Neural network programming: TensorFlow

Erik Spence

SciNet HPC Consortium

3 May 2018

Today's data, code and slides

You can get the slides for today's class at the SciNet Education web page.

`https://support.scinet.utoronto.ca/education`

Click on the link for the class, and look under "File Storage" and find the file "tensorflow.pdf".

SciNet compute • calcul CANADA

Today's class

This class will cover the following topics:

- TensorFlow constant tensors.
- Sessions.
- Variables and placeholders.
- TensorFlow neural network example.

Please ask questions if something isn't clear.

SciNet compute • calcul CANADA

TensorFlow

TensorFlow is Google's machine-learning framework.

- Released as open source in November 2015.
- The second-generation machine-learning framework developed internally at Google, successor to DistBelief.
- More flexible than some other neural network frameworks.
- Capable of running on multiple cores and GPUs.
- Provides APIs for Python, C++, Java and other languages.
- Can be quite slow.
- TensorBoard can be used for visualization of your TensorFlow graphs.

This framework continues to grow in popularity.

SciNet compute • calcul CANADA

Who cares?

This is a course about neural networks. Why do I need to know anything about TensorFlow?

- There are aspects to programming in Keras, which we will encounter later in the course, which only make sense if you have an underestanding of how Theano/TensorFlow works.
- In a nutshell, it's graph programming, rather than procedural programming.
- Seeing how things are done in raw TensorFlow will give you a greater appreciation for Keras.
- Some people choose to implement their networks in TensorFlow, so it's good to be familiar with the syntax.

Today we'll learn the basics of TensorFlow.

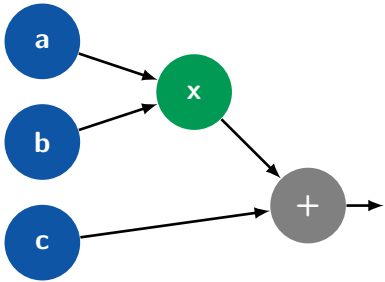
SciNet compute • calcul CANADA

The TensorFlow approach

In TensorFlow all calculations are treated as a graph. Whatever you are trying to do must be expressible in this form!

Operations ("ops") are nodes in a graph, data are edges.

If you want to calculate (a x b) + c, the graph will look like what's on the right.
It's important to remember that this is what's going on in the background, because it affects a number of things we're going to see today.



The TensorFlow approach, continued

In TensorFlow all calculations are treated as a graph.

- Everything is symbolic. This makes it powerful (we can do things like symbolic differentiation).
- The downside is that you have to change your thinking: regular procedural programming is the wrong approach.
- You can't run anything until your graph is built.
- This can also make it more difficult to debug.
- As with Python, there are two ways to operate: using scripts, or interactively.
- When we say "interactively", we mean both using the interactive Python prompt, and using an interactive TensorFlow session.

We'll use interactive to start, and show both options.

SciNet compute • calcul CANADA

TensorFlow variable types: constants

There are several types of variables. The first is the 'constant'.

It cannot be changed, so don't try.

Notice that you can't print the value of the variable, at least not in the usual interactive Python way.

Special functions are needed to print out the values of constants. These are add-ons which are generally not used.

```
In [1]:
In [1]: import tensorflow as tf
In [2]:
In [2]: x = tf.constant(1)
In [3]:
In [3]: x
Out[3]: <tf.Tensor 'Const:0' shape=() dtype=int32>
In [4]:
In [4]: print x
Tensor("Const:0", shape=(), dtype=int32)
In [5]:
In [5]: tf.contrib.util.constant_value(x)
Out[5]: array(1, dtype=int32)
In [6]:
```

TensorFlow variable types: constants

Constant lists will automatically be converted to 1D arrays.

Operations on TensorFlow objects are usually done using the TensorFlow operators.

