Introduction to TensorFlow

DL 2.0. Workshop

Gaurav Manek

TensorFlow?

It is

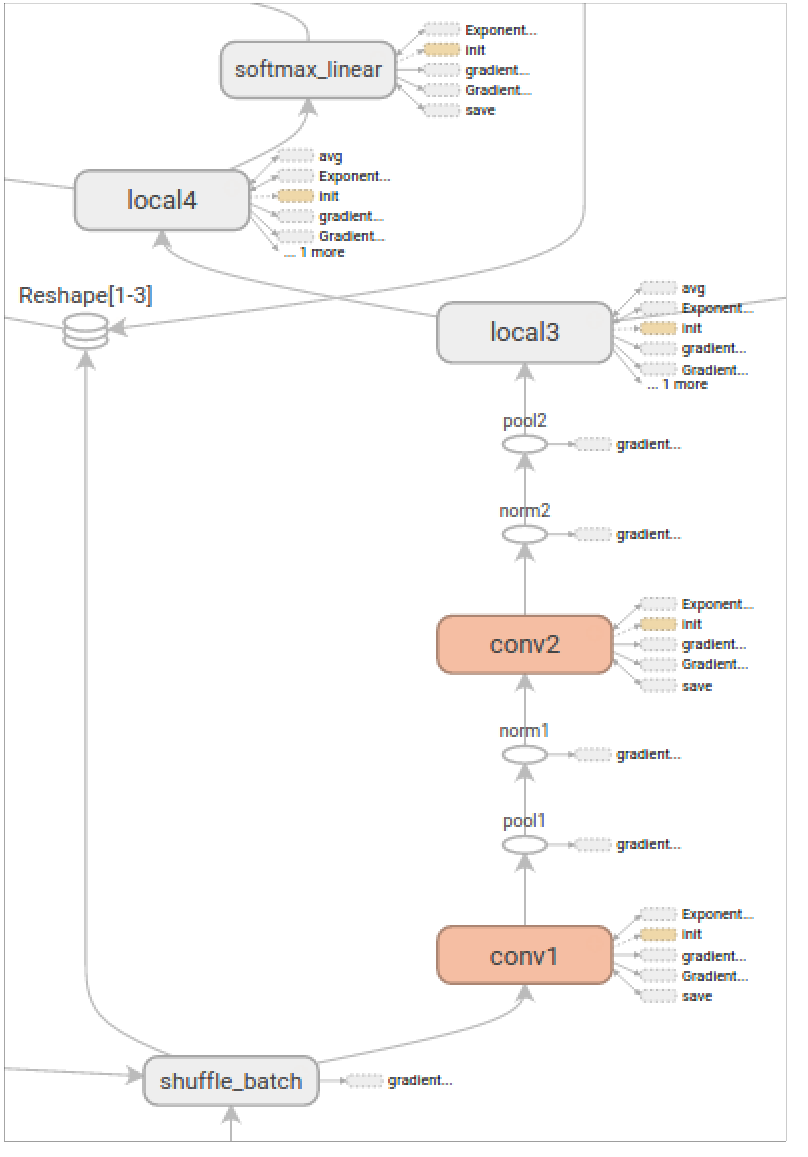
- graph-driven computation library
- support for arrays of arbitrary dimensions
- filled with common (and uncommon!) neural network primitives
  - convolutions
  - optimizers (automatic differentiation!)

It is NOT

- a programming language
- *just* a neural network library
  - it can do a lot more!
- Caffe/MatConvNet
- a substitute for NumPy/SciPy

“graph-driven computation”?

1. Design your model as a graph, including the training and evaluation.
2. Write it in Python
3. TensorFlow will build the model on the CPU/GPU and run it there.



Some Caveats

- Once the model is initialized (i.e. memory is allocated), the graph is immutable.
- Moving data between native Python/C++ and TensorFlow is inefficient.
  - Perform all the computation you can using TensorFlow primitives, including loading data from disk.
- Adding new TensorFlow primitives is difficult.

Tensor and Variable

Tensor

- The output of any computation.
- A matrix of arbitrary size.
- Can be converted to Numpy array.

