

Tensor Flow

Tensors: n-dimensional arrays

Vector: 1-D tensor

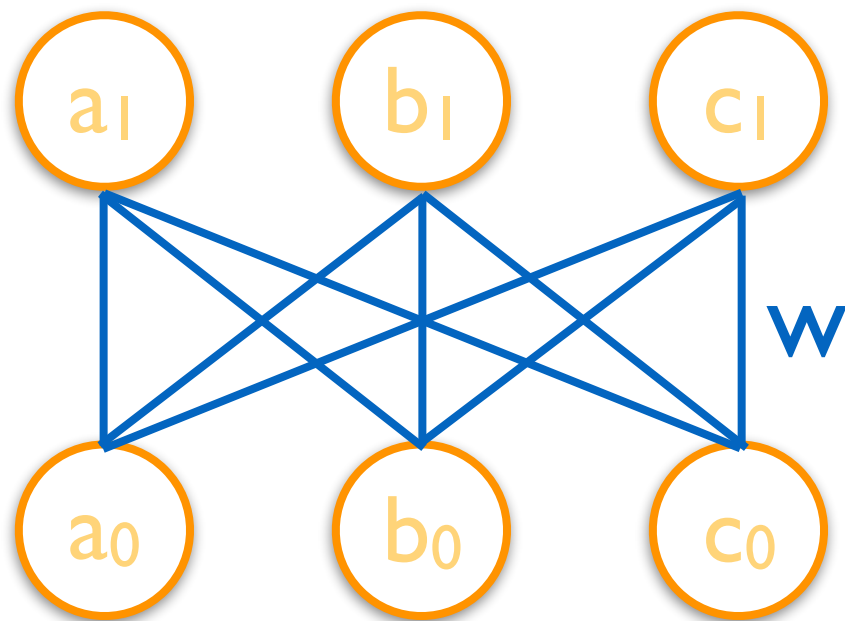
Matrix: 2-D tensor

Deep learning process are flows of tensors

A sequence of tensor operations

Can represent also many machine learning algorithms

A simple ReLU network



$$a_1 = a_0 w_{a,a} + b_0 w_{b,a} + c_0 w_{c,a}$$

$$b_1 = a_0 w_{a,b} + b_0 w_{b,b} + c_0 w_{c,b}$$

$$c_1 = a_0 w_{a,c} + b_0 w_{b,c} + c_0 w_{c,c}$$

Apply $\text{relu}(\dots)$ on a_1, b_1, c_1

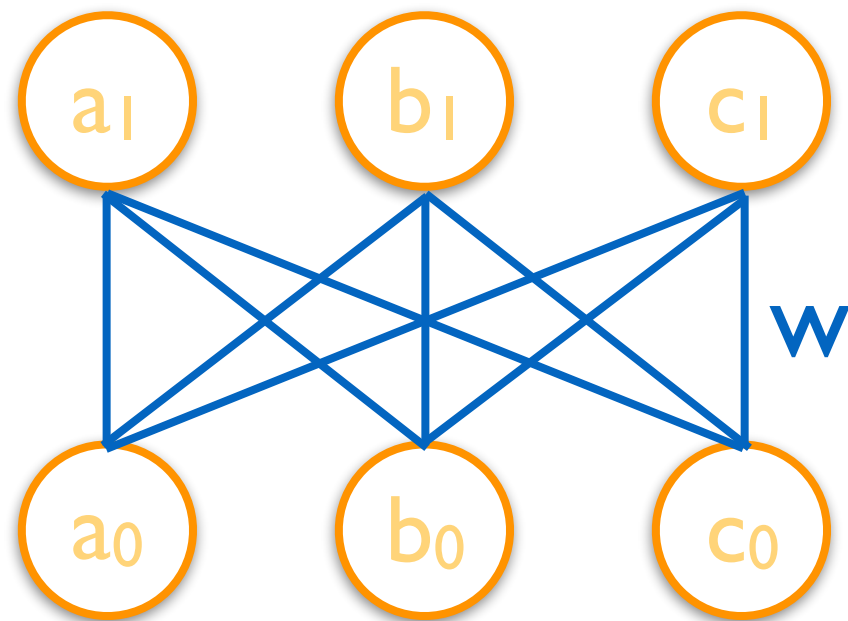
Slower approach

Per-neuron operation

More efficient approach

Matrix operation

As matrix operations

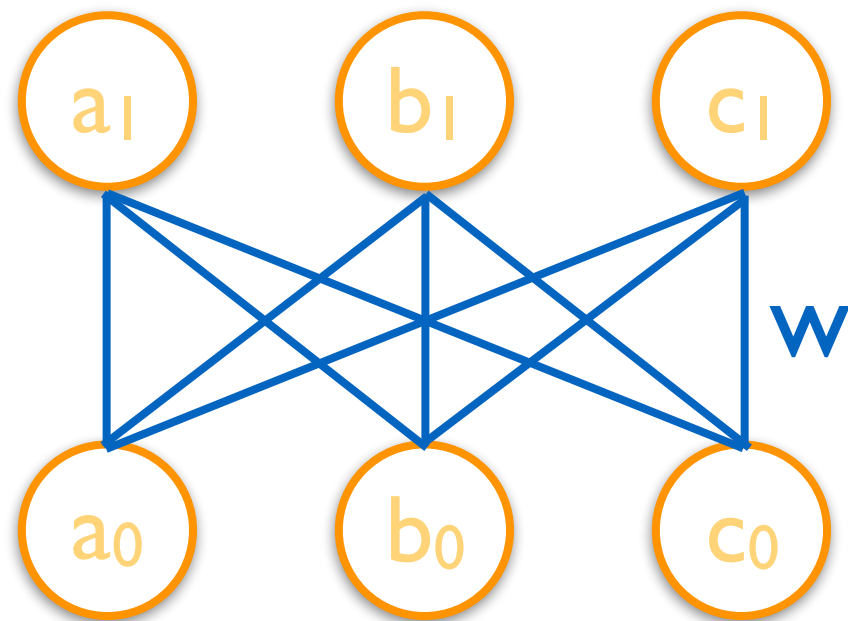


$$\begin{bmatrix} a_0 & b_0 & c_0 \end{bmatrix} \cdot \begin{bmatrix} w_{a,a} & w_{a,b} & w_{a,c} \\ w_{b,a} & w_{b,b} & w_{b,c} \\ w_{c,a} & w_{c,b} & w_{c,c} \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & c_1 \end{bmatrix}$$

$$\begin{aligned} a_1 &= \text{relu}(a_1) \\ b_1 &= \text{relu}(b_1) \\ c_1 &= \text{relu}(c_1) \end{aligned}$$

With TensorFlow

```
import tensorflow as tf
```



$$\begin{matrix} & \mathbf{x} & & \mathbf{w} & \\ \begin{bmatrix} a_0 & b_0 & c_0 \end{bmatrix} & \cdot & \begin{bmatrix} w_{a,a} & w_{a,b} & w_{a,c} \\ w_{b,a} & w_{b,b} & w_{b,c} \\ w_{c,a} & w_{c,b} & w_{c,c} \end{bmatrix} & = & \begin{bmatrix} a_1 & b_1 & c_1 \end{bmatrix} \end{matrix}$$

```
y = tf.matmul(x, w)
```

$$\begin{bmatrix} a_1 \end{bmatrix} = \text{relu}(\begin{bmatrix} a_1 \end{bmatrix})$$

