Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Fabian Barteld, Benjamin Milde

TENSORFLOW

Overview

- Tensorflow Introduction / First session
- Regression models
- Neural tagger (DNN)
- Implementing Word2Vec
- Introduction to Tensorboard
- Neural tagger (LSTM)
- RNN language model (LSTM)
- Maybe: Convolutions

Introduction

Introduction

- TensorFlow started as DistBelief at Google Brain in 2011
- Publicly released as open source software on November 9, 2015
- Written in C++, Python bindings for rapid prototyping (best documented interface)
- Other bindings exist: Java, Scala, C, Rust, Go, Haskell, JavaScript, ...

Main concepts

- TensorFlow computations are expressed as stateful dataflow graphs
- Graphs contain operations (ops) on Tensors
- In TensorFlow lingu, a tensor is any n-d array. A scalar is a tensor of rank 0, a vector of rank 1, a matrix of rank 2.
- A Tensorflow rank is not the same as a matrix rank!
- The shape of a tensor with rank 2 is a tuple of dimensions, e.g. (128, 256) is a 128 x 256 matrix.
- Tensors of rank 3 are heavily used in feed forward networks, an example is a tensor with shape (128, 128, 256)

Layer based APIs vs. Graphs based

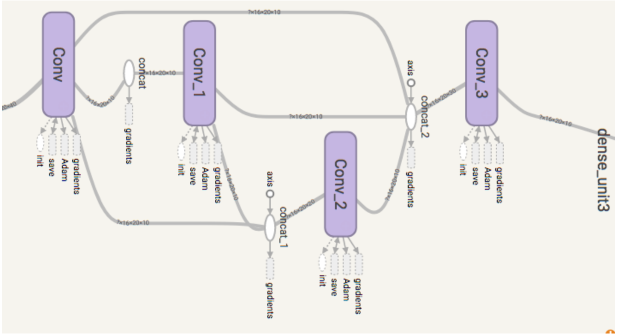
A typical layer API (not TensorFlow code):

```
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

This is fine (and very readable!) for models that can be described by stacking individual layers

Layer based APIs vs. Graphs based

Disadvantage: Difficult to express structures like these:

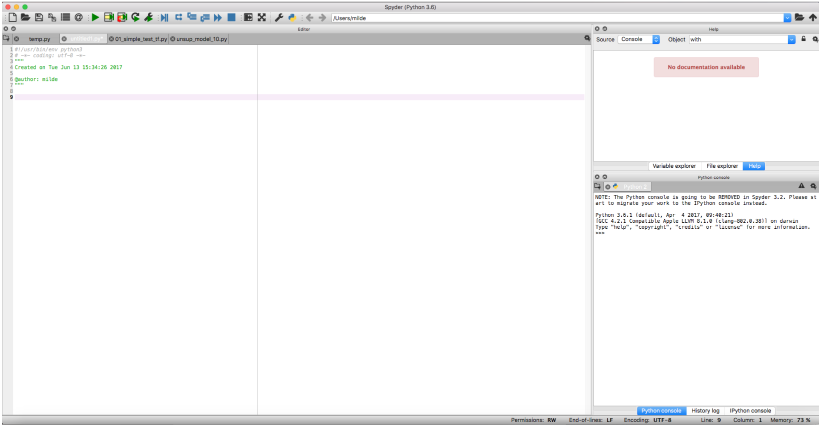


Increasing evidence that these kind of deeply connected networks are very useful.

Layer based APIs vs. Graphs based

- Since Tensorflow uses computation graphs, the declaration of the model allows for a higher expressivity
- Has a steeper learning curve in the beginning
- In the newer versions of tensorflow, you can also mix layer-like APIs with the computation graph
- We will focus on not using any short cuts, as this has a higher learning effect and only make use of standard ops in the beginning

First steps - Lets open spyder



First steps - Necessary imports

```python
import numpy as np
import tensorflow as tf
```

- Outside of graph computations, we usually store data in Numpy arrays.
- Numpy arrays are the main objects to transfer data to inputs of the graph and from outputs of the graph.
- Numpy arrays are also an abstraction for (homogeneous) multidimensional arrays.

Generating some random data

```
#some random test data
a_data = np.random.rand(256)
b_data = np.random.rand(256)
```

- Now a and b contain vectors of length 256 with random floats. E.g. print(a_data) returns:

```
[ 0.54976368   0.87790201   0.96528541  ...,
 0.05281365   0.48556404   0.46848266]
```

Declare the computation graph

```
#construct the graph
a = tf.placeholder(tf.float32, [256])
b = tf.placeholder(tf.float32, [256])

x = a+b
```

- The placeholders can later be used to input data to the computation graph
- The operation x = a+b does not immediatly add something, it creates a graph.
- In fact, print(x) returns:

```
Tensor("add:0", shape=(256,), dtype=float32)
```

A session on a computation device

```
with tf.device('/cpu'):
    with tf.Session() as sess:
        x_data = sess.run(x, {a: a_data, b: b_data})
        print(x_data)
```

- This fills the inputs a and b with a_data and b_data (our random data), runs the computation graph and retrieves the results of x in x_data
- Obviously not terrible useful as is, but you could run the operation easily on a gpu by changing tf.device('/cpu') to tf.device('/gpu:1'). Copying data to and from the GPU is handled automatically for you.

Small warm up exercise!

- We change a and b to random matrices:

```
a = np.random.rand(256, 128)
b = np.random.rand(128, 512)
```

- Calculate the resulting matrix of shape (256, 512) in TensorFlow.