TensorFlow for People Who Want to Use TensorFlow

Overview

Install TensorFlow

TensorFlow Introduction

Linear Regression from scratch

Linear Regression, the easy way

Using Queues & Checkpoints

MNIST!

DeepDream, maybe?

Install TensorFlow

Follow the instructions at
https://github.com/martinwicke/tensorflow-tutorial

A multidimensional array.

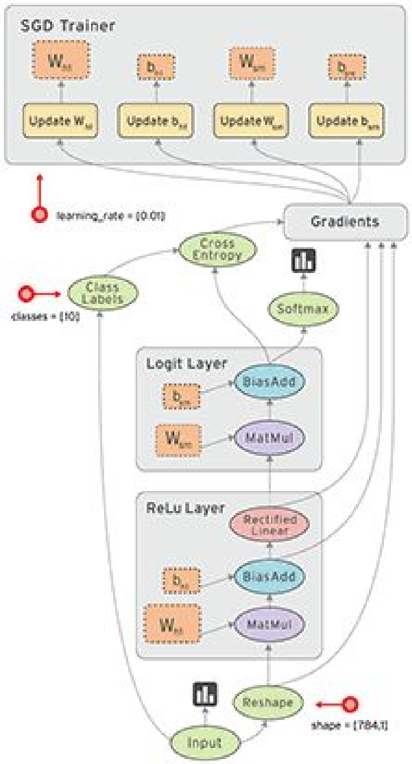
TensorFlow

A graph of operations.

Flow

Computation is defined as a directed acyclic graph (DAG) to optimize an objective function

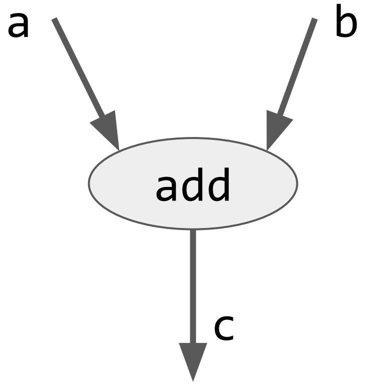
- Graph is defined in high-level language (Python)
- Graph is compiled and optimized
- Graph is executed (in parts or fully) on available low level devices (CPU, GPU)
- Data (tensors) flow through the graph
- TensorFlow can compute gradients automatically



Build a graph; then run it.

TensorFlow separates computation graph construction from execution.

```
...
c = tf.add(a, b)


session = tf.Session()
numpy_c = c.eval(session)
```

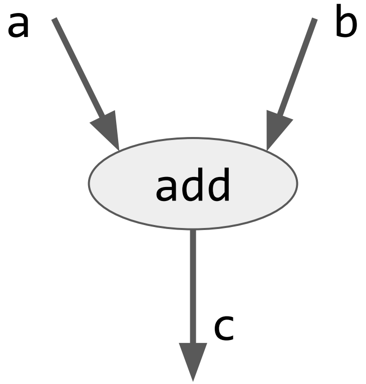


What’s in a Graph?

Edges: Tensors, Nodes: Ops

All nodes are Ops:

- Constants
- Variables
- Computation
- Debug code (Print, Assert)
- Control Flow



Variables

Some ops in a TensorFlow graph are stateful: (mainly) Variables

- Can be assigned to
- Must be initialized
- It is easy to create race conditions
    - Welcome to concurrent programming
    - Races are mostly harmless in stochastic data-parallel algorithms