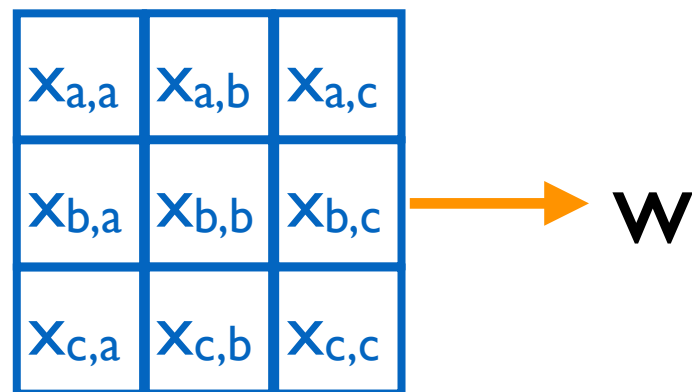
$$\begin{bmatrix} b_1 \end{bmatrix} = \text{relu}(\begin{bmatrix} b_1 \end{bmatrix})$$

$$\begin{bmatrix} c_1 \end{bmatrix} = \text{relu}(\begin{bmatrix} c_1 \end{bmatrix})$$

```
out = tf.nn.relu(y)
```

Define Tensors

$x_{a,a}$	$x_{a,b}$	$x_{a,c}$
$x_{b,a}$	$x_{b,b}$	$x_{b,c}$
$x_{c,a}$	$x_{c,b}$	$x_{c,c}$



Variable(<initial-value>,
name=<optional-name>)

```
import tensorflow as tf
w = tf.Variable(tf.random_normal([3, 3]), name='w')
y = tf.matmul(x, w)
relu_out = tf.nn.relu(y)
```

Variable stores the state of current execution

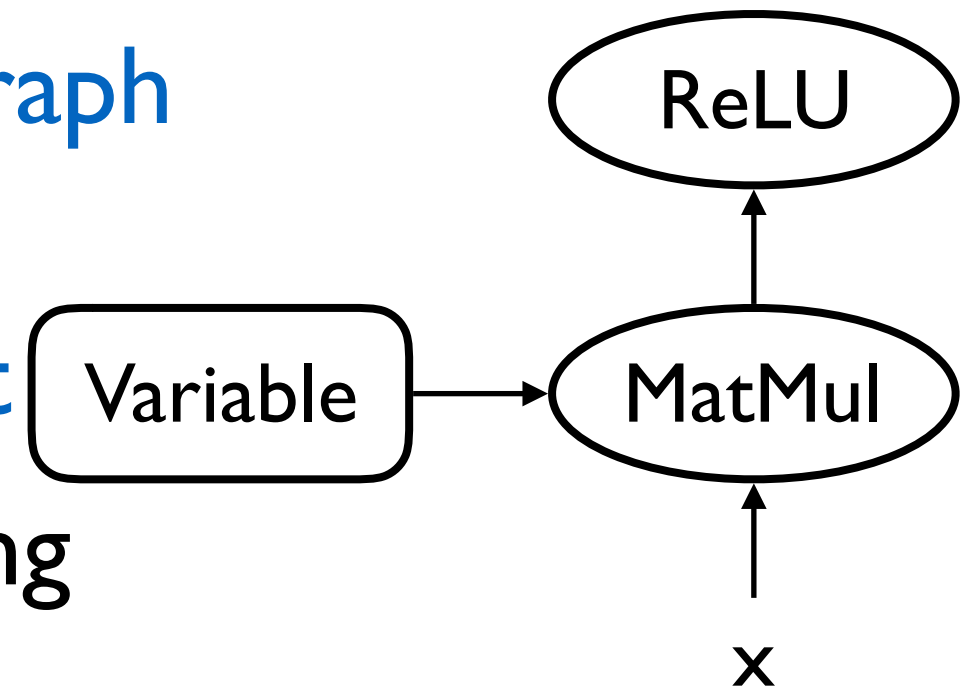
Others are operations

TensorFlow

Code so far defines a data flow **graph**

Each **variable** corresponds to a node in the graph, not the **result**

Can be confusing at the beginning



```
import tensorflow as tf
w = tf.Variable(tf.random_normal([3, 3]), name='w')
y = tf.matmul(x, w)
relu_out = tf.nn.relu(y)
```