Why can't you just print the value of a constant?

```
In [6]:
In [6]: y = tf.constant([1, 2, 3])
In [7]: z = tf.constant(range(3,6))
In [8]:
In [8]: y * z
Out[8]: <tf.Tensor 'mul:0' shape=(3,) dtype=int32>
In [9]:
In [9]: tf.multiply(y, z)
Out[9]: <tf.Tensor 'Mul:0' shape=(3,) dtype=int32>
In [10]:
```

TensorFlow sessions

Why can't we just print the value of constants? Because everything is a graph!

To get TensorFlow to do anything, we must run a session. This will evaluate the graph.

One way to do this is interactively using Python's "with" context manager.

```
In [10]:
In [10]: y = tf.constant([1, 2, 3])
In [11]: z = tf.constant(range(3, 6))
In [12]:
In [12]: answer = y * z
In [13]:
In [13]: with tf.Session() as sess:
    ...:     result = sess.run(answer)
    ...:     print result
    ...:
[ 3 8 15]
In [14]:
```

TensorFlow sessions, continued

Important note: you only need to 'run' the evaluation that you're interested in.

All dependent calculations will be automatically included.

Long chains of calculations can be constructed in this way.

You can run multiple chains of calculations using a single 'run' command.

```
In [14]: x = tf.constant(1)
In [15]: y = tf.constant(2)
In [16]: z = tf.constant(3)
In [17]:
In [17]: answer1 = 2 * x + y
In [18]: answer2 = answer1 * z
In [19]: answer3 = x + y + z
In [20]: answer4 = answer3 + 2
In [21]:
In [21]: with tf.Session() as sess:
    ...:     result = sess.run(answer2)
    ...:     result2 = sess.run([answer3, answer4])
    ...:     print result, result2
    ...:
12 [6, 8]
In [22]:
```

Interactive TensorFlow sessions

Rather than use "with", we can also set up an InteractiveSession. This is handy when running ipython, and doing development.

```
In [22]:
In [22]: sess = tf.InteractiveSession()
In [23]:
In [23]: y = tf.constant([1, 2, 3])
In [24]: z = tf.constant(range(3, 6))
In [25]:
In [25]: answer = y * z
In [26]:
In [26]: answer.eval()
Out[26]: array([ 3, 8, 15], dtype=int32)
In [27]:
In [27]: sess.close()
In [28]:
```

Interactive TensorFlow sessions, continued

Note: if you change a value in your chain of variables (your graph), you must redefine all variables which depend upon it.

It doesn't propagate the change automatically. Why?

```
In [28]: sess = tf.InteractiveSession()
In [29]:
In [29]: y = tf.constant([1, 2, 3])
In [30]: z = tf.constant(range(3, 6))
In [31]:
In [31]: answer1 = y * z
In [32]: z = tf.constant(range(6, 9))
In [33]: answer2 = y * z
In [34]:
In [34]: answer1.eval()
Out[34]: array([ 3, 8, 15], dtype=int32)
In [35]:
In [35]: answer2.eval()
Out[35]: array([ 6, 14, 24], dtype=int32)
In [36]: sess.close()
```

TensorFlow sessions within scripts

TensorFlow sessions can also be run from within scripts.

In this case the session should be closed explicitly.

Note that the print statement need not be within the session.

```
In [37]: exit
ejspence@mycomp code>
ejspence@mycomp code> python firstsession.py
[ 3 8 15]
ejspence@mycomp code>
```

```
# firstsession.py

import tensorflow as tf

y = tf.constant([1, 2, 3])
z = tf.constant(range(3, 6))
answer = y * z

# Open session.
sess = tf.Session()

# Run calculation and print.
result = sess.run(answer)
print result

# Close session.
sess.close()
```

TensorFlow variable types: placeholders

Placeholders are used to pass values from Python into TensorFlow.

The type of a placeholder must be specified.

If the placeholder is not a scalar, the shape must also be specified. If the shape is dynamic (you don't know it yet), then use 'None'.