Shape Inference

The data type of a Tensor is fixed during construction, its shape is not

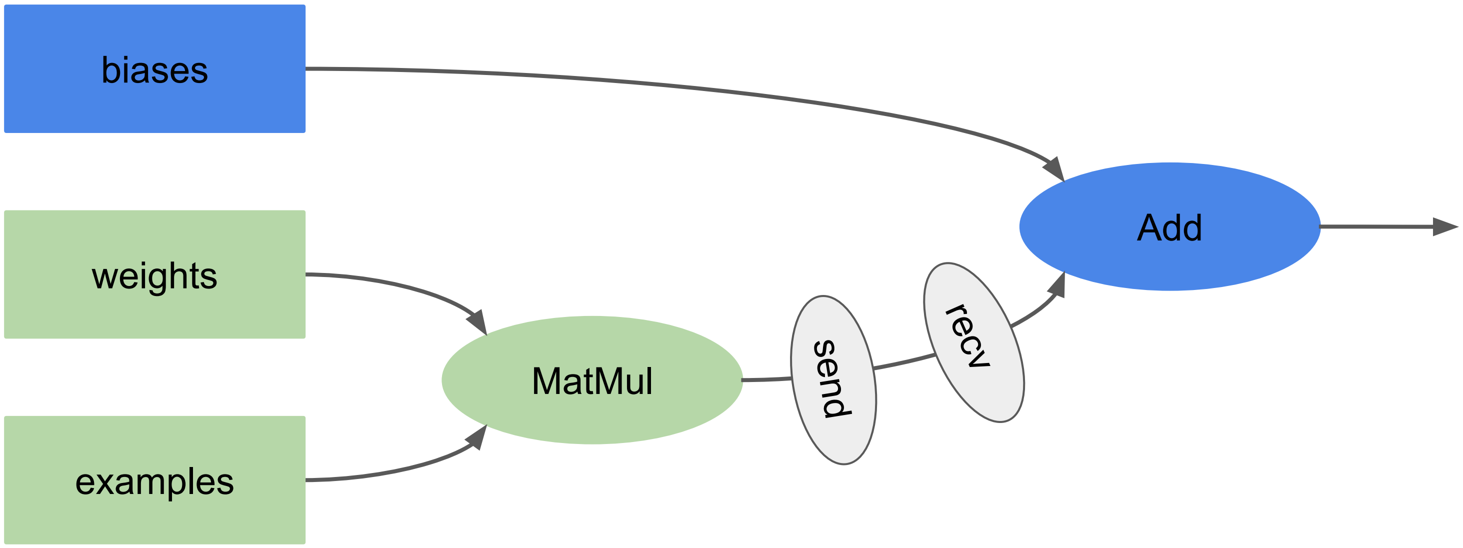
```
a = tf.decode_png(bytes, channels=3,
                  dtype=tf.uint8)

a.get_shape() ⇒ [None, None, 3]
```

Shape inference propagates shapes as much as possible

```
b = tf.tile(a, [2, 2, 3])

b.get_shape() ⇒ [None, None, 9]
```

But, why?

Graphs can be processed, compiled, remotely executed, assigned to devices.



Automatic differentiation

Graph computing gradients can be computed automatically

Every Op has a corresponding gradient Op computing partial derivatives

TensorFlow knows the chain rule

You specify the forward computation, `tf.gradients` adds gradient Ops

Graphs can be explicit-ish

You can have several independent graphs at the same time

```
with tf.Graph().as_default():
  a = tf.constant(1)
  b = tf.constant(2)
  c = tf.add(a, b)

with tf.Graph().as_default():
  a = tf.constant(10)
  b = tf.add(a, c)  # Error!
```