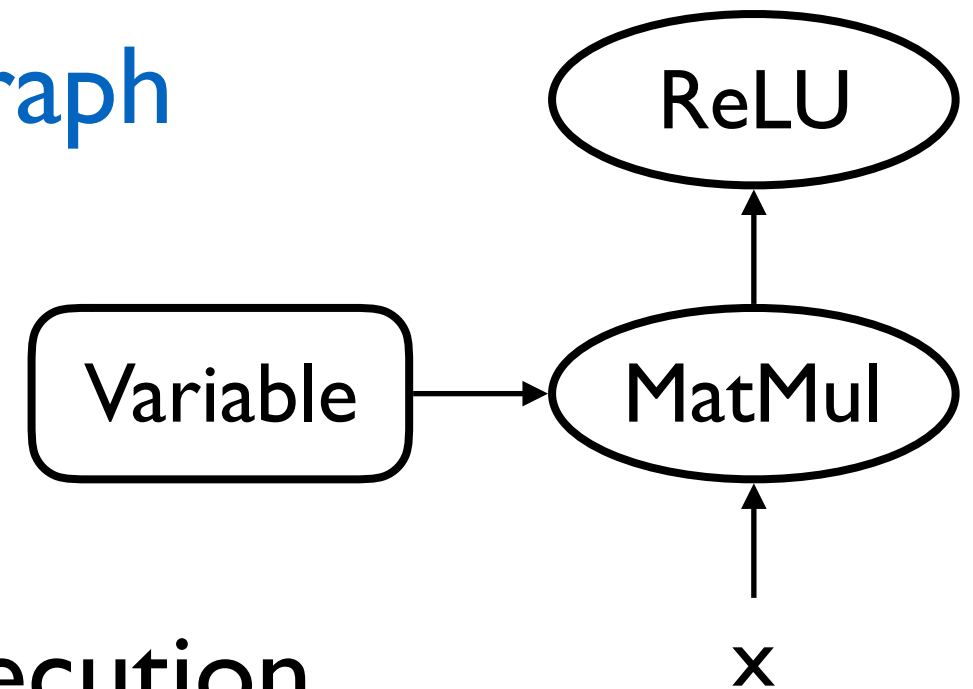
TensorFlow

Code so far defines a data flow **graph**

Needs to specify how we
want to execute the graph

Session

Manage resource for graph execution



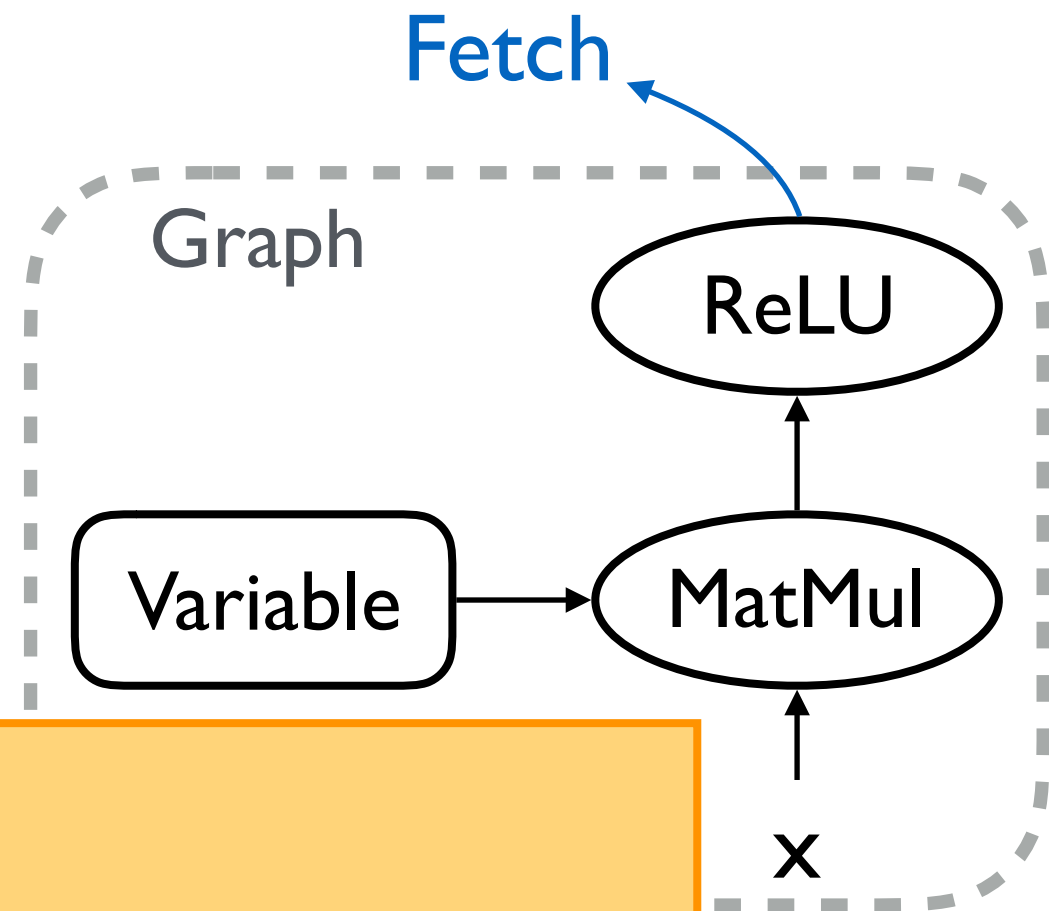
```
import tensorflow as tf
sess = tf.Session()
w = tf.Variable(tf.random_normal([3, 3]), name='w')
y = tf.matmul(x, w)
relu_out = tf.nn.relu(y)
result = sess.run(relu_out)
```

Fetch

Retrieve content from a node

We have assembled the pipes

Fetch the liquid

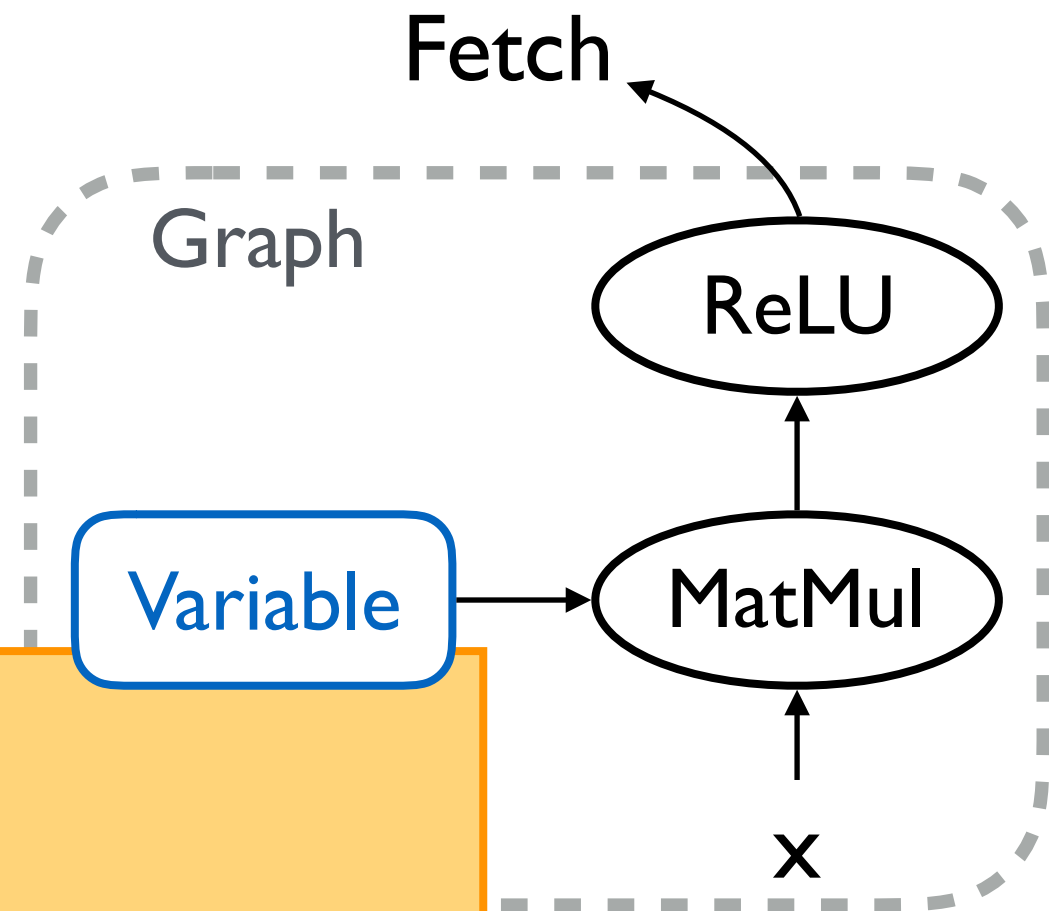


```
import tensorflow as tf
sess = tf.Session()
w = tf.Variable(tf.random_normal([3, 3]), name='w')
y = tf.matmul(x, w)
relu_out = tf.nn.relu(y)
print sess.run(relu_out)
```


Initialize Variable

Variable is an empty node
Fill in the content of a
Variable node

```
import tensorflow as tf
sess = tf.Session()
w = tf.Variable(tf.random_normal([3, 3]), name='w')
y = tf.matmul(x, w)
relu_out = tf.nn.relu(y)
sess.run(tf.initialize_all_variables())
print sess.run(relu_out)
```



