Tensorflow Basics

CS60010

Deep Learning Package Zoo

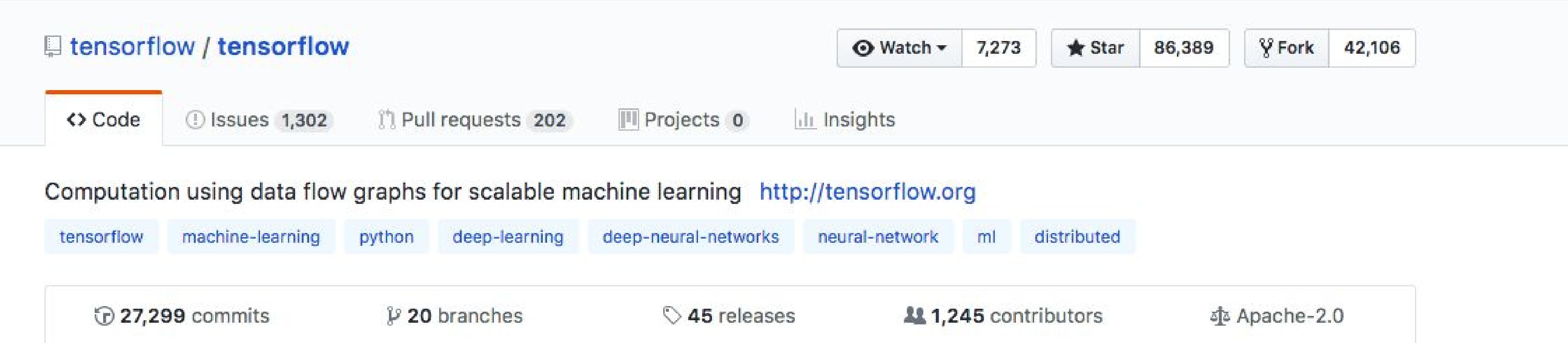
Caffe

and others ...

Deep Learning Frameworks

- Scales machine learning code
- Computes gradients!
- Standardise machine learning machine learning applications for sharing
- Zoo of deep learning advantages available with different advantages, levels of abstraction, programming languages, etc.
- Provides an interface with GPU for parallel processing

What is Tensorflow



Programming Model

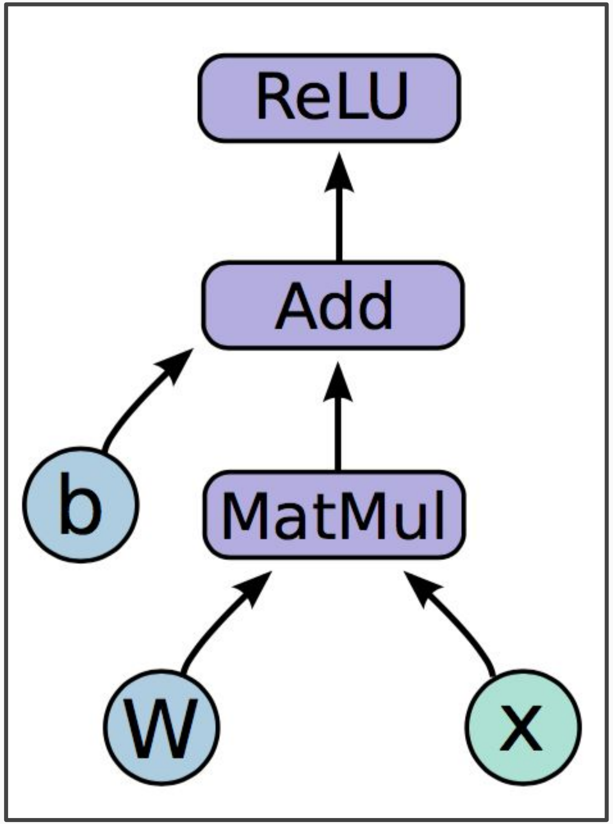
Idea: Express numeric computation as a *graph*

- Graph nodes are **operations** that can have any number of inputs and exactly one output
- Graph edges are **tensors** that flow between nodes