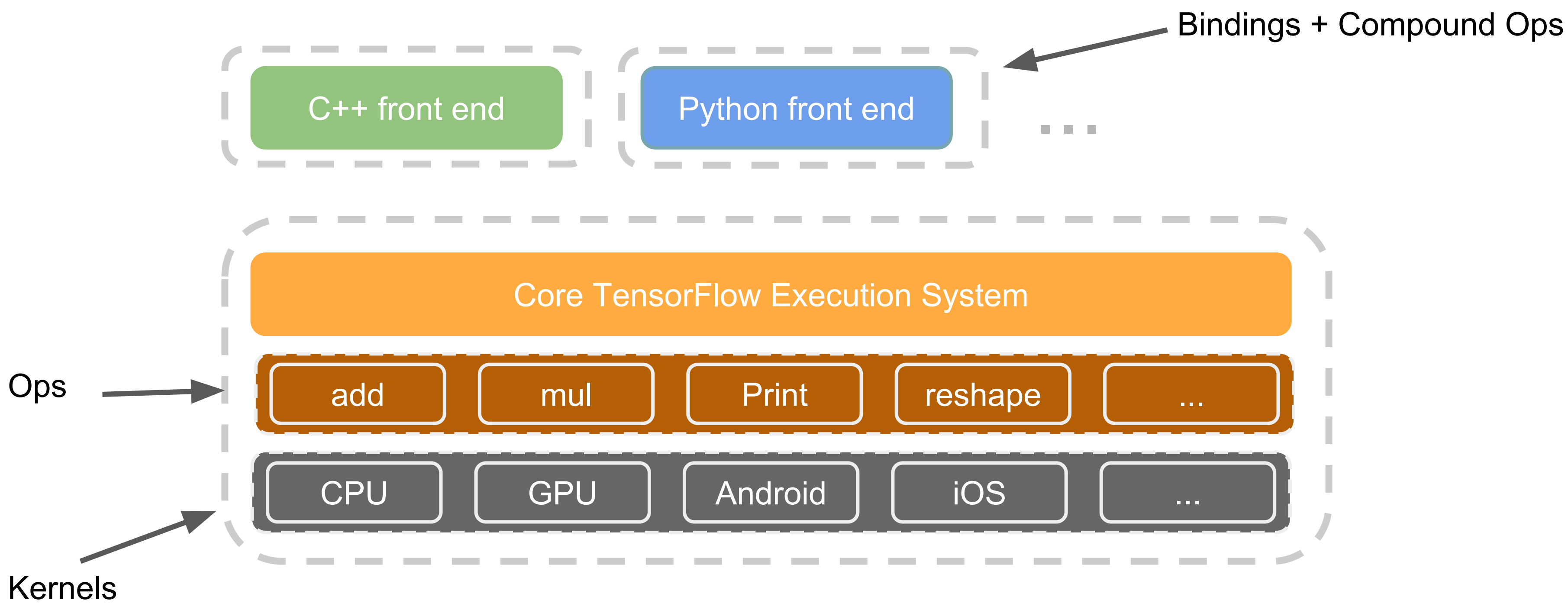
Building graphs looks *mostly* like numpy

With special functions for deep learning

Numpy	TensorFlow

`add`
`mul`
`matmul`
…
`sum`
…
	`add`
	`mul`
	`matmul`
	…
	`reduce_sum`
	…
	
	`sigmoid`
	`relu`
	…

TensorFlow Architecture



Extending TensorFlow

Ops are small, but easy to combine into bigger pieces

- Writing new algorithms requires no knowledge about TensorFlow internals

```
def my_algorithm(input, depth):
  output = input
  for i in xrange(depth):
    output = tf.contrib.layers.relu(output, 200)
  return output
```

Extending TensorFlow

Ops are small, but easy to combine into bigger pieces

- Writing new algorithms requires no knowledge about TensorFlow internals
- Writing new compound Ops requires no knowledge about lower levels

```
def my_op(t, min, max):
  t_min = math_ops.minimum(t, max)
  t_max = math_ops.maximum(t_min, min)
  return t_max
```

TensorFlow is a RISC architecture

Ops are small, but easy to combine into bigger pieces

- Writing new algorithms requires no knowledge about TensorFlow internals
- Writing new compound Ops requires no knowledge about lower levels

This flexibility makes TensorFlow ideal for Research

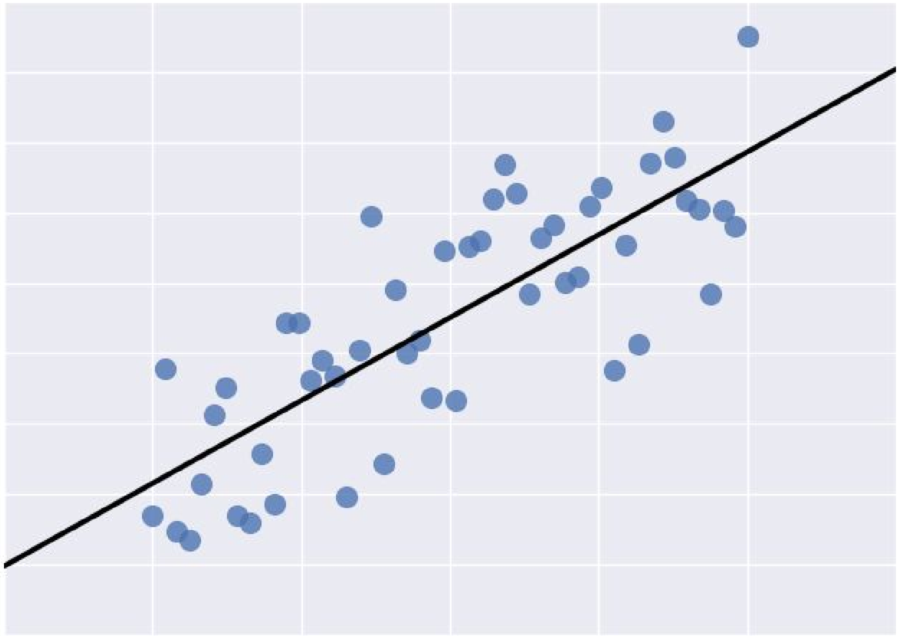
Let’s write some code.