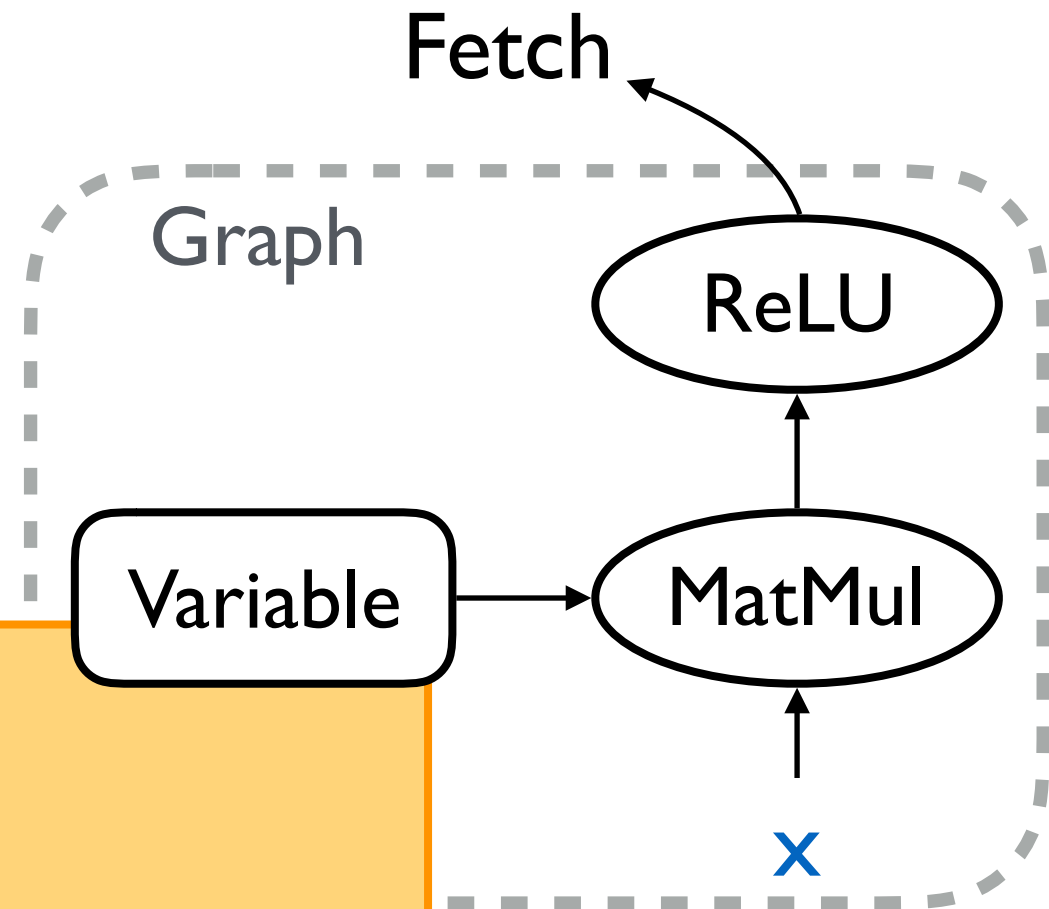
Placeholder

How about **x**?

```
placeholder(<data type>,  
            shape=<optional-shape>,  
            name=<optional-name>)
```

Its content will be fed

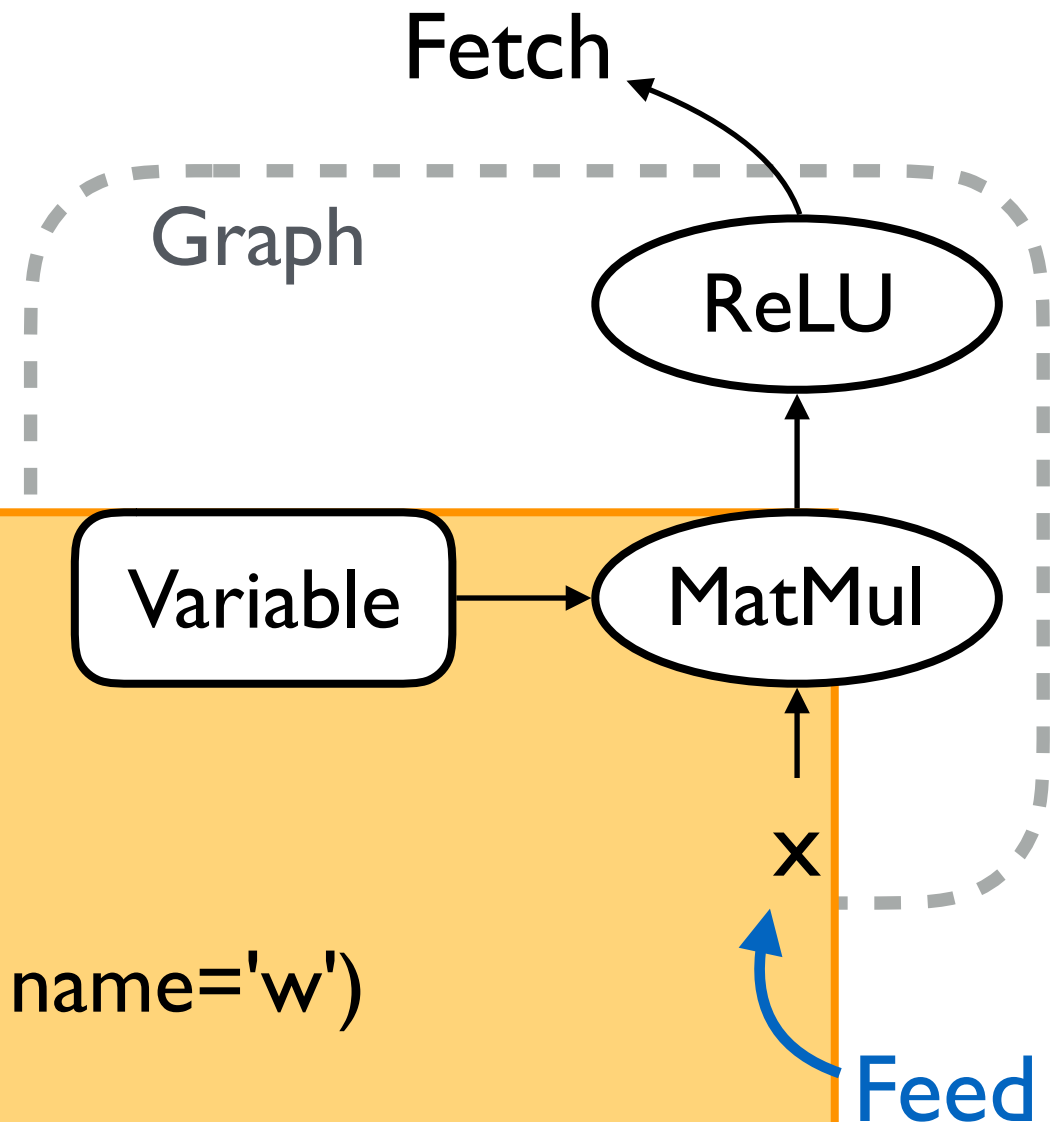
```
import tensorflow as tf  
sess = tf.Session()  
x = tf.placeholder("float", [1, 3])  
w = tf.Variable(tf.random_normal([3, 3]), name='w')  
y = tf.matmul(x, w)  
relu_out = tf.nn.relu(y)  
sess.run(tf.initialize_all_variables())  
print sess.run(relu_out)
```



