Automatic differentiation and gradient tape

Run in Google (https://colab.research.google.com/github/tensorflow/docs/blob/master/site/er Colab

In the previous tutorial we introduced `Tensors` and operations on them. In this tutorial we will cover automatic differentiation (https://en.wikipedia.org/wiki/Automatic_differentiation), a key technique for optimizing machine learning models.

Setup

```
import tensorflow as tf

tf.enable_eager_execution()
```

Gradient tapes

TensorFlow provides the tf.GradientTape (https://www.tensorflow.org/api_docs/python/tf/GradientTape) API for automatic differentiation - computing the gradient of a computation with respect to its input variables. Tensorflow "records" all operations executed inside the context of a `tf.GradientTape` (https://www.tensorflow.org/api_docs/python/tf/GradientTape) onto a "tape". Tensorflow then uses that tape and the gradients associated with each recorded operation to compute the gradients of a "recorded" computation using reverse mode differentiation (https://en.wikipedia.org/wiki/Automatic_differentiation).

For example:

```
x = tf.ones((2, 2))

with tf.GradientTape() as t:
  t.watch(x)
  y = tf.reduce_sum(x)
  z = tf.multiply(y, y)

# Derivative of z with respect to the original input tensor x
dz_dx = t.gradient(z, x)
for i in [0, 1]:
  for j in [0, 1]:
    assert dz_dx[i][j].numpy() == 8.0
```

You can also request gradients of the output with respect to intermediate values computed during a "recorded" **`tf.GradientTape`** (https://www.tensorflow.org/api_docs/python/tf/GradientTape) context.

```
x = tf.ones((2, 2))

with tf.GradientTape() as t:
  t.watch(x)
  y = tf.reduce_sum(x)
  z = tf.multiply(y, y)

# Use the tape to compute the derivative of z with respect to the
# intermediate value y.
dz_dy = t.gradient(z, y)
assert dz_dy.numpy() == 8.0
```

By default, the resources held by a GradientTape are released as soon as GradientTape.gradient() method is called. To compute multiple gradients over the same computation, create a `persistent` gradient tape. This allows multiple calls to the `gradient()` method. as resources are released when the tape object is garbage collected. For example:

```
x = tf.constant(3.0)
with tf.GradientTape(persistent=True) as t:
  t.watch(x)
  y = x * x
  z = y * y
dz_dx = t.gradient(z, x)  # 108.0 (4*x^3 at x = 3)
dy_dx = t.gradient(y, x)  # 6.0
del t  # Drop the reference to the tape
```

Recording control flow

Because tapes record operations as they are executed, Python control flow (using `if`s and `while`s for example) is naturally handled:

```
def f(x, y):
  output = 1.0
  for i in range(y):
    if i > 1 and i < 5:
      output = tf.multiply(output, x)
  return output

def grad(x, y):
  with tf.GradientTape() as t:
    t.watch(x)
    out = f(x, y)
  return t.gradient(out, x)

x = tf.convert_to_tensor(2.0)

assert grad(x, 6).numpy() == 12.0
assert grad(x, 5).numpy() == 12.0
assert grad(x, 4).numpy() == 4.0
```

Higher-order gradients

Operations inside of the `GradientTape` context manager are recorded for automatic differentiation. If gradients are computed in that context, then the gradient computation is recorded as well. As a result, the exact same API works for higher-order gradients as well. For example:

```
x = tf.Variable(1.0)  # Create a Tensorflow variable initialized to 1.0

with tf.GradientTape() as t:
  with tf.GradientTape() as t2:
    y = x * x * x
  # Compute the gradient inside the 't' context manager
  # which means the gradient computation is differentiable as well.
  dy_dx = t2.gradient(y, x)
d2y_dx2 = t.gradient(dy_dx, x)

assert dy_dx.numpy() == 3.0
assert d2y_dx2.numpy() == 6.0
```

```
WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/tensorflow/python
Instructions for updating:
Colocations handled automatically by placer.
```

Next Steps

In this tutorial we covered gradient computation in TensorFlow. With that we have enough of the primitives required to build and train neural networks.