Linear Regression from Scratch

result input

$$y = Wx + b$$

parameter



$y = Wx + b$ in TensorFlow

```
import tensorflow as tf

x = tf.placeholder(shape=[2,1], dtype=tf.float32, name=”x”)

W = tf.get_variable(shape=[1,2], name=”W”)

b = tf.get_variable(shape=[1], name=”b”)

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

Variables Must be Initialized

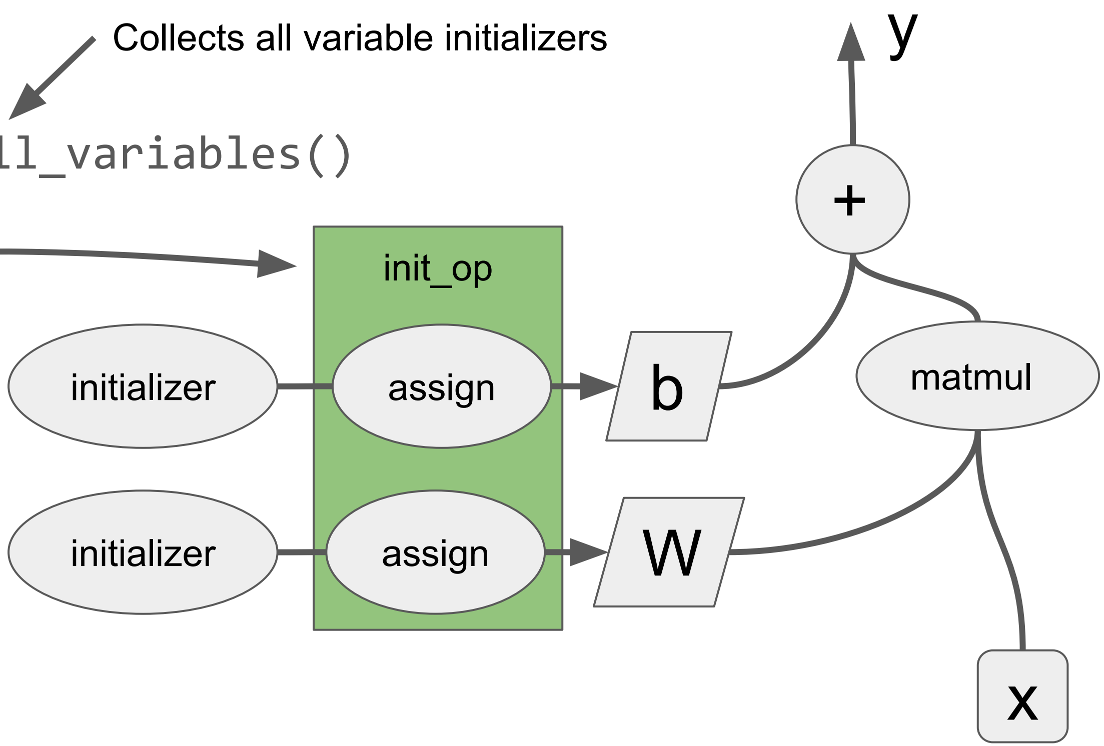
Collects all variable initializers

```
init_op = tf.initialize_all_variables()
```

Makes an execution environment

```
sess = tf.Session()

