INTRODUCTION TO CONVOLUTIONAL NETWORKS USING TENSORFLOW

Jesús Fernández Bes, jfbes@ing.uc3m.es
8 de febrero de 2016
Install

What is Tensorflow?

Implementing Softmax Regression

Deep Convolutional Networks in Tensorflow

What else?
Create a virtual environment with anaconda (it takes some time)

$ conda update conda
$ conda create -n tensorflow python=2.7 anaconda

(tensorflow is the name of the environment, it can be whatever we want)

Activate our new environment, prompt changes to (tensorflow)$

$ source activate tensorflow

To deactivate the environment you have to write (do it at the end of the session)

$ source deactivate
Install tensorflow but only in the new environment (also takes time)

Ubuntu/Linux 64-bit, CPU only:

$ pip install --upgrade
https://storage.googleapis.com/tensorflow/linux/cpu/tensorflow-0.5.0-cp27-none-linux_x86_64.whl

Ubuntu/Linux 64-bit, GPU enabled:

$ pip install --upgrade
https://storage.googleapis.com/tensorflow/linux/gpu/tensorflow-0.5.0-cp27-none-linux_x86_64.whl

Mac OS X, CPU only:

$ pip install --upgrade
https://storage.googleapis.com/tensorflow/mac/tensorflow-0.5.0-py2-none-any.whl
Test the new installation

(tensorflow)$ python

>>> import tensorflow as tf
>>> hello = tf.constant('Hello, TensorFlow!')
>>> sess = tf.Session()
>>> print sess.run(hello)
Hello, TensorFlow!
>>> a = tf.constant(10)
>>> b = tf.constant(32)
>>> print sess.run(a + b)
42
>>>
Contents

Install

What is Tensorflow?

Implementing Softmax Regression

Deep Convolutional Networks in Tensorflow

What else?
Developed by Google Brain Team and Google’s Machine Intelligence research organization.

- Interface for expressing machine learning algorithms, and an implementation for executing them.
- Deep Flexibility: If you can write a computation graph.
- True Portability.
- Connect Research and Production.
- Auto-Differentiation.
- Language Options: Python, C++.
- Maximize Performance.
In *tensorflow* computation represented using **Graphs**.

Each node is an **operation (op)**.

Data is represented a **Tensors**.

**Op** takes **Tensors** and returns **Tensors**.

Variables maintain state across executions of the graph.

Two phases in the program:

Construct the computation **graph**.

Executes a **graph** in the context of a **Session**.

Feed/fetch data to/from the graph.
EXAMPLE OF COMPUTATION GRAPH
Always the same 3-steps pattern:

1. `inference()` - Builds the graph as far as is required for running the network forward to make predictions.

2. `loss()` - Adds to the inference graph the ops required to generate loss.

3. `training()` - Adds to the loss graph the ops required to compute and apply gradients.
Install

What is Tensorflow?

Implementing Softmax Regression

Deep Convolutional Networks in Tensorflow

What else?
Each image is 28 pixels by 28 pixels.
55,000 data points of training data (mnist.train)
10,000 points of test data (mnist.test)
5,000 points of validation data (mnist.validation).

Use tensorflow.googlesource.com/tensorflow/+/master/tensorflow/examples/tutorials/mnist/input_data.py to download the data.
\[ y = \text{softmax}(W x + b) \]

\[
\text{softmax}(x)_i = \frac{\exp x_i}{\sum_j \exp x_j}
\]
import tensorflow as tf
import input_data

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
Fichero descarga de datos:
tensorflow.googlesource.com/tensorflow/+\master/tensorflow/examples/tutorials/mnist/input_data.py

**Fichero descarga de datos:**
tensorflow.googlesource.com/tensorflow/+\master/tensorflow/examples/tutorials/mnist/input_data.py

**mnist_softmax.py: Variable declaration**

```python
x = tf.placeholder(tf.float32, [None, 784])
y_ = tf.placeholder(tf.float32, [None, 10])

W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
```
♦ Inference

```python
mnist_softmax.py: Inference

    y = tf.nn.softmax(tf.matmul(x, W) + b)
```

♦ Loss

```python
mnist_softmax.py: Loos computation

    cross_entropy = -tf.reduce_sum(y_*tf.log(y))
```
Training

mnist_softmax.py : Training

```python
train_step = tf.train.GradientDescentOptimizer(0.01)
    .minimize(cross_entropy)
```

TensorFlow also provides many other **optimization algorithms**: using one is as simple as tweaking one line:

```python
class tf.train.AdagradOptimizer
class tf.train.MomentumOptimizer
class tf.train.AdamOptimizer
```
How well we are predicting the correct label?