Tensors are n dimensional array

Programming model

$$h = ReLU(Wx + b)$$



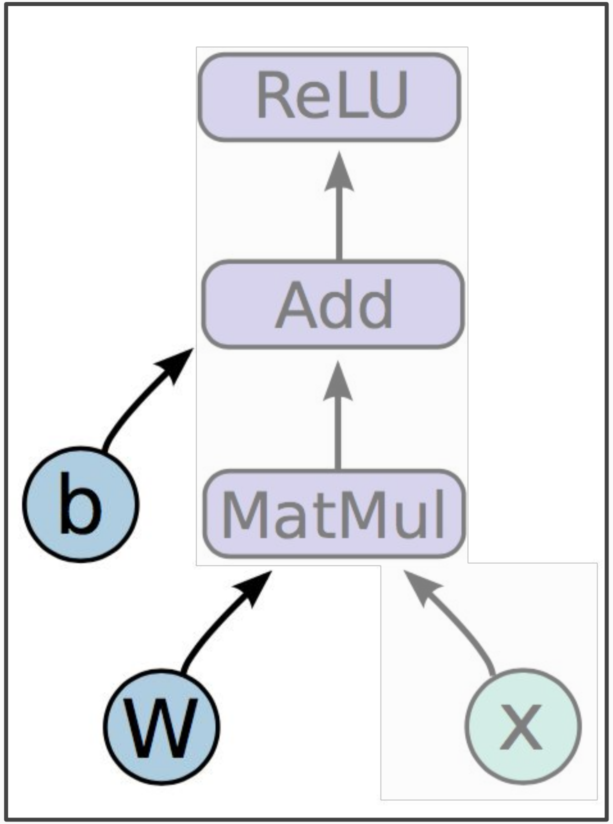
Programming model

$$h = ReLU(Wx + b)$$

Variables are stateful nodes which output their current value.

State is retained across multiple executions of a graph

(mostly parameters)