sess.run(init_op)
```

Actually initialize the variables



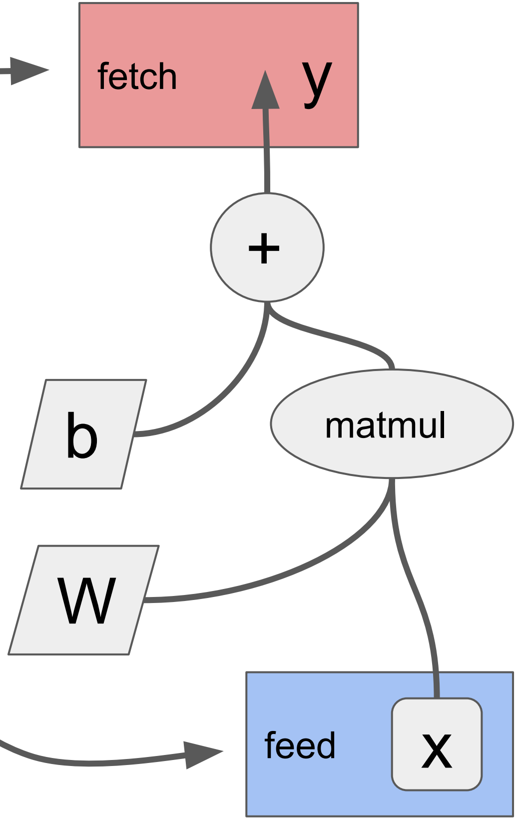
Running the Computation

```
x_in = [[3], [4]]
```

```
sess.run(y, feed_dict={x: x_in})
```

Only what's used to compute a fetch will be evaluated

All Tensors can be fed, but all placeholders must be fed



The full program

```
import tensorflow as tf
x = tf.placeholder(shape=[2,1],
                   dtype=tf.float32,
                   name=”x”)
W = tf.get_variable(shape=[1,2], name=”W”)
b = tf.get_variable(shape=[1], name=”b”)
y = tf.matmul(W, x) + b

with tf.Session() as sess:

  sess.run(tf.initialize_all_variables())

  print sess.run(y, feed_dict={x: x_in})
```

Build the graph

Prepare execution environment

Initialize variables

Run the computation (usually often)

Exercise: Define a Loss

Given x, y_{label}, compute a loss, for instance:

$$L = (y - y_{label})^2$$

Hint, numpy's `sum` is called `reduce_sum`.

Solution

```
y_label = tf.placeholder(shape=[1,1], dtype=float32,
                                name=”y_label”)
diff = y - y_label
L = tf.reduce_sum(diff * diff)
```

Evaluation

```
eval_data = np.loadtxt(open("eval_data.csv","rb"), delimiter=",")

acc = 0.

for x1, x2, y_in in eval_data:

  acc += sess.run(L, feed_dict={x: [[x1],[x2]], y_label: y_in})

print acc/len(eval_data)
```

Training

Feed (x, y_{label}) pairs and adjust W and b to decrease the loss.

$$W \leftarrow W - \eta\ (dL/dW)$$

$$b \leftarrow b - \eta\ (dL/db)$$

`tf.gradients(L, [W, b])` computes gradients of `L`.

`tf.GradientDescentOptimizer` creates Ops that perform the update step.

Training

```
L = ...

train_op = tf.train.GradientDescentOptimizer(learning_rate=0.01)
                   .minimize(L)



data = numpy.loadtxt(open("training_data.csv","rb"), delimiter=",")



for x1, x2, y_in in data:

  sess.run(train_op, feed_dict={x: [[x1],[x2]], y_label: y_in})
```

That seems complicated...

```
import tensorflow as tf

R = tf.contrib.learn.LinearRegressor(feature_columns=[
    tf.contrib.layers.real_valued_column('', dimension=2)])



R.fit(x=data[:,0:2], y=data[:,2:3], batch_size=100, max_steps=100)

R.evaluate(x=eval_data[:,0:2], y=eval_data[:,2:3])

R.predict(x=np.asarray([1.5, 3.4]))
```

Exercise: Use a DNN

Hints: Use either

- Use `DNNRegressor`, or
- Start from the "From Scratch" version and use `tf.contrib.layers.relu`

Solution

