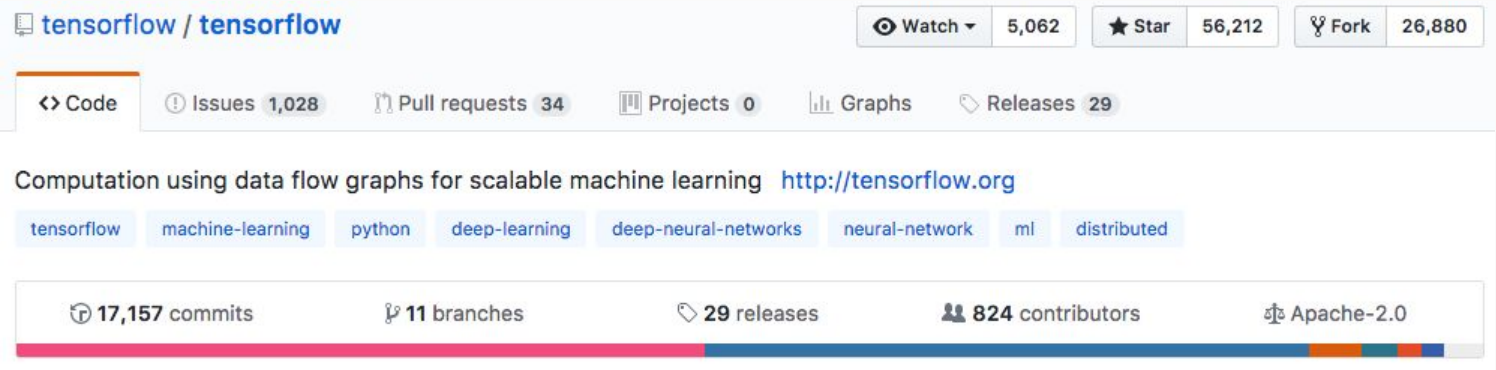
Introduction to Tensorflow

8 May 2017, Nelson Liu

What is TensorFlow?

- “An open-source software library for Machine Intelligence”
  - Think of it as NumPy, but with a computation graph and tuned for neural networks.
- Developed and maintained by Google Brain
- Very active community, used by hobbyists and corporations alike.



Part I: Basics

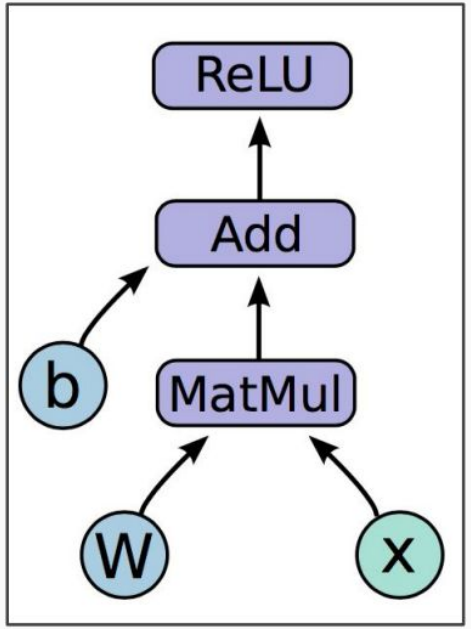
Graph Computation

Core Idea: Express computations as a **graph**

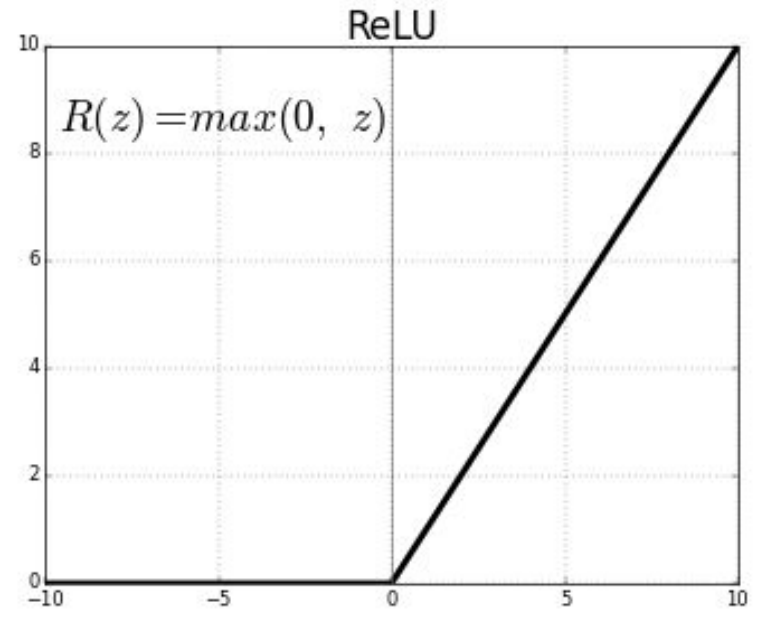
- T**he graph nodes represent numeric operations** (e.g. +, -, average), and can have any number of inputs (edges leading into the node) and outputs (edges leading out of the node).
- **The graph edges represent tensors,** n-dimensional arrays that flow through the graph through the nodes.