Feed

Pump liquid into the pipe

```
import numpy as np
import tensorflow as tf
sess = tf.Session()
x = tf.placeholder("float", [1, 3])
w = tf.Variable(tf.random_normal([3, 3]), name='w')
y = tf.matmul(x, w)
relu_out = tf.nn.relu(y)
sess.run(tf.initialize_all_variables())
print sess.run(relu_out, feed_dict={x:np.array([[1.0, 2.0, 3.0]])})
```



Session management

Needs to release resource after use

```
sess.close()
```

Common usage

```
with tf.Session() as sess:  
    ...
```

Interactive

```
sess = InteractiveSession()
```

Prediction

Softmax

Make predictions for n targets that sum to 1

```
import numpy as np
import tensorflow as tf

with tf.Session() as sess:
    x = tf.placeholder("float", [1, 3])
    w = tf.Variable(tf.random_normal([3, 3]), name='w')
    relu_out = tf.nn.relu(tf.matmul(x, w))
    softmax = tf.nn.softmax(relu_out)
    sess.run(tf.initialize_all_variables())
    print sess.run(softmax, feed_dict={x:np.array([[1.0, 2.0, 3.0]])})
```

Prediction Difference

```
import numpy as np
import tensorflow as tf

with tf.Session() as sess:
    x = tf.placeholder("float", [1, 3])
    w = tf.Variable(tf.random_normal([3, 3]), name='w')
    relu_out = tf.nn.relu(tf.matmul(x, w))
    softmax = tf.nn.softmax(relu_out)
    sess.run(tf.initialize_all_variables())
    answer = np.array([[0.0, 1.0, 0.0]])
    print answer - sess.run(softmax, feed_dict={x:np.array([[1.0, 2.0, 3.0]])})
```

Learn parameters: Loss

Define **loss** function

Loss function for softmax

softmax_cross_entropy_with_logits(
logits, labels, name=<optional-name>)

```
labels = tf.placeholder("float", [1, 3])  
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(  
    relu_out, labels, name='xentropy')
```

Learn parameters: Optimization

Gradient descent

`class GradientDescentOptimizer`

`GradientDescentOptimizer(learning rate)`

`learning rate = 0.1`

```
labels = tf.placeholder("float", [1, 3])
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(
    relu_out, labels, name='xentropy')
optimizer = tf.train.GradientDescentOptimizer(0.1)
train_op = optimizer.minimize(cross_entropy)
sess.run(train_op,
          feed_dict= {x:np.array([[1.0, 2.0, 3.0]]), labels:answer})
```


Iterative update

Gradient descent usually needs more than one step

Run multiple times

```
labels = tf.placeholder("float", [1, 3])
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(
    relu_out, labels, name='xentropy')
optimizer = tf.train.GradientDescentOptimizer(0.1)
train_op = optimizer.minimize(cross_entropy)
for step in range(10):
    sess.run(train_op,
              feed_dict= {x:np.array([[1.0, 2.0, 3.0]]), labels:answer})
```

Add parameters for Softmax

Do not want to use only non-negative input

Softmax layer

```
...  
softmax_w = tf.Variable(tf.random_normal([3, 3]))  
logit = tf.matmul(relu_out, softmax_w)  
softmax = tf.nn.softmax(logit)  
...  
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(  
    logit, labels, name='xentropy')  
...
```

Add biases

Biases initialized to zero

```
...  
w = tf.Variable(tf.random_normal([3, 3]))  
b = tf.Variable(tf.zeros([1, 3]))  
relu_out = tf.nn.relu(tf.matmul(x, w) + b)  
softmax_w = tf.Variable(tf.random_normal([3, 3]))  
softmax_b = tf.Variable(tf.zeros([1, 3]))  
logit = tf.matmul(relu_out, softmax_w) + softmax_b  
softmax = tf.nn.softmax(logit)  
...
```

Make it deep

Add layers

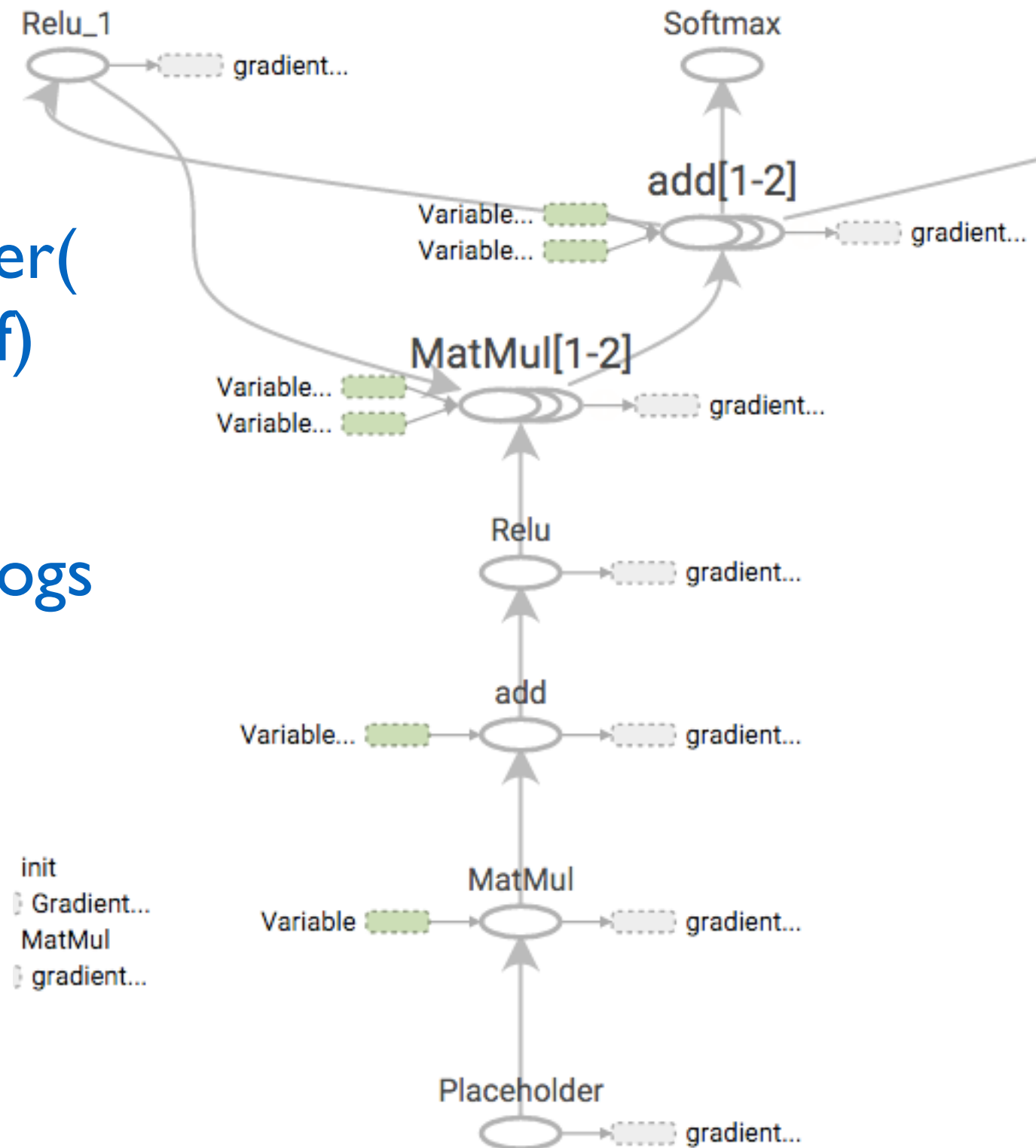
```
...  
x = tf.placeholder("float", [1, 3])  
relu_out = x  
num_layers = 2  
for layer in range(num_layers):  
    w = tf.Variable(tf.random_normal([3, 3]))  
    b = tf.Variable(tf.zeros([1, 3]))  
    relu_out = tf.nn.relu(tf.matmul(relu_out, w) + b)  
...
```