```
import tensorflow as tf

D = tf.contrib.learn.DNNRegressor(feature_columns=[
    tf.contrib.layers.real_valued_column('', dimension=2)],
                                  hidden_units=[10,]*9)
```

Reading Data From Files

TensorFlow has text and binary file readers

`WholeFileReader, TestLineReader, FixedLengthRecordReader, ...`

As well as decoders

`decode_png, decode_jpg, decode_gif, decode_csv, ...`

The file readers need a `Queue` as input.

Queues

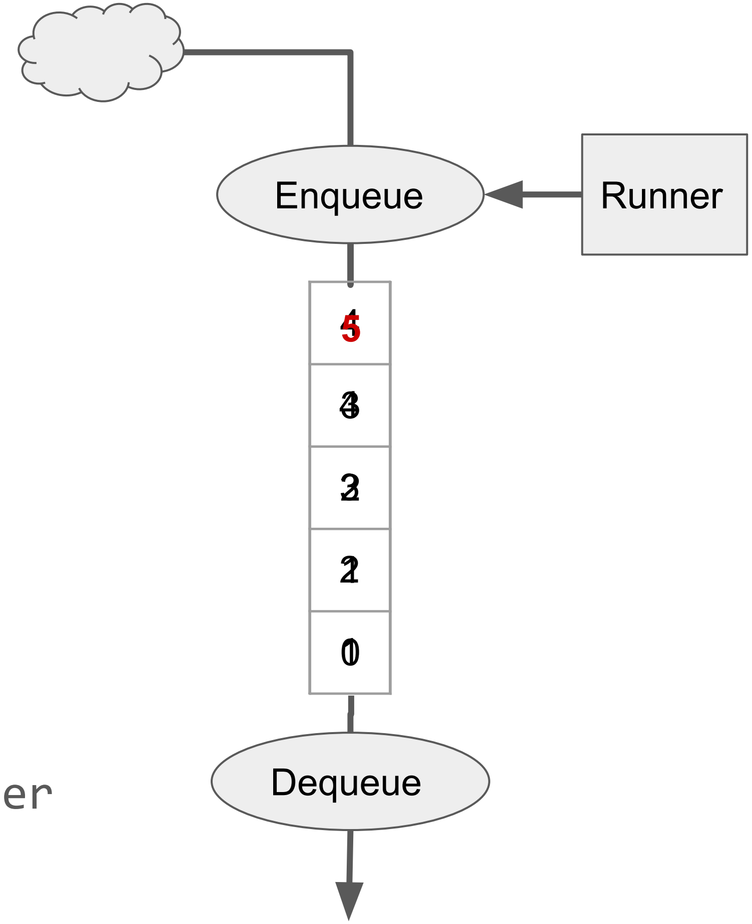
An Object with Enqueue, Dequeue Ops.

Can be asynchronously refilled by `QueueRunners`.

Internal state must be initialized!

State is not cleanly restored from Checkpoints!

Most common: `tf.train.string_input_producer`



Reading our data from file

```
filename_queue = tf.train.string_input_producer(["training_data.csv"], num_epochs=1)

reader = tf.TextLineReader()

key, line = reader.read(filename_queue)

x1, x2, y_in = tf.decode_csv(line, record_defaults=[[0.0], [0.0], [0.0]])

sess.run(tf.group(tf.initialize_all_variables(),

                  tf.initialize_local_variables()))

tf.train.start_queue_runners(sess)
```

Reading our data from file (2)

… define Lq, trainq …

```
try:
    for i in xrange(100000):
        sess.run(trainq)
        if i % 1000 == 0:
            print "i %d" % i
            eval()
except tf.errors.OutOfRangeError:
    print "done, i=%d" % i
```

Checkpoints

Checkpoints save variable content to disk

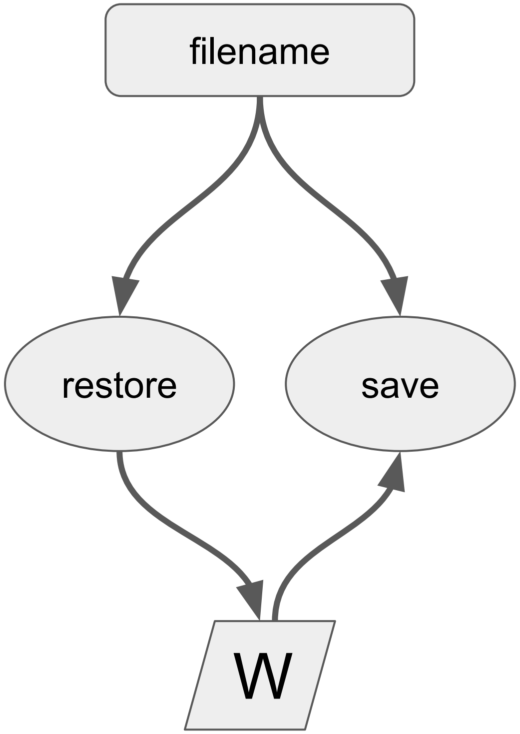
`tf.Saver` adds save/restore Ops to each variable

`Saver.save` feeds a filename and fetches ‘save’

`Saver.restore` feeds a filename and fetches ‘restore’

You can select which variables to restore

`contrib/learn/utils` has helpers for warmstarting



MNIST

the only benchmark that matters

Load the data