Example Graph: A one-layer neural net

$$h = ReLU(Wx + b)$$



Reminder: ReLU



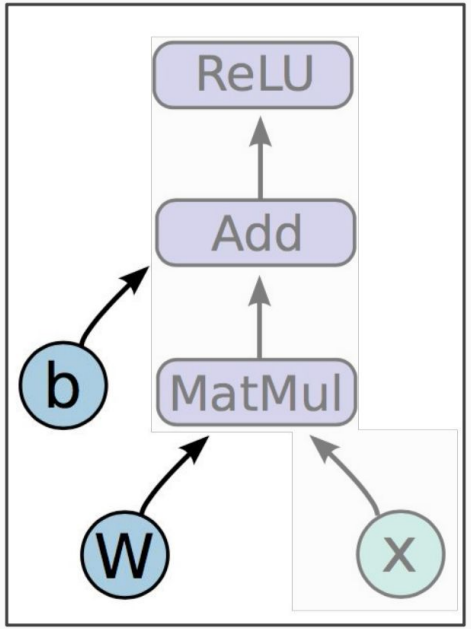
Example Graph: A one-layer neural net

Variables are nodes that output their current value.

Their value is retained across multiple “executions” of the graph.

Generally used to represent the parameters (weights) of your model.

$$h = ReLU(Wx + b)$$

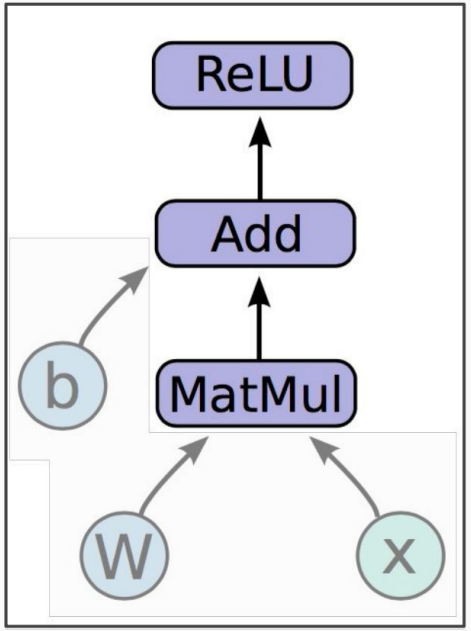


Example Graph: A one-layer neural net

Operations (ops) are nodes that represent mathematical operations to be performed on the data.

MatMul: Multiply two matrices.
Add: Elementwise addition between two tensors.
ReLU: Apply the ReLU activation function to each element of the tensor.

$$h = ReLU(Wx + b)$$



As code:

1. Create Variables W and b (the weights of the model), and initialize them.
2. Create an placeholder x
3. Build the computation graph

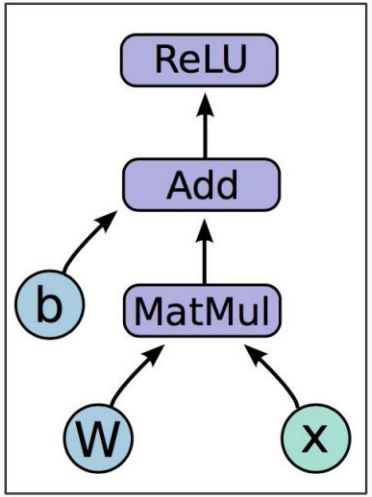
```
import tensorflow as tf

# Create Variables W and b
b = tf.Variable(tf.zeros((100,)))
W = tf.Variable(tf.random_uniform((784, 100), -1, 1))

# Create a placeholder that will take input matrices
# of size 100x784
x = tf.placeholder("float", (100, 784))

# String the placeholders and Variables together to
# build the graph
h = tf.nn.relu(tf.matmul(x, W) + b)
```