Visualize the graph

TensorBoard

```
writer = tf.train.SummaryWriter(  
    '/tmp/tf_logs', sess.graph_def)
```

```
tensorboard --logdir=/tmp/tf_logs
```



Improve naming, improve visualization

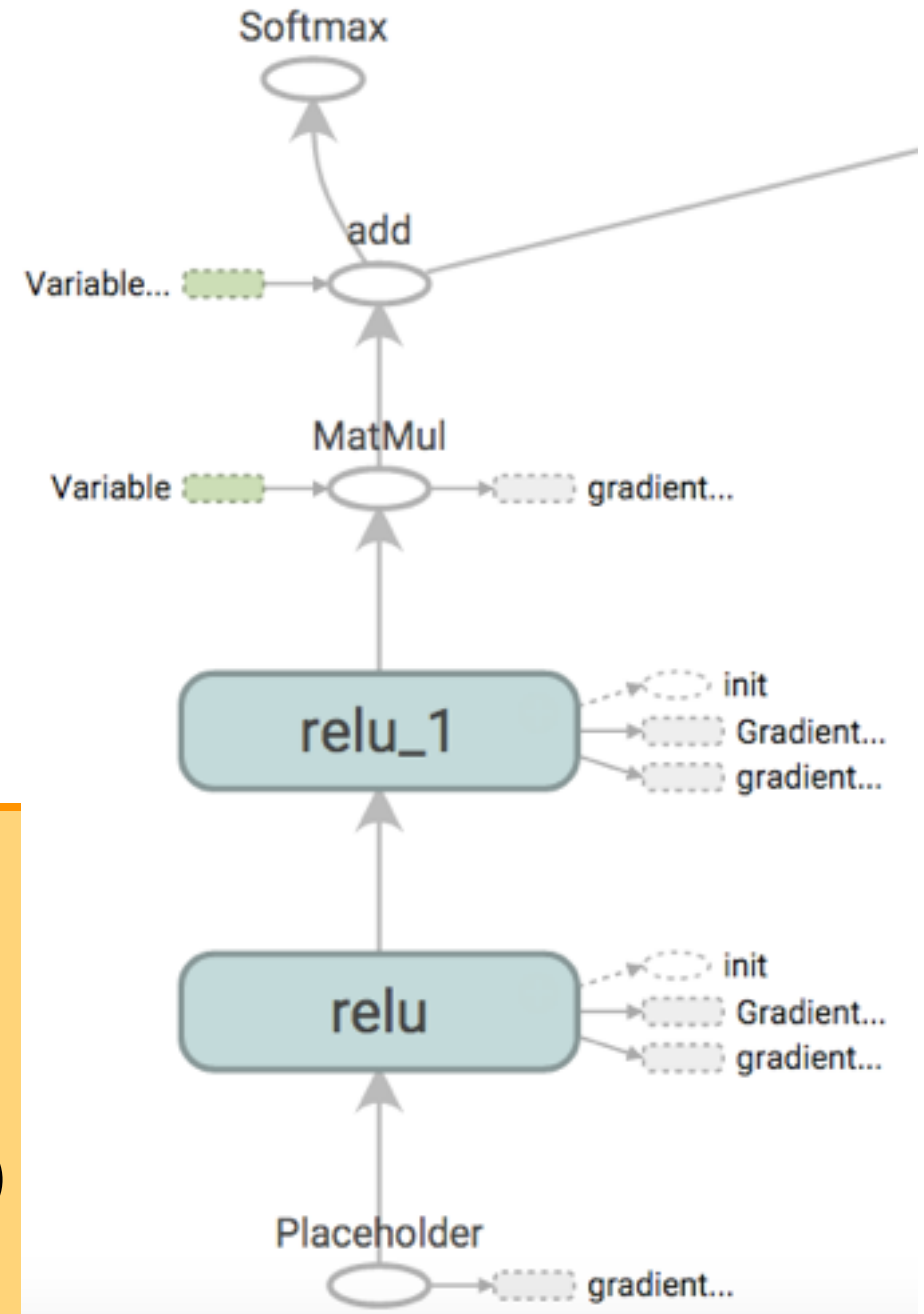
name_scope(name)