**EVALUATION**

```python
# mnist_softmax.py: Evaluation

correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
```
execute the computation graph

**mnist_softmax.py: Initialize all the variables**

```python
init = tf.initialize_all_variables()
```

**mnist_softmax.py: Start a new session**

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

# Let's train -- we'll run the training step 1000 times!
for i in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
print sess.run(accuracy, feed_dict={x: mnist.test.images,
                                      y_: mnist.test.labels})
```

Result around 91%: **VERY BAD for MNIST**
Install

What is Tensorflow?

Implementing Softmax Regression

Deep Convolutional Networks in Tensorflow

What else?
♦ State-of-the-art of Image Recognition.

♦ Traditional Approach: **Handmade features.**

![Diagram of a puppy and a "dog" label with a complex non-linear map from pixels to labels.](http://www.cs.umd.edu/~djacobs)

**Figura 1:** [http://www.cs.umd.edu/~djacobs](http://www.cs.umd.edu/~djacobs)

♦ ¿Can we learn the features + classifier?

**Complex non-linear map** from pixels to labels.
Do it in several *simple* layers: Function composition.
Figura 2: http://parse.ele.tue.nl/

Explanation of each layer.
Train using Backpropagation.
This works very well. Why?
Reuse the data loading part of `mnist_softmax.py`.

```python
import tensorflow as tf
import input_data

mnist = input_data.read_data_sets('MNIST_data', one_hot=True)

x = tf.placeholder("float", shape=[None, 784])
y_ = tf.placeholder("float", shape=[None, 10])
```
We will have to initialize a lot of weights.

**mnist_cnn.py: weight initialization**

```python
def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)

def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)
```
Convolution and pooling operations. We will use them in different layers.

**mnist_cnn.py: Convolution and pooling**

```python
def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                          strides=[1, 2, 2, 1], padding='SAME')
```

**First layer:** From input data to second layer

**mnist_cnn.py: First Convolutional Layer**

```python
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])

x_image = tf.reshape(x, [-1, 28, 28, 1])

h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)

h_pool1 = max_pool_2x2(h_conv1)
```

**Second layer:** From output of first layer to FC layer

```python
W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])

h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)
```
# Densely connected layer

\[ W_{fc1} = \text{weight\_variable}([7 \times 7 \times 64, 1024]) \]

\[ b_{fc1} = \text{bias\_variable}([1024]) \]

\[ h_{pool2\_flat} = \text{tf.\_reshape}(h_{pool2}, [-1, 7\times7\times64]) \]

\[ h_{fc1} = \text{tf.\_nn.\_relu}(\text{tf.\_matmul}(h_{pool2\_flat}, W_{fc1}) + b_{fc1}) \]
*Dropout* trick and output layer.

**mnist_cnn.py: Dropout**

```python
# Add dropout
keep_prob = tf.placeholder("float")
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
```

**mnist_cnn.py: Readout layer**

```python
# Readout layer
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])
y_conv = tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)
```
mnist_cnn.py: Train and Evaluate

cross_entropy = -tf.reduce_sum(y_*tf.log(y_conv))
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
correct_prediction = tf.equal(tf.argmax(y_conv,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
mnist_cnn.py: Execute

```python
init = tf.initialize_all_variables()
sess = tf.InteractiveSession()
sess.run(init)
for i in range(20000):
    batch = mnist.train.next_batch(50)
    if i%100 == 0:
        train_accuracy = accuracy.eval(feed_dict={
            x: batch[0], y_: batch[1], keep_prob: 1.0}
        )
        print "step %d, training accuracy %g"%(i, train_accuracy)
        train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.5})
print "test accuracy %g"%accuracy.eval(feed_dict={
    x: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0})
```

Go back to work while it finishes: Accuracy ≈ 99.2%.
Install

What is Tensorflow?

Implementing Softmax Regression

Deep Convolutional Networks in Tensorflow

What else?
WHAT ELSE?

♦ Tensorboard.

https://www.tensorflow.org/versions/0.6.0/how_tos/summaries_and_tensorboard/index.html

♦ Vector Representation of Words (word2vec).

♦ Recurrent Neural Networks (Long short-term memory Networks, seq2seq models).

♦ General Mathematics (Mandelbrot Set, Partial Differential Equations)

♦ Udacity free online course