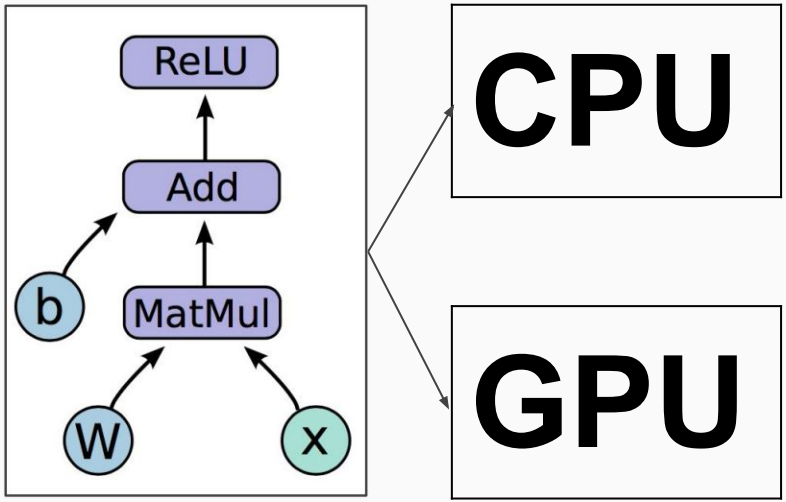
$$h = ReLU(Wx + b)$$



Running the graph

In the last slide, we successfully defined a **graph**.

To run it, we need to create a **session**, which is essentially a binding to a particular “device” (execution environment, like a CPU or GPU or both!)



Getting output

`sess.run(to_fetch, feed_dict)`

to_fetch: A list of graph nodes. You're telling Tensorflow to run as much of the graph needed to return the output of these nodes.

feed_dict: A dictionary used to "feed" values to the placeholders. The key is the placeholder object, and the value is the value you want to pass in.

```
import numpy as np
import tensorflow as tf

# Create Variables W and b
b = tf.Variable(tf.zeros((100,)))
W = tf.Variable(tf.random_uniform((784, 100), -1, 1))

# Create a placeholder that will take input matrices
# of size 100x784
x = tf.placeholder("float", (100, 784))

# String the placeholders and Variables together to
# build the graph
h = tf.nn.relu(tf.matmul(x, W) + b)

# Create a Tensorflow Session
with tf.Session() as sess:
    # Collect all the Variables in the graph and
    # returns an operation that will initialize them.
    sess.run(tf.global_variables_initializer())
    # Get the output of the op assigned to h
    sess.run(h, {x: np.random.random(100, 784)})
```

Summary

- We started by building a graph, and defined **Variables** for our weights and **placeholders** for our inputs.
- We then created a **session** to run the graph, and got the output of the single layer neural network.

- Next, we'll **train the model** (change the weights to optimize the loss)

Defining the loss

How do we pass in the correct **labels** to the graph? Use a **placeholder**!

Build a loss operation (node in the graph) that is a function of the **labels** and the model's **predictions**

```
predictions = tf.nn.softmax(...) # output of the neural net
labels = tf.placeholder("float", [100, 10])
# Categorical Cross-Entropy loss function
xentropy = tf.reduce_mean(
    -tf.reduce_sum(label * tf.log(prediction), axis=1))
```

Categorical Cross-Entropy

$$H_{y'}(y) = -\sum_{i} (y'_i \cdot log(y_i))$$

Standard loss function for multi-class classification. More details here.

Calculating the Gradients

```
# Creates an Optimizer object
optimizer = tf.train.GradientDescentOptimizer(0.5)
# Adds an optimization operation to the graph
train_op = optimizer.minimize(xentropy)
```

Tensorflow **graph nodes** (**operations**) have built-in **gradient operations**.

Tensorflow computes the gradient with respect to the **parameters/weights** with **backpropagation** automatically

Using the train_op in a model (example)

```
predictions = tf.nn.softmax(...)
labels = tf.placeholder(“float”, [None, 10])

xentropy = tf.reduce_mean(
    -tf.reduce_sum(label * tf.log(prediction)))

optimizer = tf.train.GradientDescentOptimizer(0.5)
train_op = optimizer.minimize(xentropy)
```