Variable

- Stored matrix of arbitrary shape.
- Can be *trainable* – Optimizers are allowed to change.

Op and Placeholder

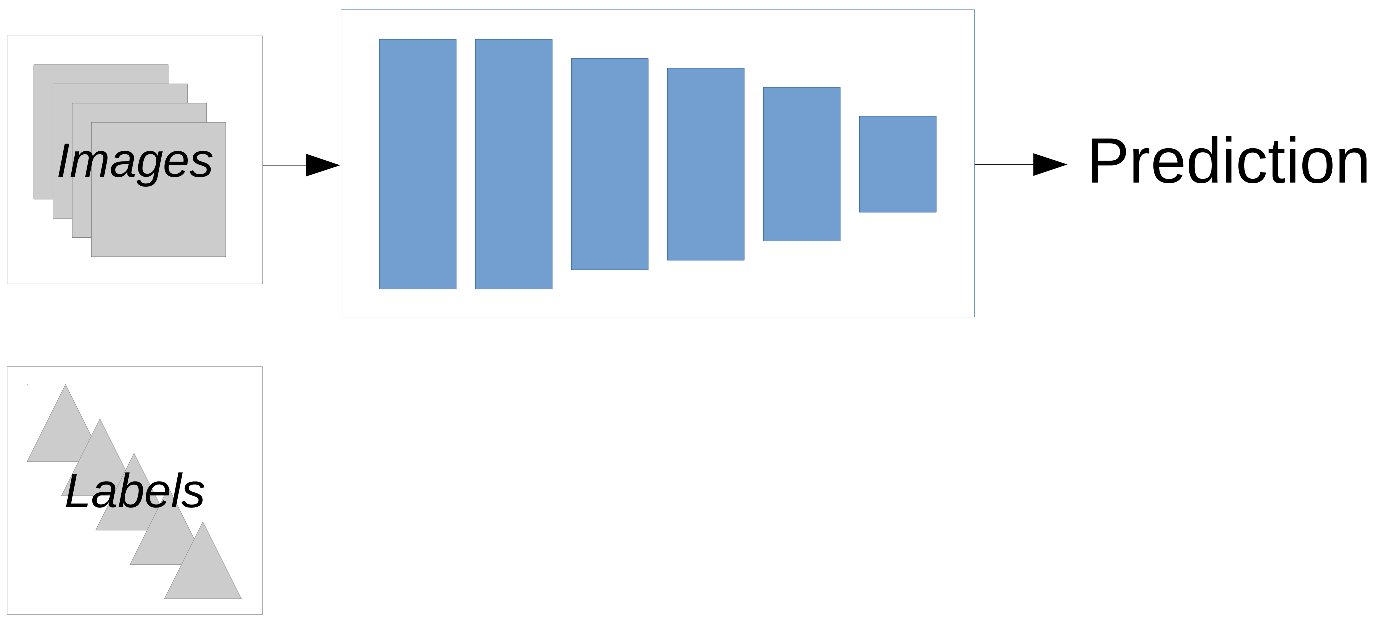
Op

- Any computation – takes variables and ops as input, and (at runtime) produces a *Tensor* as output.
- Any op can be used as a sink/output node. All dependencies are automatically computed.
- Ops and variables can be grouped into a single Op.