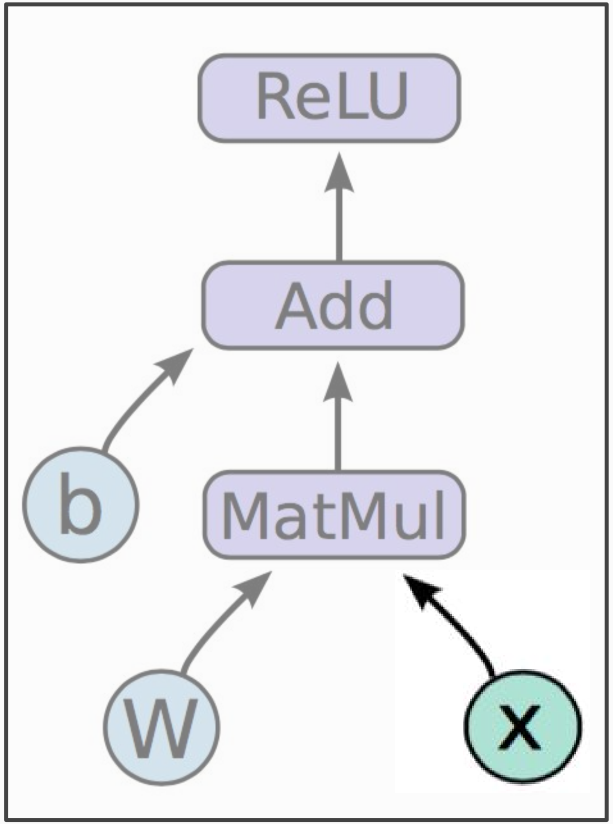


Programming model

$$h = ReLU(Wx + b)$$

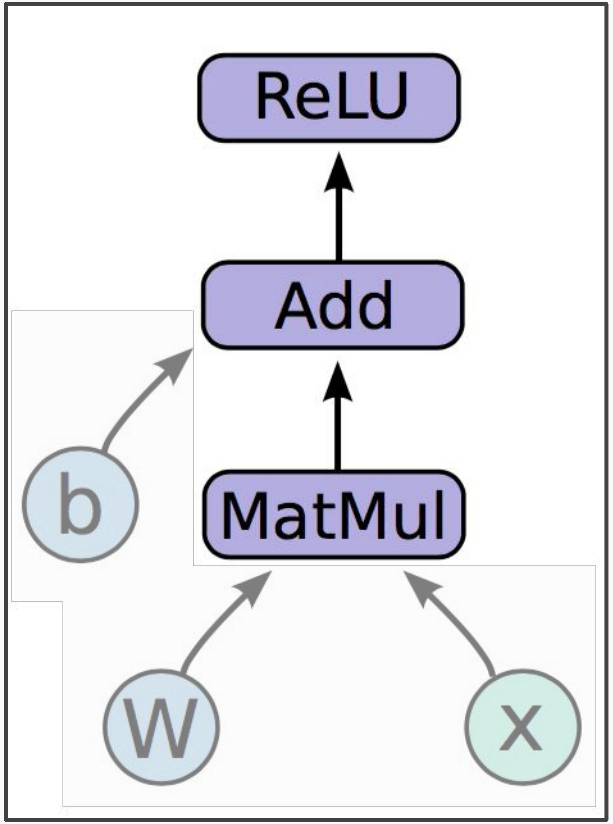
Placeholders are nodes whose value is fed in at execution time

(inputs, labels, ...)



Programming model

$$h = ReLU(Wx + b)$$



Mathematical operations:

MatMul: Multiply two matrix values.
Add: Add elementwise (with broadcasting).
ReLU: Activate with elementwise rectified linear function.

In code,

1. Create weights, including initialization

   W ~ *Uniform*(-1, 1); b = **0**

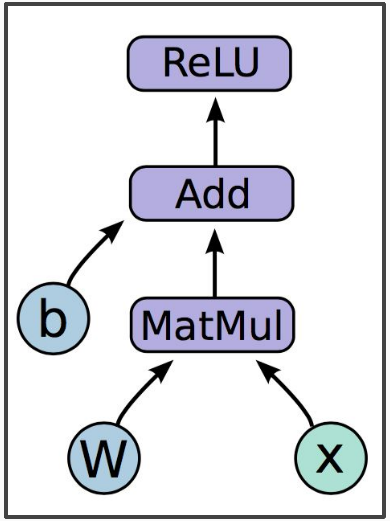
2. Create input placeholder x

   m * 784 input matrix

3. Build flow graph

```
import tensorflow as tf

b = tf.Variable(tf.zeros((100,)))
W = tf.Variable(tf.random_uniform((784, 100), -1, 1))

x = tf.placeholder(tf.float32, (100, 784))

h = tf.nn.relu(tf.matmul(x, W) + b)
```

$$h = ReLU(Wx + b)$$

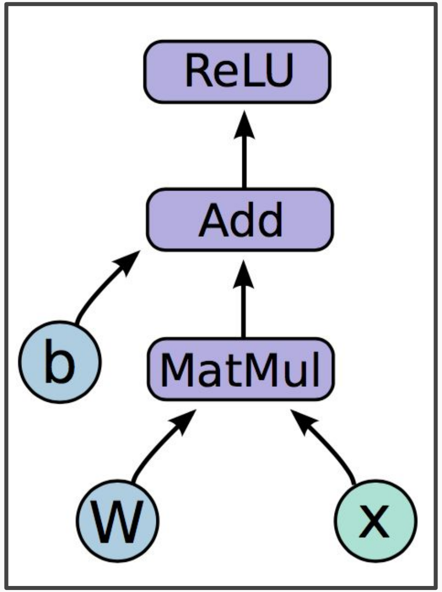


But where is the graph?

New nodes are automatically built into the underlying graph!
tf.get_default_graph().get_operations():

zeros/shape
zeros/Const
zeros
Variable
Variable/Assign
Variable/read
random_uniform/shape
random_uniform/min
random_uniform/max
random_uniform/RandomUniform
random_uniform/sub
random_uniform/mul
random_uniform
Variable_1
Variable_1/Assign
Variable_1/read
Placeholder
MatMul
add
Relu == h

