Tensorflow

Çağrı Utku Akpak

March 1, 2018

Outline

What is tensorflow?

- TensorFlow is a deep learning library recently open-sourced by Google
- TensorFlow provides primitives for defining functions on tensors and automatically computing their derivatives.
- Installation : `https://www.tensorflow.org/install/`

Tensor Values

- The central unit of data in TensorFlow is the tensor.
- A tensor consists of a set of primitive values shaped into an array of any number of dimensions.
- **Rank:** # of dimensions
- **Shape:** Length along each dimension

Example

3. a rank 0 tensor; a scalar with shape []
[1., 2., 3.] a rank 1 tensor; a vector with shape [3]
[[1., 2., 3.], [4., 5., 6.]] a rank 2 tensor; a matrix with shape [2, 3]
[[[1., 2., 3.]], [[7., 8., 9.]]] a rank 3 tensor with shape [2, 1, 3]

Outline

Graph

- A computational graph is a series of TensorFlow operations arranged into a graph. The graph is composed of two types of objects:
  - Operations (or "ops"): The nodes of the graph. Operations describe calculations that consume and produce tensors
  - Tensors: The edges in the graph. These represent the values that will flow through the graph. Most TensorFlow functions return tf.Tensors.

Graph Example

```
a = tf.constant(3.0, dtype=tf.float32)
b = tf.constant(4.0) # also tf.float32 implicitly
total = a + b
print(a)
print(b)
print(total)
```

This will produce:

```
Tensor("Const:0", shape=(), dtype=float32)
Tensor("Const_1:0", shape=(), dtype=float32)
Tensor("add:0", shape=(), dtype=float32)
```

Outline

Session

- To evaluate tensors, instantiate a tf.Session object, informally known as a session.
- A session encapsulates the state of the TensorFlow runtime, and runs TensorFlow operations.

Example

```
sess = tf.Session()
print(sess.run(total))
```

prints: `7.0`

Session Example

```
vec = tf.random_uniform(shape=(3,))
out1 = vec + 1
out2 = vec + 2
print(sess.run(vec))
print(sess.run(vec))
print(sess.run((out1, out2)))
```

This will change value everytime you ran it. But it will produce something like this:

```
[scale=0.5]
[ 0.52917576  0.64076328  0.68353939]
[ 0.66192627  0.89126778  0.06254101]
(
  array([ 1.88408756,  1.87149239,  1.84057522], dtype=float3
  array([ 2.88408756,  2.87149239,  2.84057522], dtype=float3
)
```

Outline

Feeding

- A graph can be parameterized to accept external inputs, known as placeholders.
- A placeholder is a promise to provide a value later, like a function argument.

Example

```
x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
z = x + y
# To evaluate
print(sess.run(z, feed_dict={x: 3, y: 4.5}))
print(sess.run(z, feed_dict={x: [1, 3], y: [2, 4]}))
```

This will result:

```
7.5
[ 3.  7.]
```

Datasets

- Placeholders work for simple experiments, but Datasets are the preferred method of streaming data into a model.
- To get a runnable tf.Tensor from a Dataset you must first convert it to a tf.data.Iterator, and then call the Iterator's `get_next` method.
- The simplest way to create an Iterator is with the `make_one_shot_iterator` method. For example, in the following code the `next_item` tensor will return a row from the `my_data` array on each run call:

```
my_data = [
    [0, 1,],
    [2, 3,],
    [4, 5,],
    [6, 7,],
]
slices = tf.data.Dataset.from_tensor_slices(my_data)
next_item = slices.make_one_shot_iterator().get_next()
```

Datasets cont.

- Reaching the end of the data stream causes Dataset to throw an OutOfRangeError
- For example, the following code reads the `next_item` until there is no more data to read:

```
while True:
  try:
    print(sess.run(next_item))
  except tf.errors.OutOfRangeError:
    break
```

Layers

- Layers are the preferred way to add trainable parameters to a graph.
- Layers package together both the variables and the operations that act on them
- For example a densely-connected layer performs a weighted sum across all inputs for each output and applies an optional activation function. The connection weights and biases are managed by the layer object.