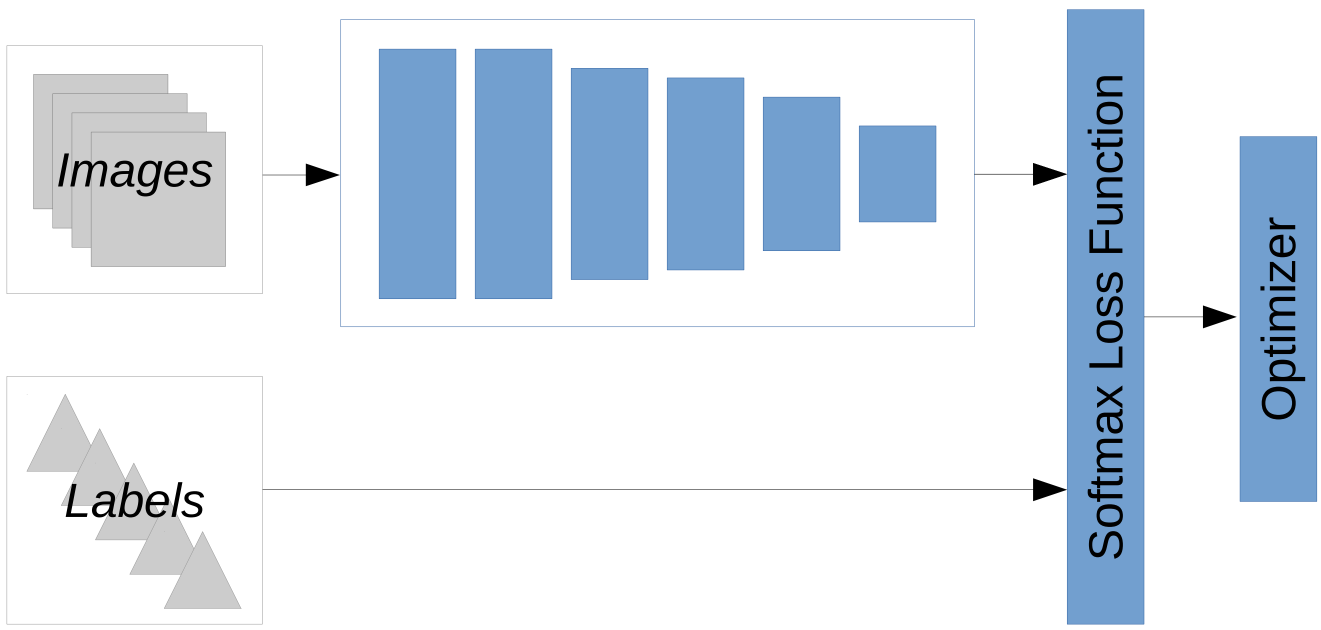
Placeholder

- Reserved space for arbitrary input.
- An actual value must be provided during execution.
- Unfilled placeholders cause exceptions.
- Typically a numpy array is expected.

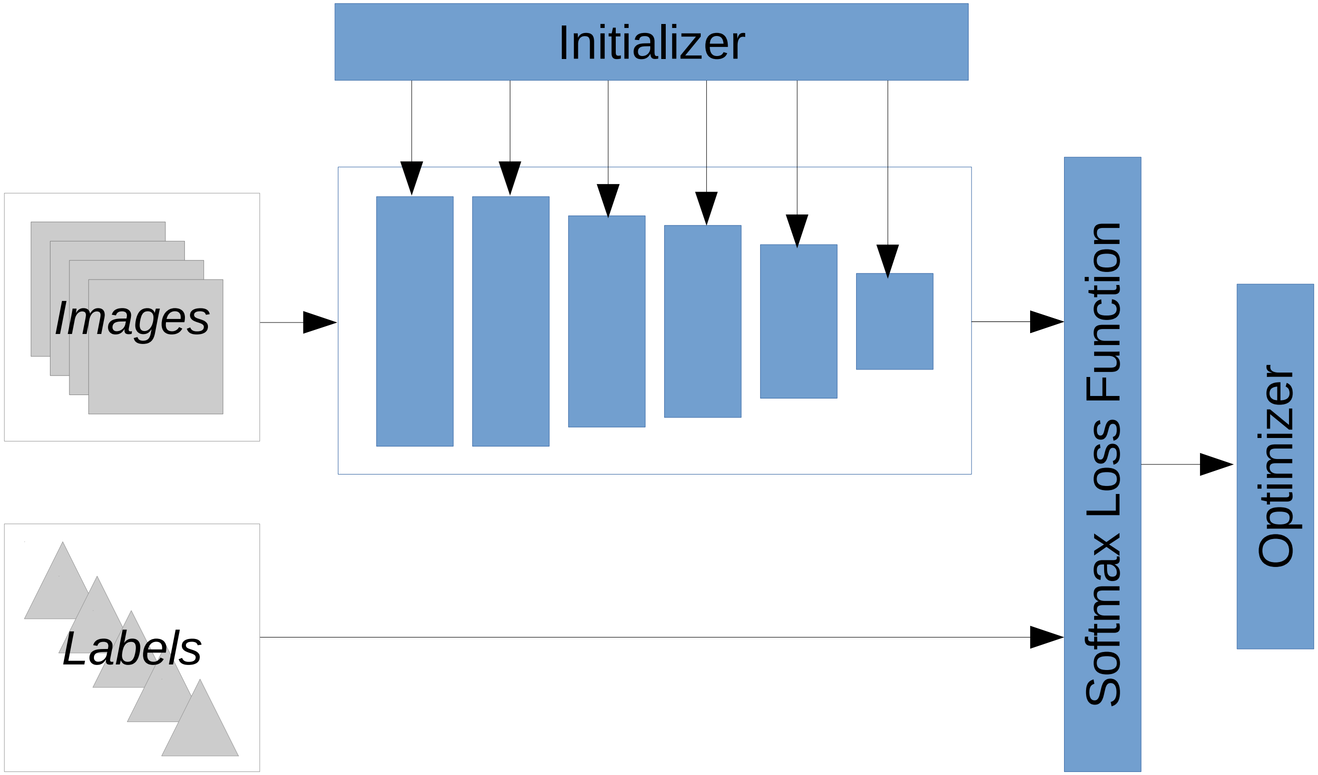
Let’s write a computational graph! (1/5)



