tensorflow_simple_app

August 24, 2017

1 TensorFlow Simple App - Testing the API

1.1 Imports

```
In [20]: import tensorflow as tf
```

1.2 Some simple computational graphs

Tensorflow constants take no input and outputs the constant value they store, they are initialized automatically

```
In [21]: node1 = tf.constant(3.0, dtype=tf.float32)
         node2 = tf.constant(4.0) # also tf.float32 implicitly
         print(node1, node2)
```

```
Tensor("Const_4:0", shape=(), dtype=float32) Tensor("Const_5:0", shape=(), dtype=float32)
```

The constants are initialized automatically but the computational graphs are not evaluated automatically, for evaluating we need to create a Session object and run it, we can run multiple computational graphs at once:

```
In [22]: sess = tf.Session()
         print(sess.run([node1, node2]))
```

```
[3.0, 4.0]
```

Combining graphs/tensors together:

```
In [23]: node3 = tf.add(node1, node2)
         print("node3:", node3)
         print("sess.run(node3):", sess.run(node3))
```

```
node3: Tensor("Add_2:0", shape=(), dtype=float32)
sess.run(node3): 7.0
```

With SummaryWriter's we can write the output graph to a directory which then can be read and visualized by tensorboard.

```
In [24]: writer = tf.summary.FileWriter("output", sess.graph)
         print(sess.run(node3))
         writer.close()
```

```
7.0
```

Start tensorboard to visualize the computational graph with the following command: `tensorboard --logdir=output/`

The visualized graph:

Node3 Computational Graph

Notice that operations are nodes too.

The graph above is constant it will always be the same thing, to make things more useful and interesting we use parameterize our model/graphs by using **placeholders**, a placeholder is basically a promise that later a value will be provided. When creating graphs with placeholders it is a bit like a lambda which later gets evaluated with some input.

```
In [26]: a = tf.placeholder(tf.float32)
         b = tf.placeholder(tf.float32)
         adder_node = a + b  # + provides a shortcut for tf.add(a, b)
         print(sess.run(adder_node, {a: 3, b: 4.5}))
         print(sess.run(adder_node, {a: [1, 3], b: [2, 4]}))
```

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

Now we've seen constants and placeholders, placeholders allow us to parameterize our graph. For machine learning we typically require variables that can be modified and updated, not just constants, for example the bias and weights in a neural network would be variables. Tensorflow api conviniently calls these variables for simply "variables".

As mentioned earlier constants are initialized automatically and their values can never change but for variables we **must manually initialize them**, to initialize **all** variables we can use the following operation (must run it in a session for it to take action):

```
In [39]: W = tf.Variable([.3], dtype=tf.float32)
         b = tf.Variable([-.3], dtype=tf.float32)
```

```
x = tf.placeholder(tf.float32)
linear_model = W * x + b
init = tf.global_variables_initializer()
sess.run(init)
sess.close()
```

Python got a construct called "with xx as xx: statement" which is similar to a try-finally, python will automatically call the __exit__method of the object called with on, so instead of having to use sess.close() we can use with:

```
In [40]: with tf.Session() as sess:
             sess.run(init)
```

The run operation of session takes the following inputs (all but one argument are optional):

- fetches: The computational graph, kan också vara en lista av graphs, isåfall returneras en lista av värden
- feed_dict=None: Input values, (en pyton dict, i.e map), key kan vara en tensor eller en placeholder exemepelvis.
- options=None: Extra options, ex sätta på tracing
- run_metadata=None: Extra metadata

I exemplet nedan så kör vi vår computation graph `linear_model` och ger den input värde för placeholdern `x`:

```
In [41]: with tf.Session() as sess:
             sess.run(init)
             print(sess.run(linear_model, {x: [1, 2, 3, 4]}))
```

```
[ 0.          0.30000001  0.60000002  0.90000004]
```

Nu har vi en väldigt simpel linjär model med 2 variabler: W och b, men vi vet inte hur bra vår modell är, vi måste evaluera den så vi använder ytterligare en placeholder y för att kunna parameterisera modellen med "desired value", i.e label. Denna label kan sedan användas för att beräkna loss exempelvis genom sum of squared loss.

** Glöm inte att initialisera variabler, alltid! **

```
In [49]: y = tf.placeholder(tf.float32)
         squared_deltas = tf.square(linear_model - y)
         loss = tf.reduce_sum(squared_deltas)
         with tf.Session() as sess:
             sess.run(init)
             print(sess.run(loss, {x: [1, 2, 3, 4], y: [0, -1, -2, -3]}))
```

```
23.66
```

Notera att i exemplet ovan så är W och b variabler men vi gör ingen träning whatsoever så vi får ingen optimal resultat, vi kan assigna variablerna manuellt för att få optimalt resultat:

```
In [50]: with tf.Session() as sess:
             fixW = tf.assign(W, [-1.])
             fixb = tf.assign(b, [1.])
             sess.run([fixW, fixb])
             print(sess.run(loss, {x: [1, 2, 3, 4], y: [0, -1, -2, -3]}))
```

```
0.0
```

1.3 Learning

Poängen med machine learning är ju att maskinen själv ska hitta de optimala värdena, inte att vi ska behöva assigna dem. Enter, **training**:

Innan vi inspekterar hög-nivå API för tensorflow modeller och training som kallas **estimators** kan vi kolla på ett mer låg-nivå API: **optimizers**

Optimizers minimiserar loss-funktion genom att gradvis förändra variablernas värden i modellen, exempelvis kan optimizern använda sig av gradient descent algoritmen (backprop).

Det är väldigt enkelt att applicera en optimizer som finns färdigimplementerad i tensorflow om du redan har din computational graph med inkluderade variabler färdig.

Notera att du måste själva bestämma hur länge träningen ska pågå, i.e när du nå konvergens, eller om du helt enkelt använder ett fixed antal iterationer:

```
In [54]: with tf.Session() as sess:
             optimizer = tf.train.GradientDescentOptimizer(0.01)
             train = optimizer.minimize(loss)
             sess.run(init) # reset values to incorrect defaults.
             for i in range(1000):
                   sess.run(train, {x: [1, 2, 3, 4], y: [0, -1, -2, -3]})
             print(sess.run([W, b]))
```

```
[array([-0.9999969], dtype=float32), array([ 0.99999082], dtype=float32)]
```

Voila! ovan är exempel på simpel machine learning, där maskinen lär sig parametrarna W och b för att minimisera error.

1.4 Estimators

Estimators är ett hög-nivå API i tensorflow för att skapa machine learning modeller och computational graphs.

Estimators har en del inbyggda implementationer som kan användas direkt, ex LinearRegressor etc, men du kan även skapa dina egna custom estimators.

Estimators är typiskt parameteriserade med funktioner kallade `input_fn`, ex input till estimator.train är en funktion, input till estimator.evaluate är en funktion, etc.

Estimators låter dig även skapa custom modeller, där du själv definierar modellen genom en `model_fn`

```
In [ ]:
```