Help specify hierarchical names

Will help visualizer to better understand hierarchical relation

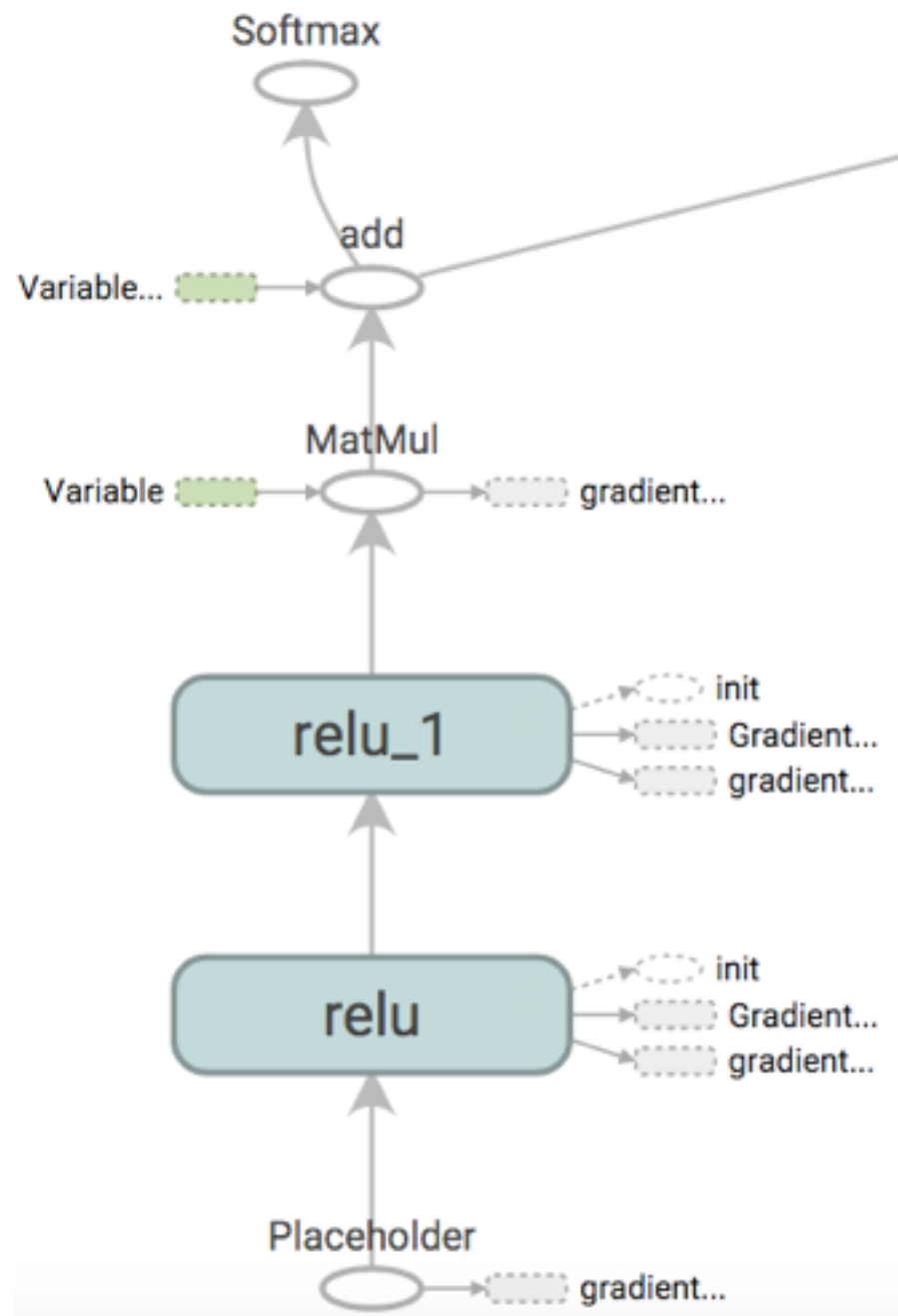
```
...  
for layer in range(num_layers):  
    with tf.name_scope('relu'):  
        w = tf.Variable(tf.random_normal([3, 3]))  
        b = tf.Variable(tf.zeros([1, 3]))  
        relu_out = tf.nn.relu(tf.matmul(relu_out, w) + b)  
...
```

Move to outside the loop?