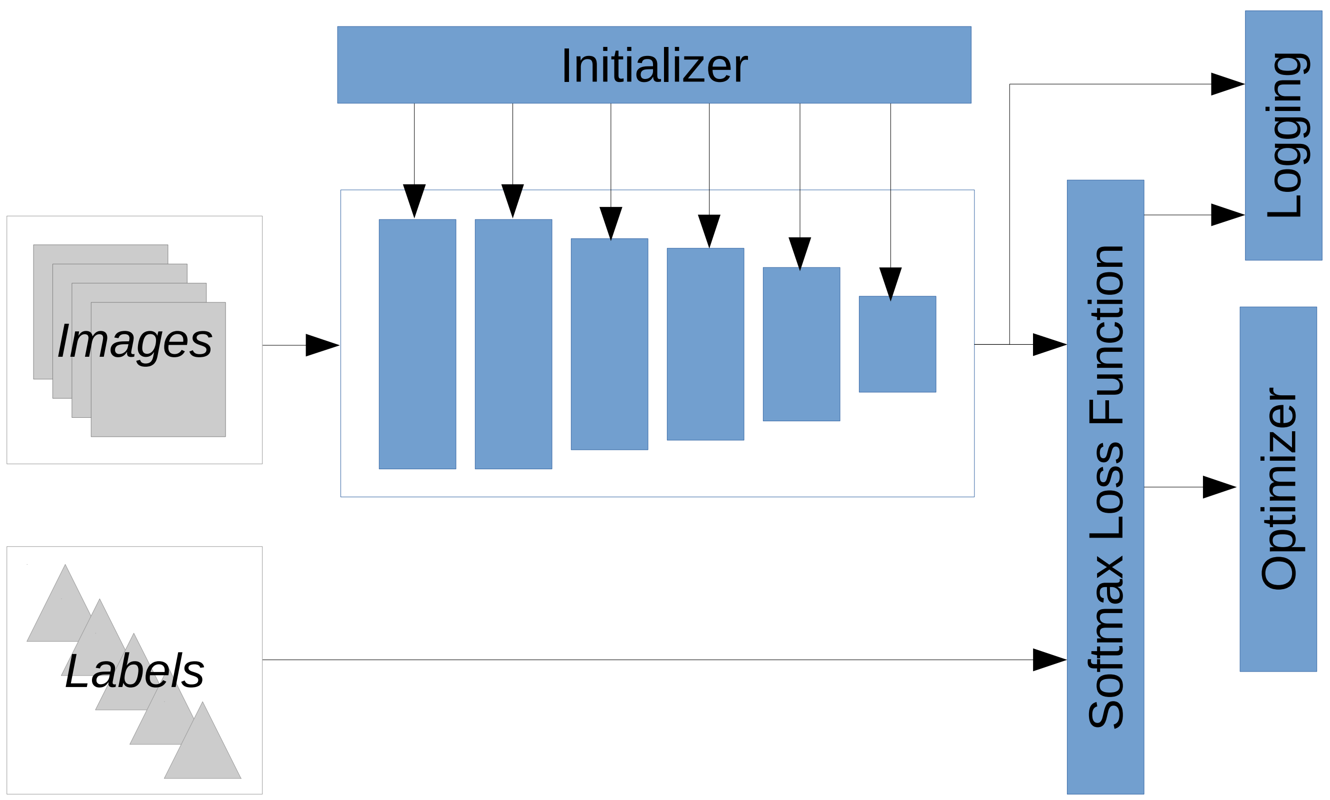
Let's write a computational graph! (2/5)