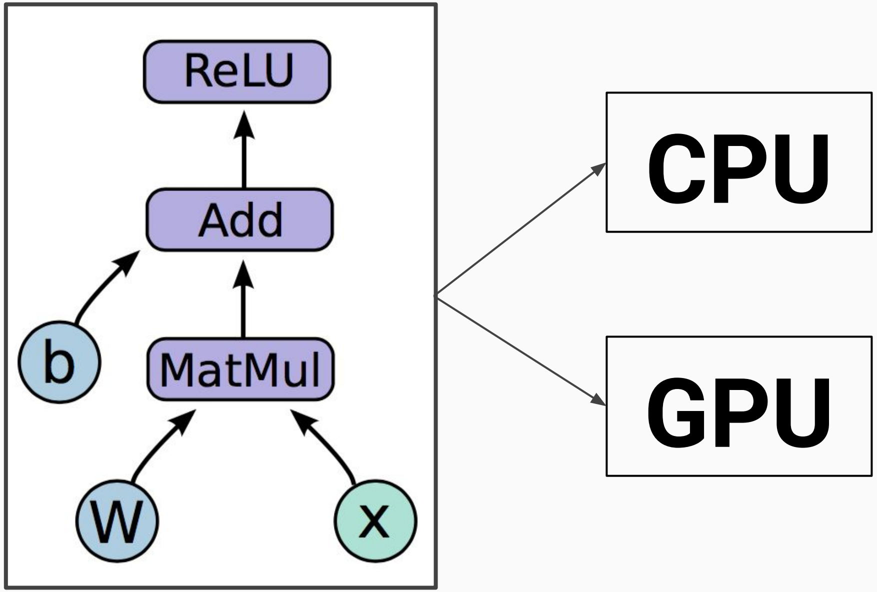
h refers to an op!



How do we run it?

So far we have defined a **graph**.

We can deploy this graph with a **session**: a binding to a particular execution context (e.g. CPU, GPU)



Getting output

```
sess.run(fetches, feeds)
```

Fetches: List of graph nodes. Return the outputs of these nodes.

Feeds: Dictionary mapping from graph nodes to concrete values. Specifies the value of each graph node given in the dictionary.

```
import numpy as np
import tensorflow as tf

b = tf.Variable(tf.zeros((100,)))
W = tf.Variable(tf.random_uniform((784, 100),
                -1, 1))

x = tf.placeholder(tf.float32, (100, 784))
h = tf.nn.relu(tf.matmul(x, W) + b)

sess = tf.Session()
sess.run(tf.initialize_all_variables())
sess.run(h, {x: np.random.random(100, 784)})
```

So what have we covered so far?

We first built a **graph** using **variables** and **placeholders**

We then deployed the graph onto a **session**, which is the **execution environment**

Next we will see how to **train** the **model**

How do we define the loss?

Use **placeholder** for **labels**

Build loss node using labels and **prediction**

```
prediction = tf.nn.softmax(...)  #Output of neural network
label = tf.placeholder(tf.float32, [100, 10])

cross_entropy = -tf.reduce_sum(label * tf.log(prediction), axis=1)
```

How do we compute Gradients?

```
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
```

`tf.train.GradientDescentOptimizer` is an **Optimizer** object

`tf.train.GradientDescentOptimizer(lr).minimize(cross_entropy)` adds optimization **operation** to computation graph

TensorFlow graph **nodes** have **attached gradient operations**

Gradient with respect to **parameters** computed with **backpropagation**

...automatically

Creating the train_step op

```
prediction = tf.nn.softmax(...)
label = tf.placeholder(tf.float32, [None, 10])

cross_entropy = tf.reduce_mean(-tf.reduce_sum(label * tf.log(prediction),
reduction_indices=[1]))

train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
```

Training the Model

`sess.run(train_step, feeds)`

1. Create Session
2. Build training schedule
3. Run `train_step`

```
sess = tf.Session()
sess.run(tf.global_variables_initializer())

for i in range(1000):
     batch_x, batch_label = data.next_batch()
     sess.run(train_step, feed_dict={x: batch_x,
                                    label: batch_label}
```

Variable sharing: naive way

```
variables_dict = {
    "weights": tf.Variable(tf.random_normal([782, 100]),
                           name="weights")
    "biases": tf.Variable(tf.zeros([100]), name="biases")
    }
```

Not good for encapsulation!

Variable sharing: naive way

```
variables_dict = {
    "conv1_weights": tf.Variable(tf.random_normal([5, 5, 32, 32]),
        name="conv1_weights")
    "conv1_biases": tf.Variable(tf.zeros([32]), name="conv1_biases")
    ... etc. ...
}

def my_image_filter(input_images, variables_dict):
    conv1 = tf.nn.conv2d(input_images, variables_dict["conv1_weights"],
        strides=[1, 1, 1, 1], padding='SAME')
    relu1 = tf.nn.relu(conv1 + variables_dict["conv1_biases"])

    conv2 = tf.nn.conv2d(relu1, variables_dict["conv2_weights"],
        strides=[1, 1, 1, 1], padding='SAME')
    return tf.nn.relu(conv2 + variables_dict["conv2_biases"])

# The 2 calls to my_image_filter() now use the same variables
result1 = my_image_filter(image1, variables_dict)
result2 = my_image_filter(image2, variables_dict)
```

What’s in a Name?