Training the model

`sess.run(train_op, feed_dict)`

1. Create a session
2. Build a train loop over the data
3. Run the `train_op`

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    # Loop 1000 times, so take 1000 “train steps”
    for i in range(1000):
        batch_x, batch_label = data_gen.next_batch()
        train_loss, _ = sess.run(
            [loss, train_op],
            feed_dict={x: batch_x, label:batch_label})
```

Summary

Steps for building a model in Tensorflow:

1. Construct the graph.
   a. **Variables** for weights, **placeholders** for inputs / labels
   b. Define the l**oss function.**
   c. Create the **training op**.
2. Initialize a session to run the graph in
3. Set up train loop, and train with `session.run(train_step, feed_dict)`

Part II: Live Coding Example

Part III: Using TensorBoard

What’s TensorBoard?

- A program bundled with Tensorflow to visualize the model’s training curve, graph, and more.

TensorBoard SCALARS IMAGES AUDIO GRAPHS DISTRIBUTIONS HISTOGRAMS EMBEDDINGS TEXT

Write a regex to create a tag group

Show data download links

Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing

0.6

Horizontal Axis

STEP RELATIVE WALL

Runs

Write a regex to filter runs

siamese_bilstm/00/train

siamese_bilstm/00/val

siamese_bilstm_keep_0.75/00/train

siamese_bilstm_keep_0.75/00/val

siamese_matching_bilstm/00/train

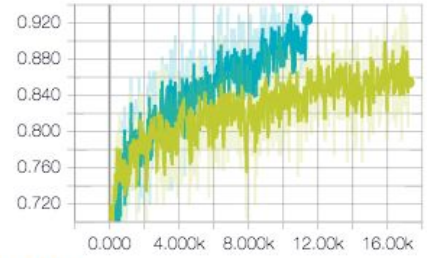
siamese_matching_bilstm/00/val

TOGGLE ALL RUNS

logs

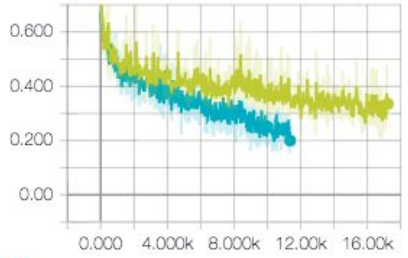
train_summaries 2

train_summaries/accuracy



run to download CSV JSON

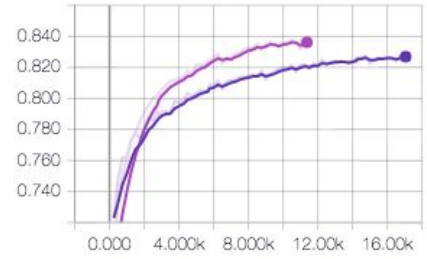
train_summaries/loss



run to download CSV JSON

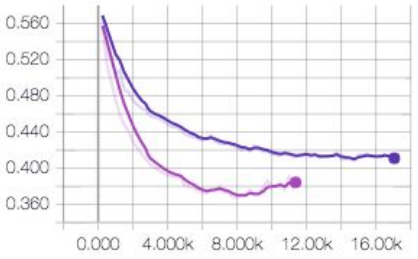
val_summaries 2

val_summaries/accuracy

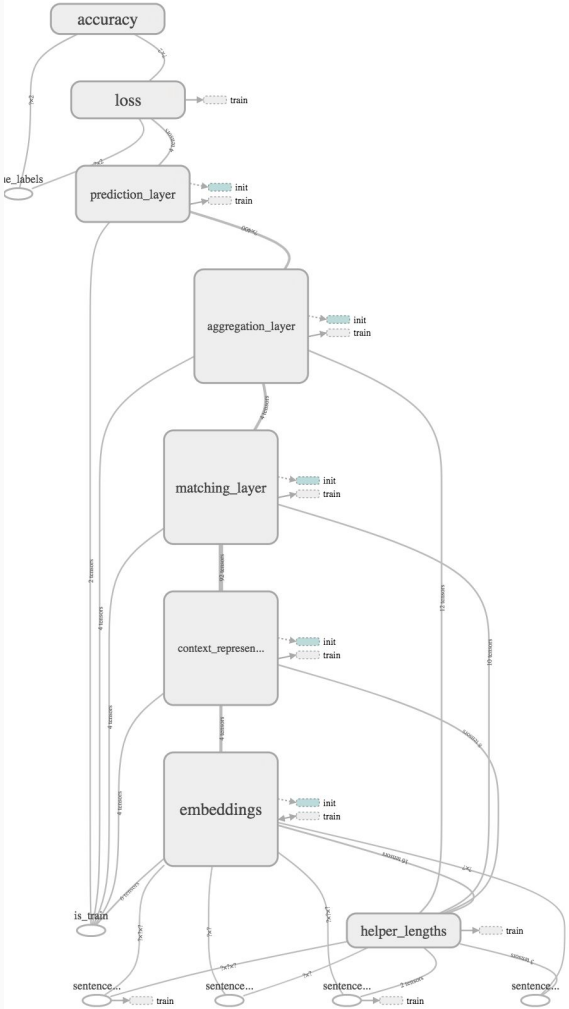


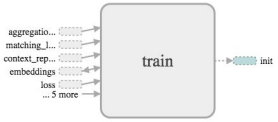
run to download CSV JSON

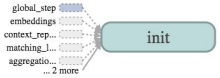
val_summaries/loss



run to download CSV JSON


accuracy
loss
train
prediction_layer
init
train
aggregation_layer
init
train
matching_layer
init
train
context_represen...
init
train
embeddings
init
train
helper_lengths
train
sentence...
train
sentence...
sentence...
train
sentence...


aggregatio...
matching_l...
context_rep...
embeddings
loss
... 5 more
train
init


global_step
embeddings
context_rep...
matching_l...
aggregatio...
... 2 more
init


global_step
init
train

Logging stats to Tensorboard (pt. 1): Summaries

