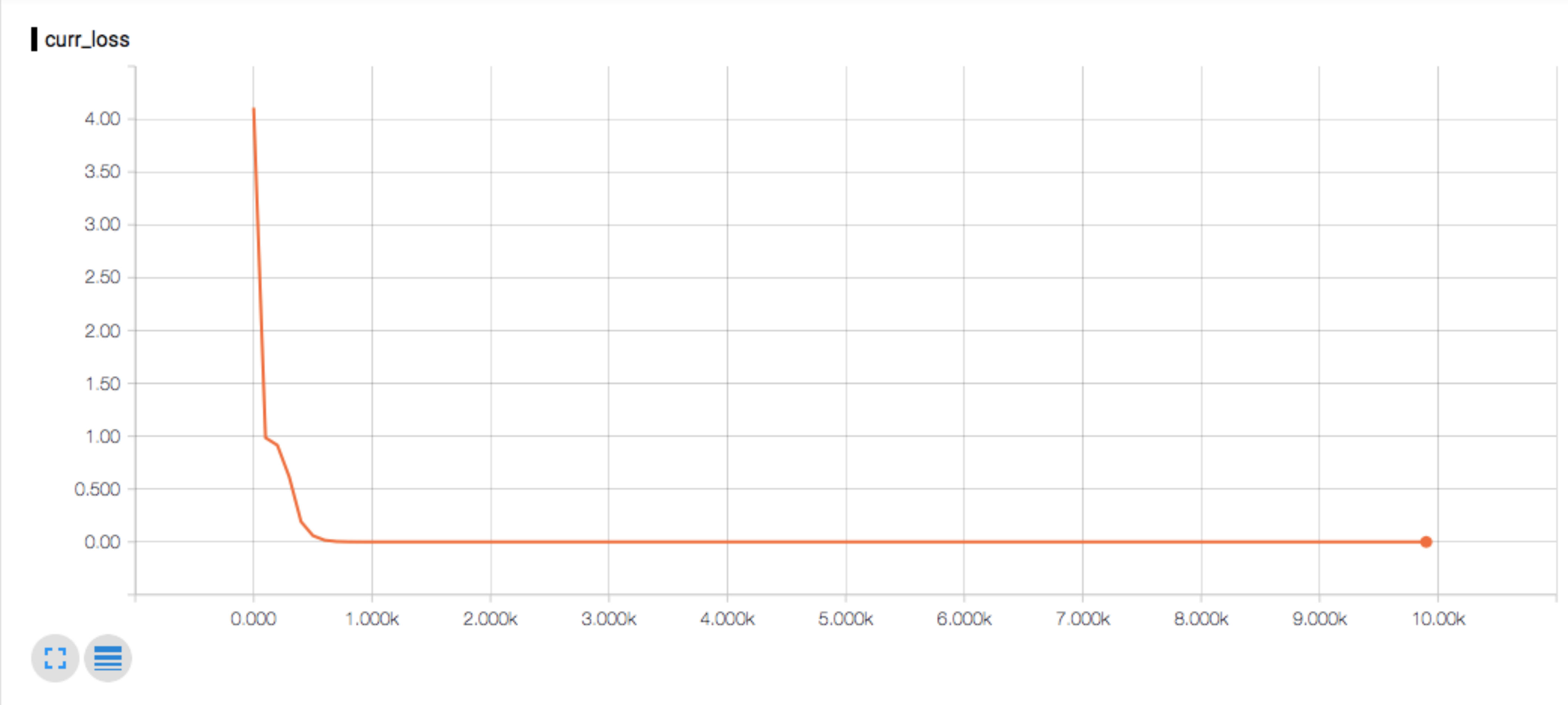
collect all summary values

Create object to output summaries

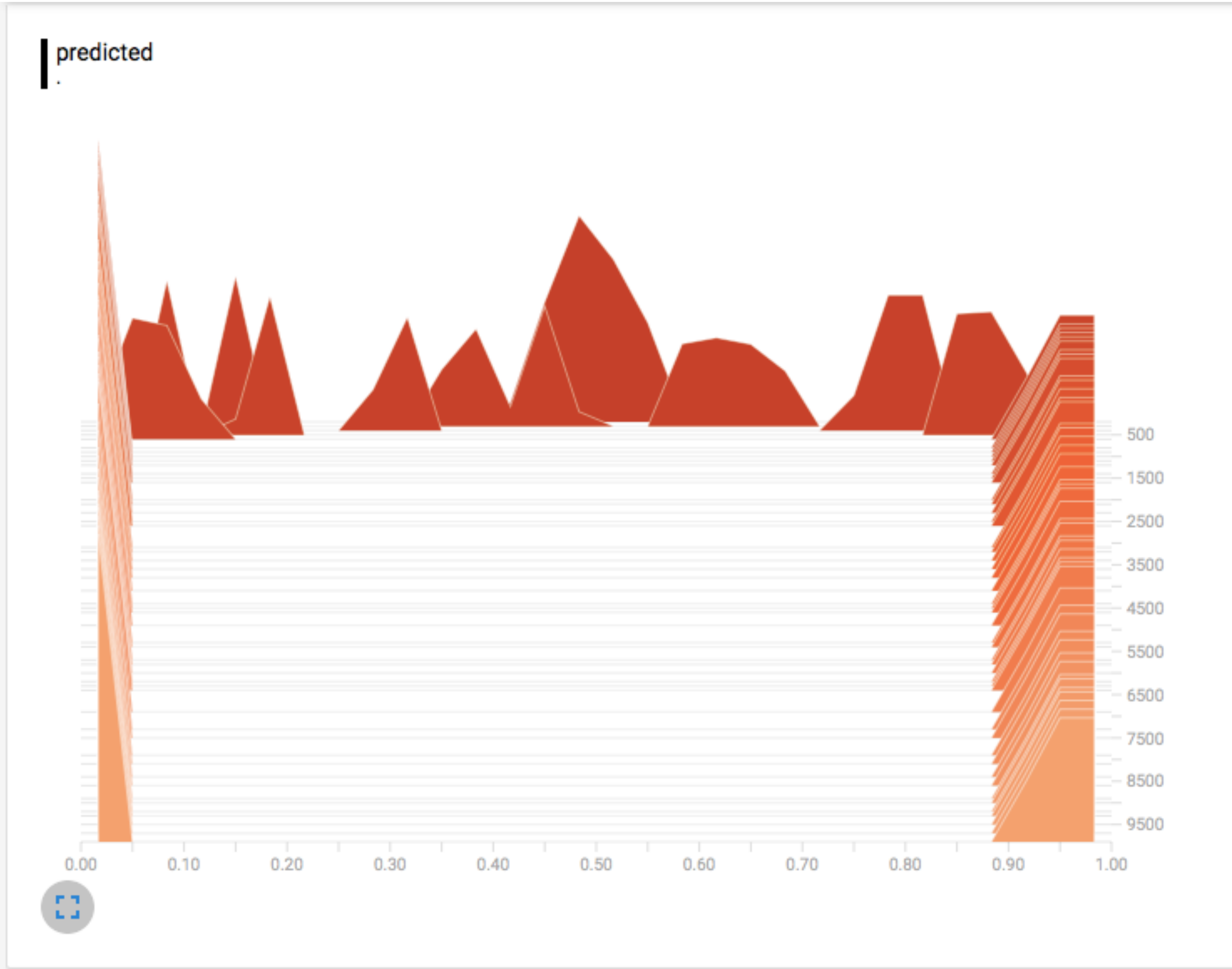
Gather summary values for this iter

Write out summaries to file and refresh

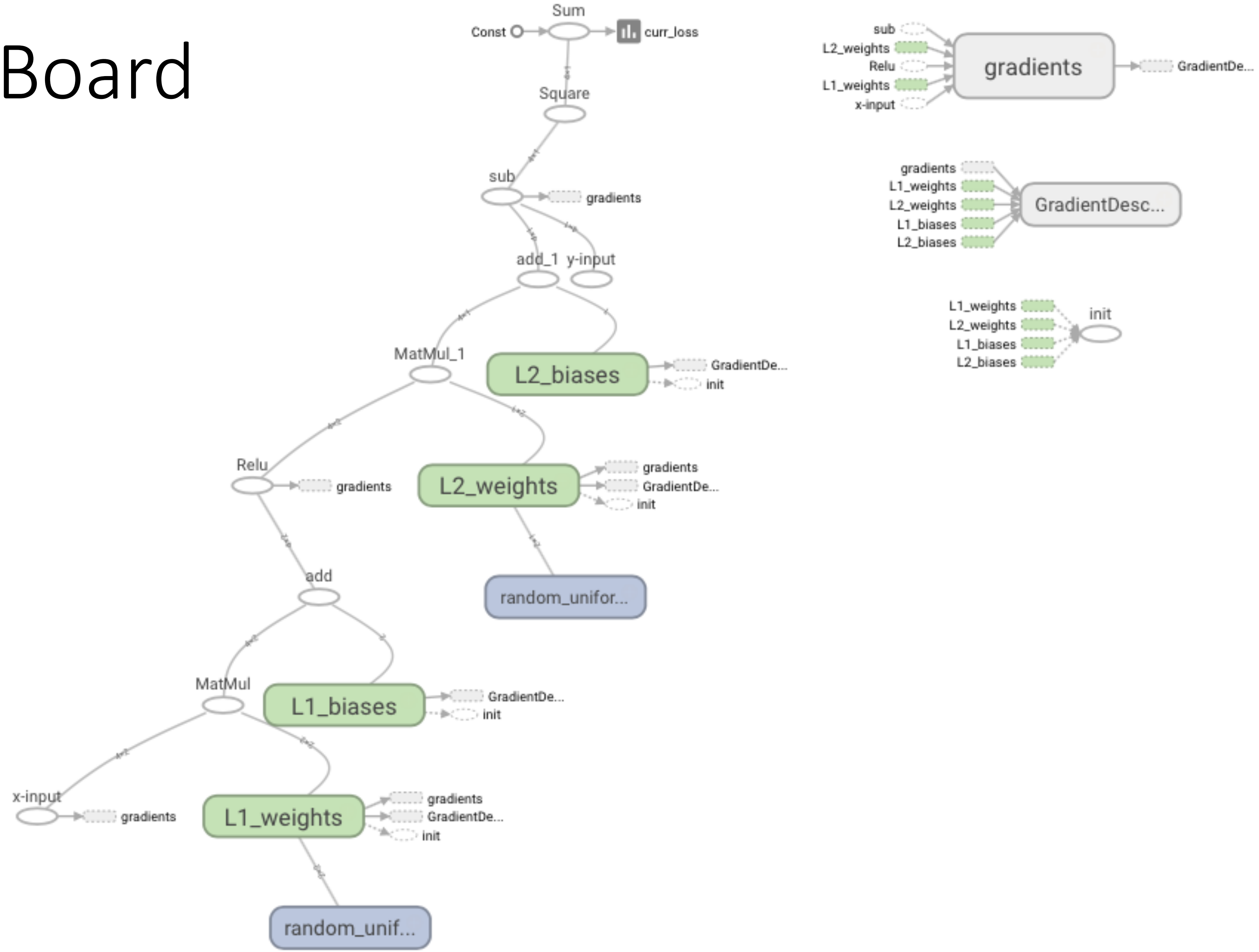
TensorBoard
Summary Scalar



TensorBoard
Summary Histogram



TensorBoard Graph



Summary

- TF Python API
  - https://www.tensorflow.org/get_started/get_started
  - https://www.tensorflow.org/programmers_guide/
  - https://www.tensorflow.org/api_docs/python/
- Building a network
  - https://www.youtube.com/watch?v=vq2nnJ4g6N0
- XOR TF example:
  - NSC211_BKlecture_code.ipynb
- Diagnostic tools:
  - https://www.tensorflow.org/get_started/summaries_and_tensorboard
  - https://www.tensorflow.org/api_guides/python/summary
  - https://www.tensorflow.org/api_guides/python/tfdbg
  - https://www.tensorflow.org/api_docs/python/tf/InteractiveSession

Additional considerations

- High-level API
  - E.g., tf.contrib.train
- Data preparation
  - E.g., batch normalization
- Batch processing
  - E.g., cross-validation