`tf.variable_scope()` provides simple name-spacing to avoid clashes

`tf.get_variable()` creates/accesses variables from within a variable scope

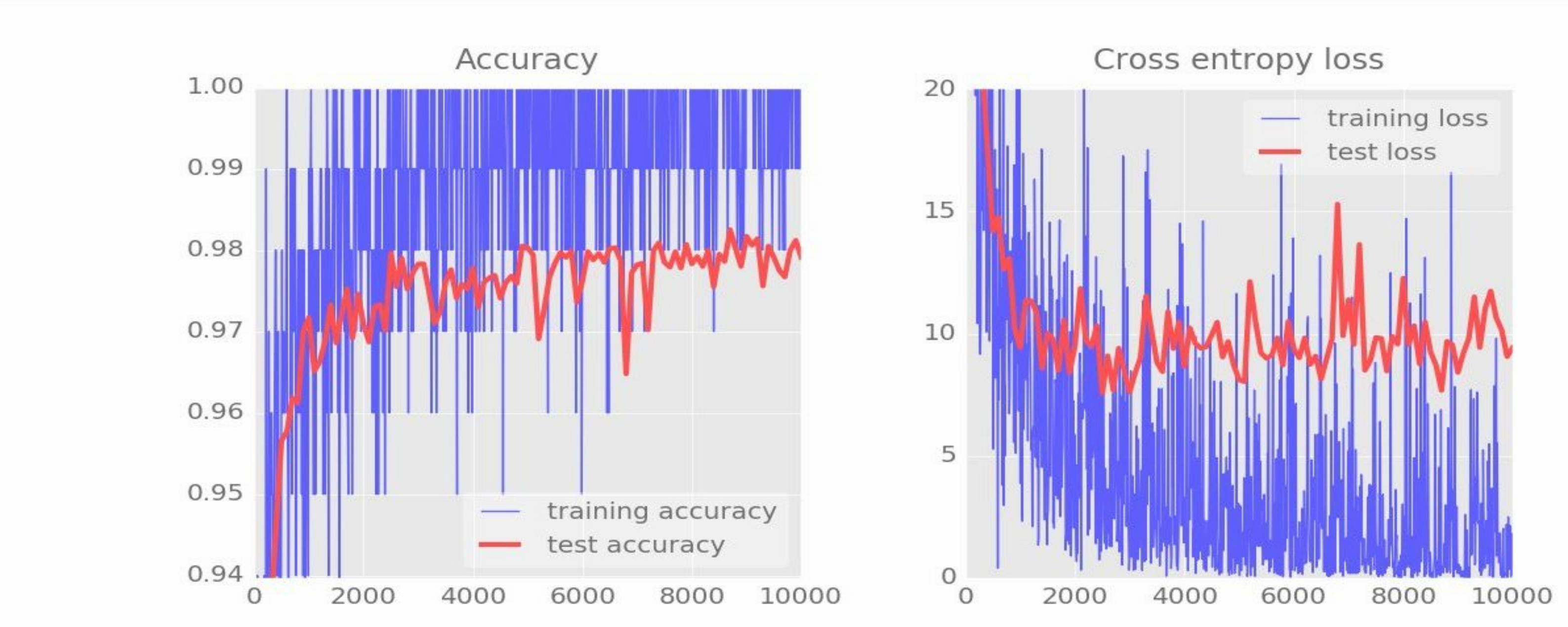
```
with tf.variable_scope("foo"):
    v = tf.get_variable("v", shape=[1])  # v.name == "foo/v:0"

with tf.variable_scope("foo", reuse=True):
    v1 = tf.get_variable("v")      # Shared variable found!

with tf.variable_scope("foo", reuse=False):
    v1 = tf.get_variable("v")      # CRASH foo/v:0 already exists!
```

What's in a Name?