```
tf.summary.scalar(“name”,value)
tf.summary.merge_all()
```

1. Create your graph
2. Add ops for statistics you want to log to tensorboard (e.g. loss, accuracy)
3. Create summary values for these ops, depending on their type (scalar for float)
4. Merge the various summary values into one summary op

```
# Statistics that we want to log
loss = ...
accuracy = ...

# Create summary values
tf.summary.scalar(“loss”, loss)
tf.summary.scalar(“acc”, accuracy)
summary_op = tf.summary.merge_all()
```

The global_step

Many Tensorflow programs have a `global_step` variable that is used to keep track of how many steps a model has seen.

Needs to be set to `trainable=False` -- why?

```
#  Create the global_step variable, and make it untrainable
tf.Variable(0, name="global_step", trainable=False)
... build your graph here ...
# Pass the global_step variable to the Optimizer
# which will keep track of how many gradient updates it has done.
opt = tf.train.AdamOptimizer(global_step=global_step)
```

Why is `global_step` useful?

Big Aspect: Keeping track of which Tensorboard summaries correspond to what step.

Think of the global_step as representing the time step that you're at. Tensorboard simply plots a point (the summary value) at that time step.

But what if you have to save a model, then load it back? How do you know how many gradient updates the model has done?

The global_step will tell you, and it enables you to keep plotting Tensorboard data right where you left off

Logging stats to Tensorboard (pt. 2): global_step

```
tf.Variable(
    0, name="global_step",
    trainable=False)
```

1. Set up an untrainable `global_step` variable.
2. Pass it to the `Optimizer`, which will automatically handle incrementing it.
3. When you're running the model in the session, you can get it's value the same way you would evaluate any other op: `sess.run(global_step)`

```
# Create the global_step variable,
# and make it untrainable
tf.Variable(0, name="global_step", trainable=False)
... build your graph here ...

# Pass the global_step variable to the Optimizer
# which will keep track of how many gradient
# updates it has done.
opt = tf.train.AdamOptimizer(global_step=global_step)

# Set up a session
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for i in range(1000):
        global_step_loop = sess.run(global_step)
        train_loss, _ = sess.run(
            [loss, train_op],
            feed_dict={x: batch_x, label:batch_label})
        ... etc ...
```

Logging stats to Tensorboard (pt. 3): Writing the Summaries

```
writer = tf.summary.FileWriter(
    summaries_dir,
    session_variable.graph)
writer.add_summary(
    summary_output, global_step)
```

1. Set up a `tf.summary.FileWriter` object that will write the Tensorboard logs to a directory
2. Evaluate the summary op,and use `writer.add_summary` to write it to the disk

```
# Create the global_step variable
# Build the graph
# Set up summary op

log_period = 10
# Set up a session
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for i in range(1000):
        global_step_loop = sess.run(global_step)
        train_loss, _ = sess.run(
            train_op,
            feed_dict={x: batch_x, label:batch_label})
        if global_step_loop % log_period == 0:
            summary = sess.run(
                summary_op,
                feed_dict={x: batch_x,
                           label:batch_label})
            writer.add_summary(summary, global_step_loop)
```

Viewing the results

To start Tensorboard, run:

```
tensorboard --logdir=<summarywriter_output_directory>
```

And then navigate to http://localhost:6006 in your browser.

Organizing the graph layout

To group nodes of the graph with a similar function together in Tensorboard, you can surround them in:

```
with tf.name_scope(“grouping_name”):
    ...ops go here...
```

This makes the graph a lot more readable and functional.

Part IV:
Live Coding: Adding TensorBoard to our example model