Theory of Neural Networks

Architecture and Types of Layers:

- Fully Connected (FC)
- Convolutional Neural Network (CNN)
- Pooling
- Drop out
- Residual
- Recurrent
Some Frameworks

TensorFlow and Caffe are the big two.

There are many others:

- Scikit-learn: python module with ML algorithms
- Torch: a collection of algorithms connected via LuaJIT
- Theano: Fast linear algebra from python - architecturally a bit like TensorFlow
- Keras: Higher-level, more intuitive set of abstractions that acts as an interface to TensorFlow, Theano, and others

- Keras is great for quickly prototyping solutions.
- Windows users: Theano is better supported on Windows than TensorFlow.
Setting Up an Environment

Before installing any frameworks you need to consider what kind of environment you want to use.

- Native Environment:
  - Installed software libraries can be used by any programs on the local system.
  - Possibility of getting conflicts from other projects on the system.

- Virtual Environment:
  - Installed software libraries can only be used by programs in the virtual environment.
  - Keeps your environment isolated so software versions can't conflict with other projects.
Virtual Environment Example: Using Anaconda

Anaconda provides data science packages along with the Conda package manager and virtual environment manager:


Example of using Conda to create a virtual environment and installing python and tensorflow into that environment.

```
$ conda create -n tensorflow-env python=3.5

$ source activate tensorflow-env

(tensorflow-env) $ conda install tensorflow
```

- There are other virtual environment managers: (that I can't think of...)
- In Windows, remove "source" from command to activate: "activate tensorflow-env"
- "tensorflow-env" is the name for the environment: it can be any string.
Caffe or TensorFlow?

Caffe is:
- From Berkeley Vision and Learning Center (BVLC)
- A set of plug-and-play layers
- You can add your own
- Parallelism is supported 'after the fact'

TensorFlow is:
- From Google Brain
- Much more "flexible" and confusing
- You can wrap your own layers, but mostly you work in python
- Parallelism is supported in the core design

Both can use GPUs or CPUs.

Both have active development communities.
A Sample Dataset: MNIST

- 10k test images of hand-written digits
- Each 28x28 1-bit pixels
- Each is labeled with the 'correct' answer

Train A NN To Recognize MNIST

- This is a classification problem - given input, select from a finite set of possible interpretations.
- Divide the 10K samples into training set and a test set
  - ... because we can't test on the inputs we trained on - it might only recognize specific 28x28 arrays.
- Want a network with:
  - 28x28 inputs, all 1 or 0.
  - 10 outputs, "one-hot" (meaning only one non-zero)
Caffe provides a catalog of layers

- Vision related:
  - Convolution, Pooling, Local Response Normalization
- Activation/Neuron:
  - ReLU, Sigmoid, Etc.
- Loss:
  - Softmax, Sum-of-Squares, Etc.
- Data:
  - HDF5, In-memory, Etc.
- Misc:
  - Inner Product, Flatten, Etc.
- Special Purpose:
  - Segnet-provided layers
**A Caffe Layer Definition**

- This is a Google Protobuf language.
- For each filter block there is a 'layer' of a preexisting 'type', with set-up parameters.
- This code defines the first convolution step.
- Each layer type knows how to calculate its gradient.
layer {
    name: "conv1"
    type: "Convolution"
    bottom: "data"
    top: "conv1"
    param {
        lr_mult: 1
    }
    param {
        lr_mult: 2
    }
    convolution_param {
        num_output: 20
        kernel_size: 5
        stride: 1
        weight_filler {
            type: "xavier"
        }
        bias_filler {
            type: "constant"
        }
    }
}
Caffe Framework Takes Care of Learning

- The network shown learns to identify MNIST at about 98% accuracy
- Specify training parameters with another protobuf
- Training time: a few minutes
- GPU or CPU? Just 'solver mode: GPU'

TensorFlow is Very Different

- Everything is in Python
- Your code defines a directed graph
  - Edges are Tensors - really n-dimensional rectangular arrays
  - Nodes are Tensors Transformations
- The calculation doesn't happen when your code executes - only the setup. Calculation happens later when you explicitly execute the graph
- TensorFlow handles assigning graph elements to hardware and taking gradients (Free parallelism)
- Large catalog of graph elements, analogous to Caffe layers.
Execution happens later!

- If you say:

\[
\text{tensorA} = \text{tensorB} + \text{tensorC}
\]

- You mean that is it to build the following graph:

Tensorflow Example: Linear Regression

We can use tensorflow to find the best fitting line.

Using the linear equation \( Y = m \times X + b \) to build our model, we can feed \( X \) and \( Y \) (our data) into tensorflow and have it compute the values of \( m \) and \( b \) that satisfy the equation.
In [ ]:

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

x_data = np.random.rand(100).astype(np.float32)
y_data = x_data * 0.1 + 0.3  # some phony data

# Try to find values for W and b that compute y_data = W * x_data + b
W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.zeros([1]))
y = W * x_data + b

# Minimize the mean square errors
loss = tf.reduce_mean(tf.square(y - y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)

# Before starting, initialize the variables. We will 'run' this first.
init = tf.global_variables_initializer()  # create a 'graph' that does initialization

# Launch the graph.
sess = tf.Session()
sess.run(init)  # do initialization by running the graph above

# Fit the line.
for step in range(201):
    sess.run(train)
    if step % 20 == 0:
        print(step, sess.run(W), sess.run(b))

# Learns best fit is W: [0.1], b: [0.3]
```
A TensorFlow Layer Definition

- This would be analogous to the Caffe Convolution definition we wrote in Protobuf

```python
filter = tf.Variable(tf.random_normal([BLKSZ, 5, 5, 20]))
conv1_unbiased = tf.nn.conv2d(data, filter, strides=[1, 1, 1, 1], padding='VALID')
biases = tf.Variable(tf.random_normal([BLKSZ, 5, 5, 20]))
conv1 = tf.nn.bias_add(conv1_unbiased, biases)
```

TensorBoard!

- TensorFlow comes with a wonder tool for visualizing networks and their convergence
- To use TensorBoard, insert calls like scalar_summary(), histogram_summary(), or image_summary()
- Then use a SummaryWriter instance to serialize that data
- TensorBoard provides a web server that gives visualizations of the summaries as the run proceeds
TensorBoard shows up as http://localhost:6006/ (http://localhost:6006/)

Dynamic Images of the Calculation Graph