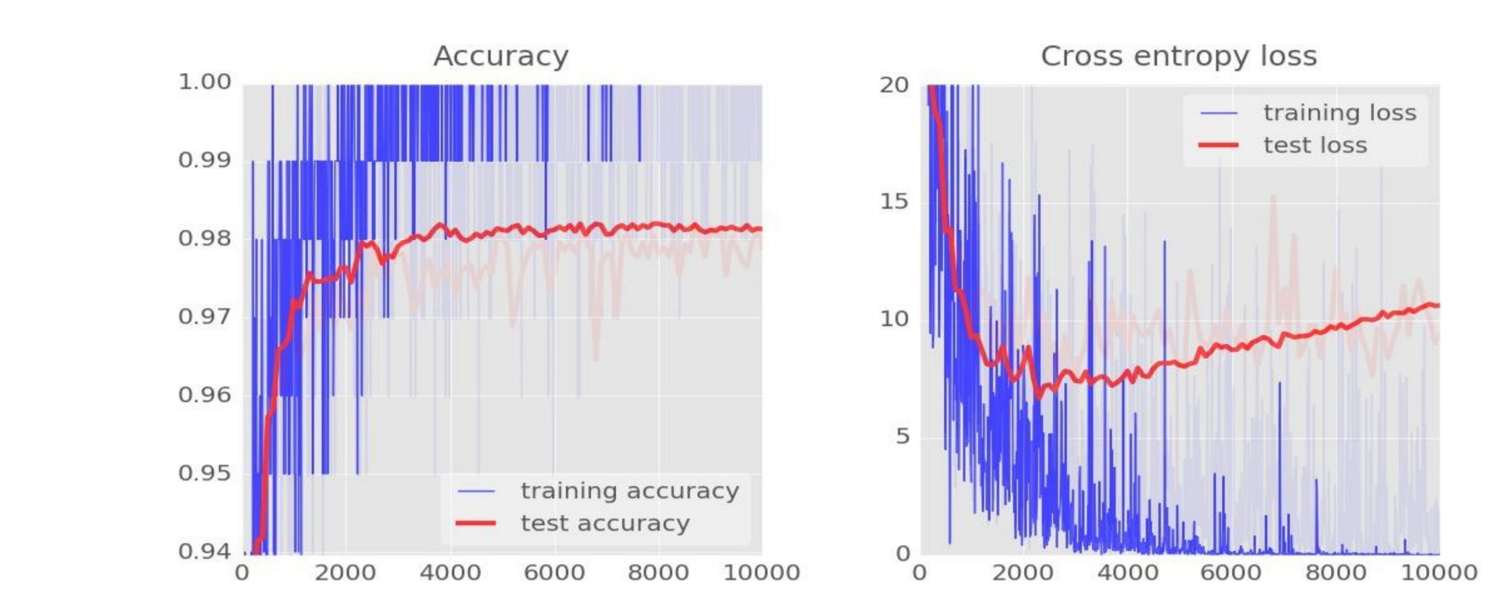
```python
def conv_relu(input, kernel_shape, bias_shape):
    # Create variable named "weights".
    weights = tf.get_variable("weights", kernel_shape,
        initializer=tf.random_normal_initializer())
    # Create variable named "biases".
    biases = tf.get_variable("biases", bias_shape,
        initializer=tf.constant_initializer(0.0))
    conv = tf.nn.conv2d(input, weights,
        strides=[1, 1, 1, 1], padding='SAME')
    return tf.nn.relu(conv + biases)
```

```python
def my_image_filter(input_images):
    with tf.variable_scope("conv1"):
        # Variables created here will be named "conv1/weights", "conv1/biases".
        relu1 = conv_relu(input_images, [5, 5, 32, 32], [32])
    with tf.variable_scope("conv2"):
        # Variables created here will be named "conv2/weights", "conv2/biases".
        return conv_relu(relu1, [5, 5, 32, 32], [32])
```

Noisy Accuracy Curve



Slow down



Learning rate 0.003 at start then dropping exponentially to 0.0001

Regularisation

```
t = tf.Variable(...)

reg_loss = tf.nn.l2_loss(t,name=None)
```

Next class:
More on regularization and best practices in Tensorflow

Thanks!

References:

1. CS231n
2. CS224n
3. Martin Gorner's Slides on Tensorflow
4. tensorflow.org