```
mnist = learn.datasets.load_dataset('mnist')


data = mnist.train.images

labels = np.asarray(mnist.train.labels, dtype=np.int32)

eval_data = mnist.test.images

eval_labels = np.asarray(mnist.test.labels, dtype=np.int32)
```

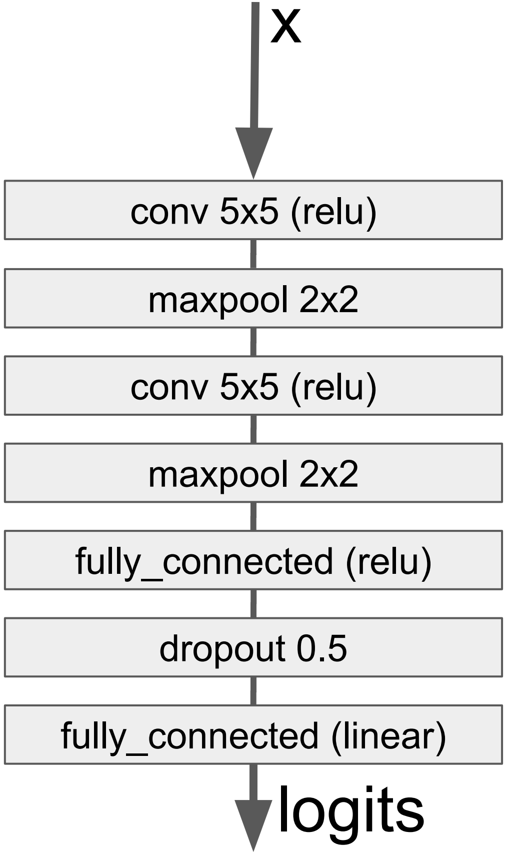
Linear mnist

```
feature_columns = learn.infer_real_valued_columns_from_input(data)

classifier = learn.LinearClassifier(feature_columns=feature_columns,
                                    n_classes=10)

classifier.fit(data, labels, batch_size=100, steps=1000)

classifier.evaluate(eval_data, eval_labels)
```

Let’s make a proper model

Use layers to quickly assemble neural networks

```
x = tf.contrib.layers.conv2d(x, kernel_size=[5,5], ...)

x = tf.contrib.layers.max_pool2d(x, kernel_size=[2,2], ...)

x = tf.contrib.layers.conv2d(x, kernel_size=[5,5], ...)

x = tf.contrib.layers.max_pool2d(x, kernel_size=[2,2], ...)

x = tf.contrib.layers.relu(x)

x = tf.contrib.layers.dropout(x, 0.5)

logits = tf.config.layers.linear(x)
```



Use `Classifier` for training

```
classifier = learn.Classifier(model_fn=model_fn, n_classes=10)
```

`model_fn` takes `(inputs, labels, mode)`,
returns `(logits, loss, train_op)`

`mode == TRAIN`: don't need `logits`
`mode == EVAL`: don't need `train_op`
`mode == INFER`: only need `logits` (`labels` can be None)

How to make `loss`, `train_op`

```
def model_fn(inputs, labels, mode)
  logits = <call our logits function>
  onehot = tf.one_hot(labels, 10)
  loss, train_op = None, None

  if mode != tf.contrib.learn.ModeKeys.INFER:
    loss = tf.contrib.losses.softmax_cross_entropy(logits, onehot)

  if mode == tf.contrib.learn.ModeKeys.TRAIN:
    train_op = tf.contrib.layers.optimize_loss(loss,
        tf.contrib.framework.get_global_step(),
        learning_rate=0.001, 'SGD')

  return logits, loss, train_op
```

What I did not tell you

You can control where things are executed:

```
with tf.device('/gpu:2'):
```

You can make a cluster and then distribute computation:

```
with tf.device("/job:worker/task:7"):
```

Using data-parallelism, this is surprisingly automatic

Want more?

Do the DeepDream notebook

Do the tensorflow.org tutorials

Write your next model in TensorFlow