CS 224S: TensorFlow Tutorial

Lecture and Live Demo

Pujun Bhatnagar
Rishabh Bhargava
Intro to Deep Learning Frameworks

- Scales machine learning code
- Computes gradients!
- Standardizes machine learning applications for sharing
- Zoo of Deep Learning frameworks available with different advantages, paradigms, levels of abstraction, programming languages, etc
- Interface with GPUs for parallel processing

In some ways, rightfully gives Deep Learning its name as a separate *practice*
What is TensorFlow?

- Open source software library for numerical computation using data flow graphs
- Originally developed by Google Brain Team to conduct machine learning research
- “Tensorflow is an interface for expressing machine learning algorithms, and an implementation for executing such algorithms”
Big idea: express a numeric computation as a graph.

- Graph nodes are **operations** which have any number of inputs and outputs
- Graph edges are **tensors** which flow between nodes
\[ h = ReLU(Wx + b) \]


\[ h = ReLU(Wx + b) \]

**Variables** are stateful nodes which output their current value. State is retained across multiple executions of a graph (mostly parameters)
Programming model

\[ h = ReLU(Wx + b) \]

**Placeholders** are nodes whose value is fed in at execution time

(inputs, labels, ...)
Programming model

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

Mathematical operations:

**MatMul**: Multiply two matrix values.

**Add**: Add elementwise (with broadcasting).

**ReLU**: Activate with elementwise rectified linear function.
In code,

1. Create weights, including initialization
   \[ W \sim \text{Uniform}(-1, 1); \quad b = 0 \]

2. Create input placeholder \( x \)
   \[ m \times 784 \text{ input matrix} \]

3. Build flow graph
   \[
   h = \text{ReLU}(Wx + b)
   \]

```python
import tensorflow as tf

b = tf.Variable(tf.zeros((100,)))
W = tf.Variable(tf.random_uniform((784, 100), -1, 1))
x = tf.placeholder(tf.float32, (100, 784))
h = tf.nn.relu(tf.matmul(x, W) + b)
```
But where is the graph?

New nodes are automatically built into the underlying graph!

tf.get_default_graph().get_operations():

- zeros/shape
- zeros/Const
- zeros
- Variable
- Variable/Assign
- Variable/read
- random_uniform/shape
- random_uniform/min
- random_uniform/max
- random_uniform/RandomUniform
- random_uniform/sub
- random_uniform/mul
- random_uniform
- Variable_1
- Variable_1/Assign
- Variable_1/read
- Placeholder
- MatMul
- add
- Relu == h

h refers to an op!
How do we run it?

So far we have defined a graph.

We can deploy this graph with a session: a binding to a particular execution context (e.g. CPU, GPU)
Getting output

`sess.run(fetches, feeds)`

**Fetches:** List of graph nodes. Return the outputs of these nodes.

**Feeds:** Dictionary mapping from graph nodes to concrete values. Specifies the value of each graph node given in the dictionary.

```python
import numpy as np
import tensorflow as tf

b = tf.Variable(tf.zeros((100,)))
W = tf.Variable(tf.random_uniform((784, 100), -1, 1))

x = tf.placeholder(tf.float32, (100, 784))
h = tf.nn.relu(tf.matmul(x, W) + b)

sess = tf.Session()
sess.run(tf.initialize_all_variables())
sess.run(h, {x: np.random.random(100, 784)})
```
So what have we covered so far?

We first built a **graph** using **variables** and **placeholders**

We then deployed the graph onto a **session**, which is the **execution environment**

Next we will see how to **train** the **model**
How do we define the loss?

Use **placeholder** for labels

Build loss node using labels and **prediction**

```python
prediction = tf.nn.softmax(...)  # Output of neural network
label = tf.placeholder(tf.float32, [100, 10])

cross_entropy = -tf.reduce_sum(label * tf.log(prediction), axis=1)
```
How do we compute Gradients?

train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

- tf.train.GradientDescentOptimizer is an Optimizer object
- tf.train.GradientDescentOptimizer(lr).minimize(cross_entropy) adds optimization operation to computation graph

TensorFlow graph nodes have attached gradient operations

Gradient with respect to parameters computed with backpropagation...automatically
Creating the train_step op

prediction = tf.nn.softmax(...)
label = tf.placeholder(tf.float32, [None, 10])

cross_entropy = tf.reduce_mean(-tf.reduce_sum(label * tf.log(prediction),
reduction_indices=[1]))

train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
Training the Model

1. Create Session
2. Build training schedule
3. Run train_step

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

for i in range(1000):
    batch_x, batch_label = data.next_batch()
    sess.run(train_step, feed_dict={x: batch_x,
                                         label: batch_label})
```
Variable sharing: naive way

```python
variables_dict = {
    "weights": tf.Variable(tf.random_normal([782, 100]), name="weights"),
    "biases": tf.Variable(tf.zeros([100]), name="biases")
}
```

Not good for encapsulation!
tf.variable_scope() provides simple name-spacing to avoid clashes

tf.get_variable() creates/accesses variables from within a variable scope

```python
with tf.variable_scope("foo"):
  v = tf.get_variable("v", shape=[1])  # v.name == "foo/v:0"

with tf.variable_scope("foo", reuse=True):
  v1 = tf.get_variable("v")  # Shared variable found!

with tf.variable_scope("foo", reuse=False):
  v1 = tf.get_variable("v")  # CRASH foo/v:0 already exists!
```
Live Demo: Learning the MNIST dataset
RNNs

Cell types:
- Vanilla
- LSTM
- GRU

RNN sequences

http://karpathy.github.io/2015/05/21/rnn-effectiveness/
RNN + Tensorflow

- Most calculations are performed on a per-batch basis.
- RNNs expects tensors of shape `[B, T, ..]` as input, where `B` is batch size, `T` is length in time of each input.
- **What if each sequence is not the same length `T`?**

- **Solution:** *Batching and Padding*
Imagine one of your sequences is of length 1000, but the average sequence length is 20.

- Pad all sequences to length 1000 -- huge waste of memory and time!
- Make batches of size 32, and pad all examples in the batch to maximum sequence length in batch. Only one batch suffers!
- Padding: Appending 0’s to shorter sequences in a batch to make them equal length.

Dynamically unroll the RNN (example on next slide)
# Create input data
X = np.random.randn(2, 10, 8)

# The second example is of length 6
X[1,6:] = 0
X_lengths = [10, 6]

cell = tf.nn.rnn_cell.LSTMCell(num_units=64, state_is_tuple=True)

outputs, last_states = tf.nn.dynamic_rnn(
    cell=cell,
    dtype=tf.float64,
    sequence_length=X_lengths,
    inputs=X)

In Summary:

1. Build a graph
   a. Feedforward / Prediction
   b. Optimization (gradients and train_step operation)

2. Initialize a session

3. Train with session.run(train_step, feed_dict)
Questions?
Acknowledgments

Barak Oshri, Nishith Khandwala, CS224N CAs last quarter

Jon Gauthier, Natural Language Processing Group, Symbolic Systems

Bharath Ramsundar, PhD Student, Drug Discovery Research

Chip Huyen, Undergraduate, teaching CS20SI: TensorFlow for Deep Learning Research last quarter
Live Demo: Tackling MNIST using Tensorflow

Insert link to the ipython notebook
Link to the github repo: https://github.com/pbhatnagar3/cs224s-tensorflow-tutorial