In this case the shape will be determined at runtime.

```
In [1]:
In [1]: import tensorflow as tf
In [2]:
In [2]: a = tf.placeholder(tf.float64, [None, 5])
In [3]:
In [3]: a.shape
Out[3]: TensorShape([Dimension(None), Dimension(5)])
In [4]:
In [4]: a.shape.as_list()
Out[4]: [None, 5]
In [5]:
```

TensorFlow variable types: placeholders continued

The values are passed to the placeholders at runtime.

```
In [5]:
In [5]: import numpy.random as npr
In [6]:
In [6]: a.shape.as_list()
Out[6]: [None, 5]
In [7]:
In [7]: b = tf.placeholder(tf.float64, [1, 5])
In [8]:
In [8]: with tf.Session() as sess:
   ...:     result = sess.run(a + b, {a: npr.random([2, 5]), b: npr.random([1, 5])})
   ...:     print result
   ...:
[[ 1.19712866 0.93481508 0.88437654 0.55079983 0.86196034]
 [ 1.02124209 1.13603451 0.5930084 0.67743547 0.40674412]]
In [9]:
```

Placeholder notes

Some notes about the last slide.

- The values are passed to the placeholder at runtime.
- The values are passed using a dictionary as an argument to sess.run. The keys of the dictionary are the variables, the values are the values that passed.
- The dictionary is called a "feed dictionary".
- You can also use the syntax: sess.run(a + b, feed_dict = {...}).
- Note that you must pass all values which are needed anywhere in the chain of variables and functions, not just the one specified in sess.run.

TensorFlow variable types: Variables

Variables are in-memory buffers to hold parameters. Unlike Tensors, their values can change.

Variables must be explicitly initialized at runtime, unless they are implicitly initialized by assignment.

```
In [9]:
In [9]: a = tf.constant(2)
In [10]:
In [10]: b = tf.Variable(3)
In [11]:
In [11]: with tf.Session() as sess:
    ...:     print sess.run(a)
    ...:     sess.run(tf.global_variables_initializer())
    ...:     print sess.run(b)
    ...:
2
3
In [12]:
```

Tensorflow variable types: Variables, continued

As with Tensors and constants, Tensorflow will not cast variables for you. Make sure your data types are consistent!

```
In [12]:
In [12]: a = tf.ones([2, 2])
In [13]:
In [13]: a.dtype
Out[13]: tf.float32
In [14]:
In [14]: b = tf.Variable(np.ones([2, 2]))
In [15]:
In [15]: b.dtype
Out[15]: tf.float64_ref
In [16]:
In [16]: b = tf.Variable(np.ones([2, 2], dtype = float32))
In [17]:
In [17]: b.dtype
Out[17]: tf.float32_ref
In [18]:
```

TensorFlow variable types: Variables, continued more

We change values of variables using the tf.assign(b, a) function. It returns the variable being updated.

```
In [18]:
In [18]: a.dtype
Out[18]: tf.float32
In [19]:
In [19]: b.dtype
Out[19]: tf.float32_ref
In [20]:
In [20]: c = tf.assign(b, np.array([[1, 0], [3, 4]]) + a)
In [21]:
In [21]: tf.Session().run(c)
Out[21]:
array([[ 2., 1.],
       [ 4., 5.]], dtype=float32)
In [22]:
```

SciNet compute • calcul CANADA

Variables revisited

There is a second way to declare variables: tf.get_variable().

If a variable of that name already exists, it is returned. Otherwise a new variable of that name is created, and initialized randomly.

This allows you to share variables between programs. Be careful when you use this: all the usual dangers associated with the use of global variables apply.

```
In [22]:
In [22]: a = tf.get_variable(name = "a", shape = [])
In [23]:
In [23]: print a.name
a:0
In [24]:
In [24]: with tf.Session() as sess:
    ...:     sess.run(tf.global_variables_initializer())
    ...:     print a.eval()
    ...:
1.08962
In [25]:
```

TensorFlow goodies

TensorFlow contains all the NN goodies you've come to expect:

- All of the usual neural network layers, cost functions, activation functions, etc, are available.
- an object detection API
- data preprocessing facilities (tf.Transform)
- TensorFlow Fold which allow data of different sizes to be worked on simultaneously.
- Many example networks of various kinds.
- Several standard data sets are available.