Outline

Creating Layers

- The following code creates a Dense layer that takes a batch of input vectors, and produces a single output value for each. To apply a layer to an input, call the layer as if it were a function.

Example

```
x = tf.placeholder(tf.float32, shape=[None, 3])
linear_model = tf.layers.Dense(units=1)
y = linear_model(x)
```

Outline

Initializing Layer

The layer contains variables that must be initialized before they can be used. While it is possible to initialize variables individually, you can easily initialize all the variables in a TensorFlow graph as follows:

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

Also note that this `global_variables_initializer` only initializes variables that existed in the graph when the initializer was created. So the initializer should be one of the last things added during graph construction.

Outline

Executing Layers

Now that the layer is initialized, we can evaluate the `linear_model`'s output tensor as we would any other tensor. For example, the following code:

```
print(sess.run(y, {x: [[1, 2, 3],[4, 5, 6]]}))
```

will generate a result similar to this:

```
[[-3.41378999]
 [-9.14999008]]
```

Layer Shortcuts

Not recommended.

```
x = tf.placeholder(tf.float32, shape=[None, 3])
y = tf.layers.dense(x, units=1)

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

print(sess.run(y, {x: [[1, 2, 3], [4, 5, 6]]}))
```

Outline

Define Data

Let's define some inputs, x, and the expected output for each input, y_true:

```
x = tf.constant([[1], [2], [3], [4]], dtype=tf.float32)
y_true = tf.constant([[0], [-1], [-2], [-3]], dtype=tf.float32)
```

Outline

Define Model

Next, build a simple linear model, with 1 output:

```
linear_model = tf.layers.Dense(units=1)

y_pred = linear_model(x)
```

To evalute:

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

print(sess.run(y_pred))
```

Result:

```
[[ 0.02631879]
 [ 0.05263758]
 [ 0.07895637]
 [ 0.10527515]]
```

Outline

Loss

- To optimize a model, you first need to define the loss.
- While you could do this manually with lower level math operations, the tf.losses module provides a set of common loss functions.
- In our example, we will use mean squared error:

```
loss = tf.losses.
mean_squared_error(labels=y_true, predictions=y_pred)

print(sess.run(loss))
```

Outline

Optimizers

- TensorFlow provides optimizers implementing standard optimization algorithms.
- These are implemented as sub-classes of tf.train.Optimizer
- They incrementally change each variable in order to minimizethe loss.
- The simplest optimization algorithm is gradient descent, implemented by tf.train.GradientDescentOptimizer. It modifies each variable according to the magnitude of the derivative of loss with respect to that variable.

Optimizer Example

```
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)
```

This code builds all the graph components necessary for the optimization, and returns a training operation. When run, the training op will update variables in the graph. You might run it as follows:

```
for i in range(100):
  _, loss_value = sess.run((train, loss))
  print(loss_value)
```

Output similar to:

```
1.35659
1.00412
0.759167
0.588829
0.470264
0.387626
...
```

Complete Program

```
x = tf.constant([[1], [2], [3], [4]], dtype=tf.float32)
y_true = tf.constant([[0], [-1], [-2], [-3]], dtype=tf.float32)

linear_model = tf.layers.Dense(units=1)

y_pred = linear_model(x)
loss = tf.losses.mean_squared_error(labels=y_true, predictions=y_pred)

optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)

init = tf.global_variables_initializer()

sess = tf.Session()
sess.run(init)
for i in range(100):
  _, loss_value = sess.run((train, loss))
  print(loss_value)

print(sess.run(y_pred))
```

Important Links

- Read low level APIs from `https://www.tensorflow.org/programmers_guide/`.
- Examine these examples from `https://github.com/nlintz/TensorFlow-Tutorials`
- More in depth explanation: `http://web.stanford.edu/class/cs20si/syllabus.html`. You can read slides up until Regression algorithms.