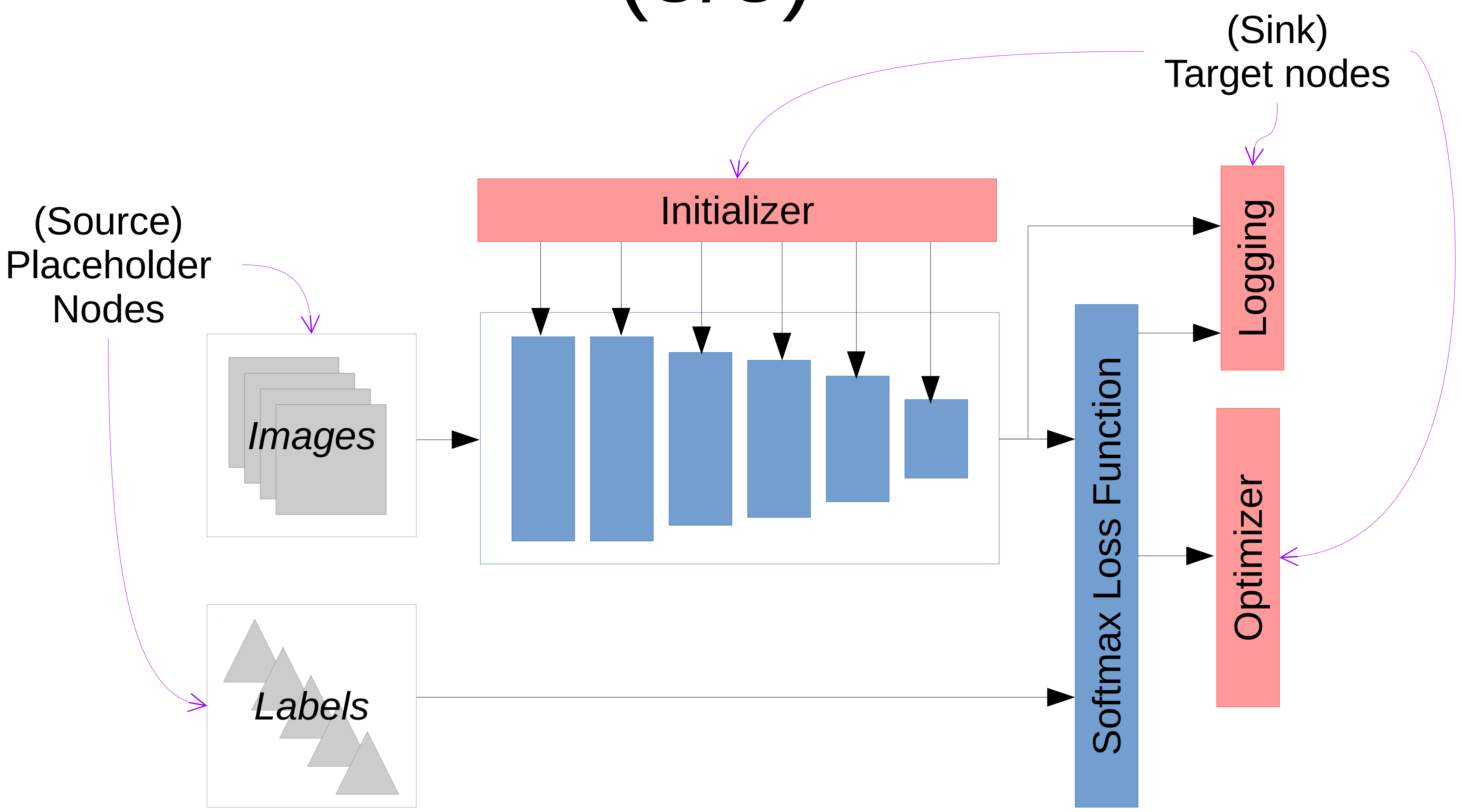
Let's write a computational graph! (3/5)