However, using Tensorflow's neural network infrastructure is a bit more complicated.

SciNet compute • calcul CANADA

TensorFlow example

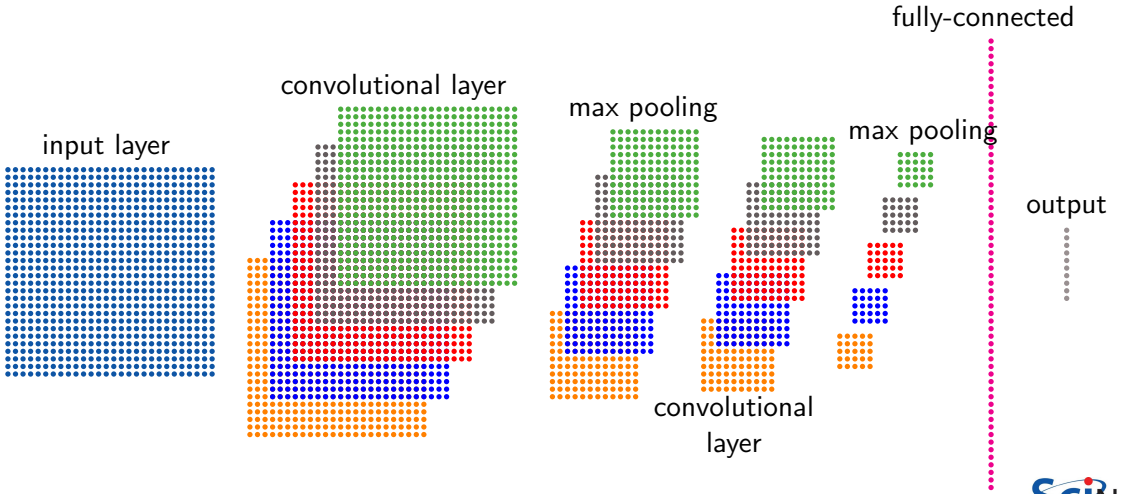
Let's re-implement our last network using TensorFlow, and apply it to the MNIST data set.

- Each image is 28 x 28. Let the input layer consist of our 2D images. The data set is available from within Tensorflow itself.
- The output will consist of a one-hot-encoding of the networks analysis of the input data. This means that, if the input image depicts a '7', the output vector should be $[\mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{1}, \mathbf{0}, \mathbf{0}]$.
- Thus, let there be 10 output nodes, one for each possible digit.
- We will, as per last class, have two convolutional/max-pooling combinations, as well as a single fully-connected hidden layer.

This will be the same network we implemented towards the end of last class.

SciNet compute • calcul CANADA

Our best network



TensorFlow, example, the code

```
# tf_cnn.py
import tensorflow as tf
import numpy as np

def cnn_model(features, labels, mode):

  input_layer = tf.reshape(features["x"],
    [-1, 28, 28, 1])
  conv1 = tf.layers.conv2d(inputs = input_layer,
    filters = 20, kernel_size = [5, 5],
    activation = tf.nn.relu)
  pool1 = tf.layers.max_pooling2d(inputs = conv1,
    pool_size = [2, 2], strides = 2)
  conv2 = tf.layers.conv2d(inputs = pool1,
    filters = 40, kernel_size = [3, 3],
    activation = tf.nn.relu)
  pool2 = tf.layers.max_pooling2d(inputs = conv2,
    pool_size = [2, 2], strides = 2)
```

```
# tf_cnn.py, continued

pool2_flat = tf.reshape(pool2, [-1, 5 * 5 * 40])
  [-1, 5 * 5 * 40])
dense = tf.layers.conv2d(inputs = pool2_flat,
  units = 100, activation = tf.nn.tanh)
output_layer = tf.layers.conv2d(inputs = dense,
  units = 10)

predictions = {
  "classes": tf.argmax(input = output, axis = 1),
  "probs": tf.nn.softmax(input = output_layer)}

loss = tf.losses.sparse_softmax_cross_entropy(
  labels = labels, logits = output)
```