Let’s write a computational graph! (4/5)



Let’s write a computational graph! (5/5)



cnn/train.py

- Download the example with:

  `git clone https://github.com/gauravmm/dl2w.git`

- Go to the directory and run the example with:

  `python3 workshop.py cnn train`

- You can download pretrained weights using:

  `python3 workshop.py --pretrained cnn`

- The first time you run any example, it will download the dataset.
- You need a wired connection or VPN to access the datasets.

Data Precision

- Supports:
  - `tf.int(8|16|32|64)`
  - `tf.float(16|32|64)`
  - ...
- `tf.float16` offers 10-bit precision, and is a good compromise.
- Caveat:
  - Half-precision on GPU requires hardware support.
    - TITAN X / Tesla cards offer this.

Data Precision + Automatic Differentiation

- Almost all operations support automatic differentiation.
  - Optimizers use this automatically.
- Caveats: Precision Issues!
  - Vanishing Gradients – Avoid `tf.softmax(tf.softmax(x))`
    - The cross-entropy loss functions have this by default!
  - Order of Operations
    - `tf.reduce_sum(tf.log(x))` is better than `tf.log(tf.reduce_prod(x))`

Logging Training Progress

- Insert summary ops in the graph.
  - `tf.summary.*`
- Run the summary op
  - You can run it with some computation (e.g. training) or by itself.
  - Merge all summaries using `tf.summary.merge(_all)?`
  - Output of this is a `tf.Summaries` object.
- Save the summary object
  - Use a `tf.summary.FileWriter` object.

Supervisor

- Automates loading, initialization, summary writing.
- Refer to cnn.py:46-51 for example.
- If any variables are saved, they are transparently loaded when the managed session is created.
  - `with sv.managed_session() as sess:`

What Ops can I Use?

- `tf.nn`
  - Ops that perform computation.
  - An interface to the underlying implementation.
- `tf.layers`
  - Neural network layers!
- `tf.contrib`
  - A huge variety of ops that handle distributions, audio, kernel methods, linear algebra, linear optimization, sparse matrices, sequence-to-sequence, etc.

Some Op Caveats

- Batch Normalization
  - Additional UPDATE_OPS are created, and must be run with the optimizer.
- `<tensor>.get_shape()` vs `<tensor>.shape()`
  - `<tensor>.get_shape()`
    - Shape at construction time
    - Unknown dimensions are ?
  - `<tensor>.shape()`
    - Is an op.
    - Shape at runtime, when all dimensions are known.

`tf.layers`

- Examine `cnn/vgg.py`

Scopes

- `with tf.variable_scope('conv1')`:
  - Used to group variables and ops into logical groups.
- `with tf.name_scope('opname'):`
  - Used to define a new op.

Variable Reuse

- You can share structure without sharing weights:

```
with tf.variable_scope(‘scope_1’) as scope:
        y1 = make_model(x1)
with tf.variable_scope(‘scope_2’) as scope:
        y2 = make_model(x2)
```

- You can share model structure *and* weights:

```
with tf.variable_scope(‘scope_1’) as scope:
        y1 = make_model(x1)
        scope.reuse_variables()
        y2 = make_model(x2)
```

In Summary

- TensorFlow’s computation model
- Comptuational Graph cannot be changed after it is initialized
- Automatic dependency calculations
  - You tell it what sinks you want,
  - If you’re missing any sources, it barfs
- Training supervisors automate a lot of the overhead.
- Some Ops are special – read the documentation!