SciNet compute • calcul CANADA

TensorFlow, example, the code, continued

Some notes:

- The behaviour of the function is modified by the 'mode', one of the arguments to the function.
- A dimension of -1 indicates an unknown dimension, which will be supplied at runtime.
- An 'Estimator' is created to do the actual training.

```
# cnn_model function, continued

  if mode == tf.estimator.ModeKeys.TRAIN:
    optimizer = tf.train.GradientDescentOptimizer(
      learning_rate = 0.001)
    train_op = optimizer.minimize(loss = loss,
      global_step = tf.train.get_global_step())
    return tf.estimator.EstimatorSpec(mode = mode,
      loss = loss, train_op = train_op)

  eval_metrics = {"accuracy": tf.metrics.accuracy(
    labels = labels,
    predictions = predictions["classes"])}
```

SciNet compute • calcul CANADA

TensorFlow, example, the code, continued more

```
# tf_cnn.py, continued
def main(unused_argv):
  # Get the data.
  mnist = tf.contrib.lear.datasets.load_dataset("mnist")
  x_train = mnist.train.images;     x_test = mnist.test.images
  y_train = np.asarray(mnist.train.labels, dtype = np.int32)
  y_test = np.asarray(mnist.test.labels, dtype = np.int32)

  # The Estimator.
  mnist_classifier = tf.estimator.Estimator(model_fn = cnn_model, model_dir = "/tmp/mnist_model")
  train_fn = tf.estimator.inputs.numpy_input_fn(x = {"x": x_train}, y = y_train,
    batch_size = 100, num_epochs = None, shuffle = True)
  mnist_classifier.train(input_fn = train_fn, steps = 10000)
  train_fn = tf.estimator.inputs.numpy_input_fn(x = {"x": x_test}, y = y_test,
    num_epochs = 1, shuffle = False)

  print mnist_classifier.evaluate(input_fn = eval_fn)
if __name__ == "__main__": tf.app.run()
```

TensorFlow example, continued

```
ejspence@mycomp ~>
ejspence@mycomp ~> python tf_cnn.py
INFO:tensorflow:loss = 2.35316, step = 1
INFO:tensorflow:global_step/sec:  15.8989
INFO:tensorflow:loss = 2.30593, step = 101 (6.290 sec)
INFO:tensorflow:global_step/sec:  18.0232
INFO:tensorflow:loss = 2.26787, step = 201 (5.548 sec)
INFO:tensorflow:global_step/sec:  17.2351
⋮
INFO:tensorflow:Loss for final step:  0.135627.
INFO:tensorflow:Starting evaluation at 2018-05-03-00:40:43
INFO:tensorflow:Restoring parameters from /tmp/mnist_model/model.ckpt-10000
INFO:tensorflow:Finished evaluation at 2018-05-03-00:40:45
INFO:tensorflow:Saving dict for global step 10000:  accuracy = 0.9334, global_step = 10000, loss =
0.246652
'loss':  0.24665214, 'global_step':  10000, 'accuracy':  0.93339998
ejspence@mycomp ~>
```

Linky goodness

There are tonnes of TensorFlow resources out there. These I've found useful.

- https://www.tensorflow.org
- https://www.tensorflow.org/tutorials
- https://github.com/vahidk/EffectiveTensorflow
- https://learningtensorflow.com
- http://eneskemalergin.github.io/blog/TensorFlow_Tutorial_1
- https://www.datacamp.com/community/tutorials/tensorflow-tutorial
- http://leotam.github.io/general/2016/03/13/DistributedTF.html

SciNet compute • calcul CANADA