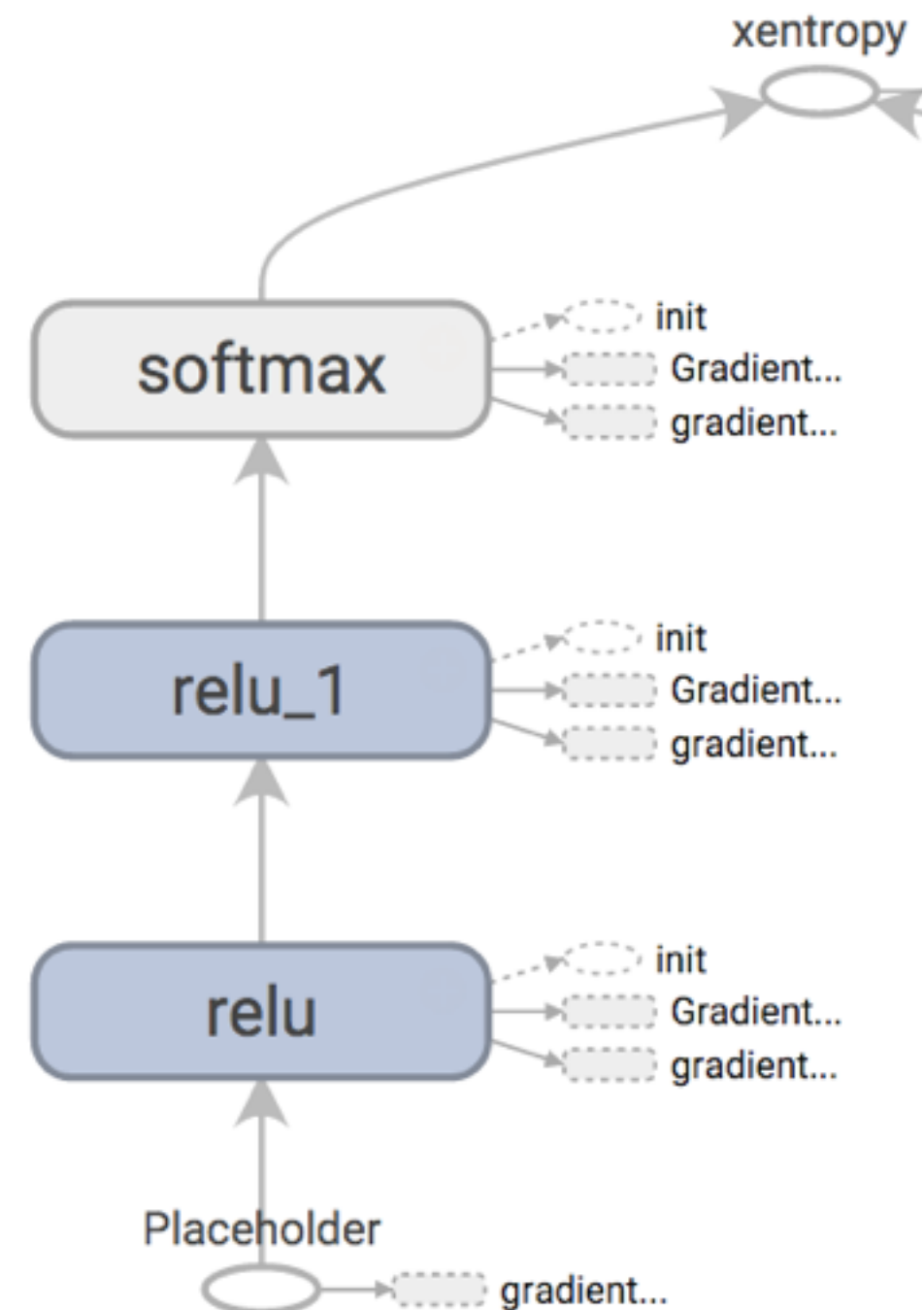


Add name_scope for softmax

Before



After



Add regularization to the loss

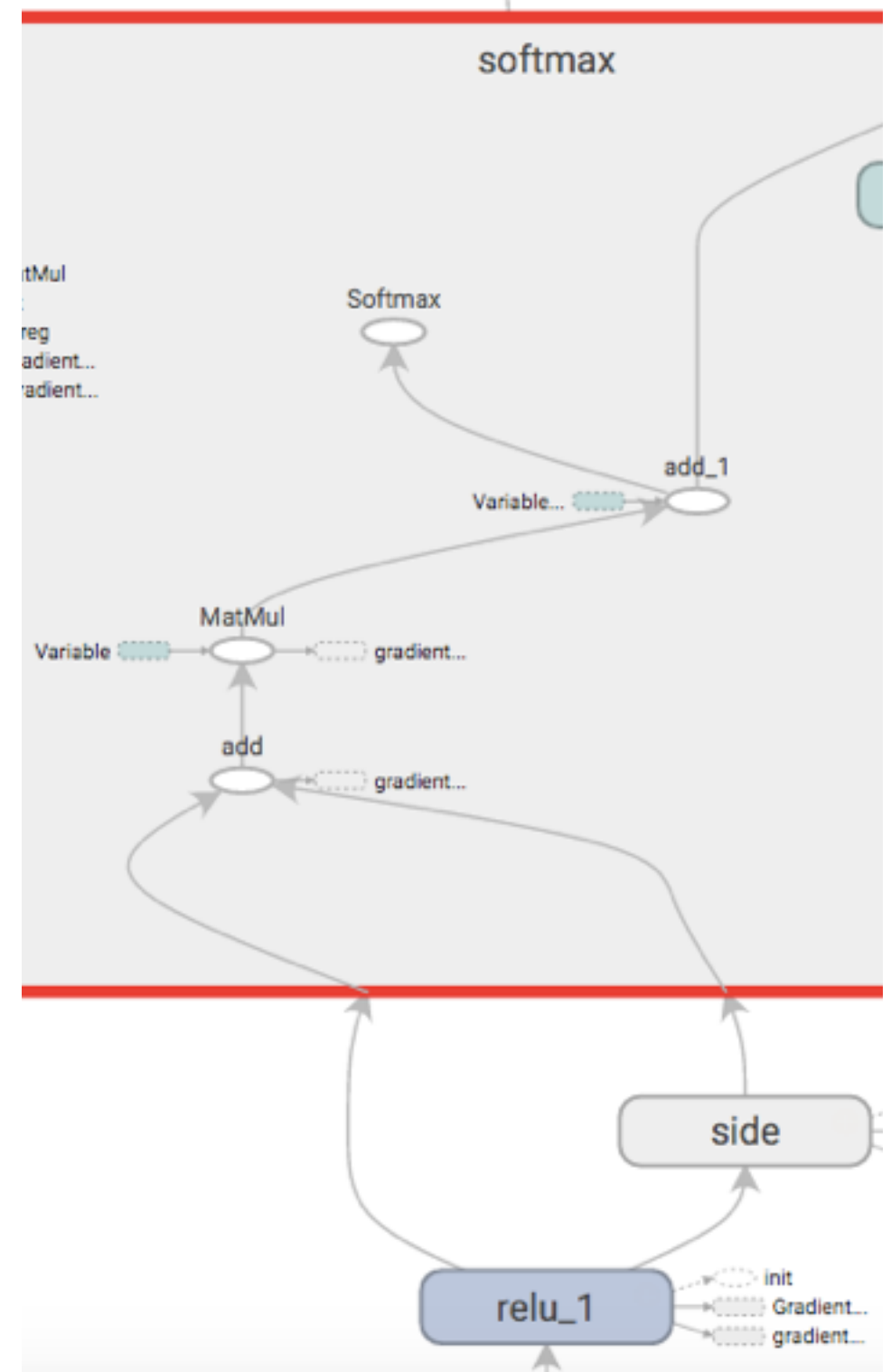
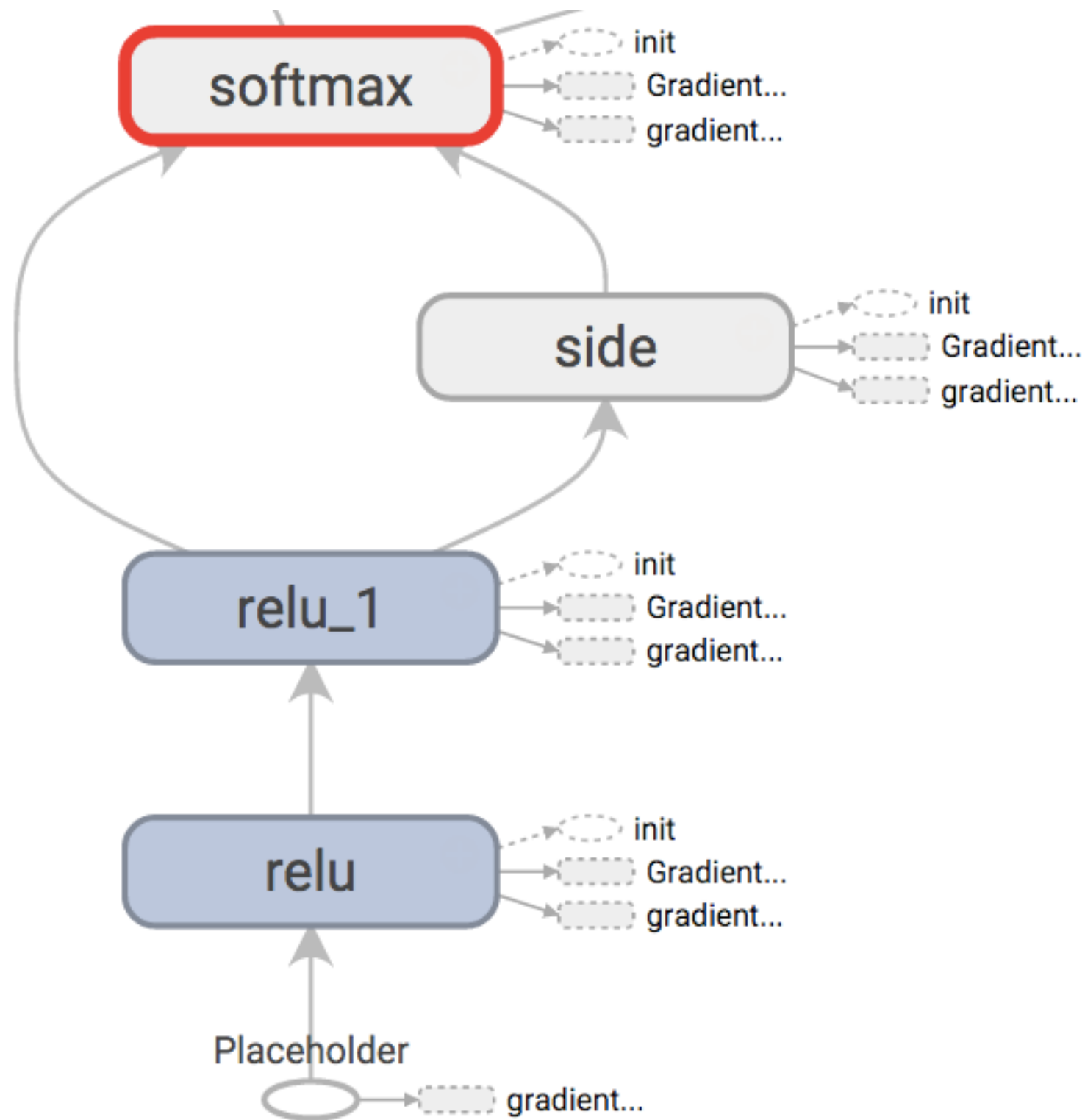
eg. L2 regularize on the Softmax layer parameters

Add it to the loss

Automatic gradient calculation

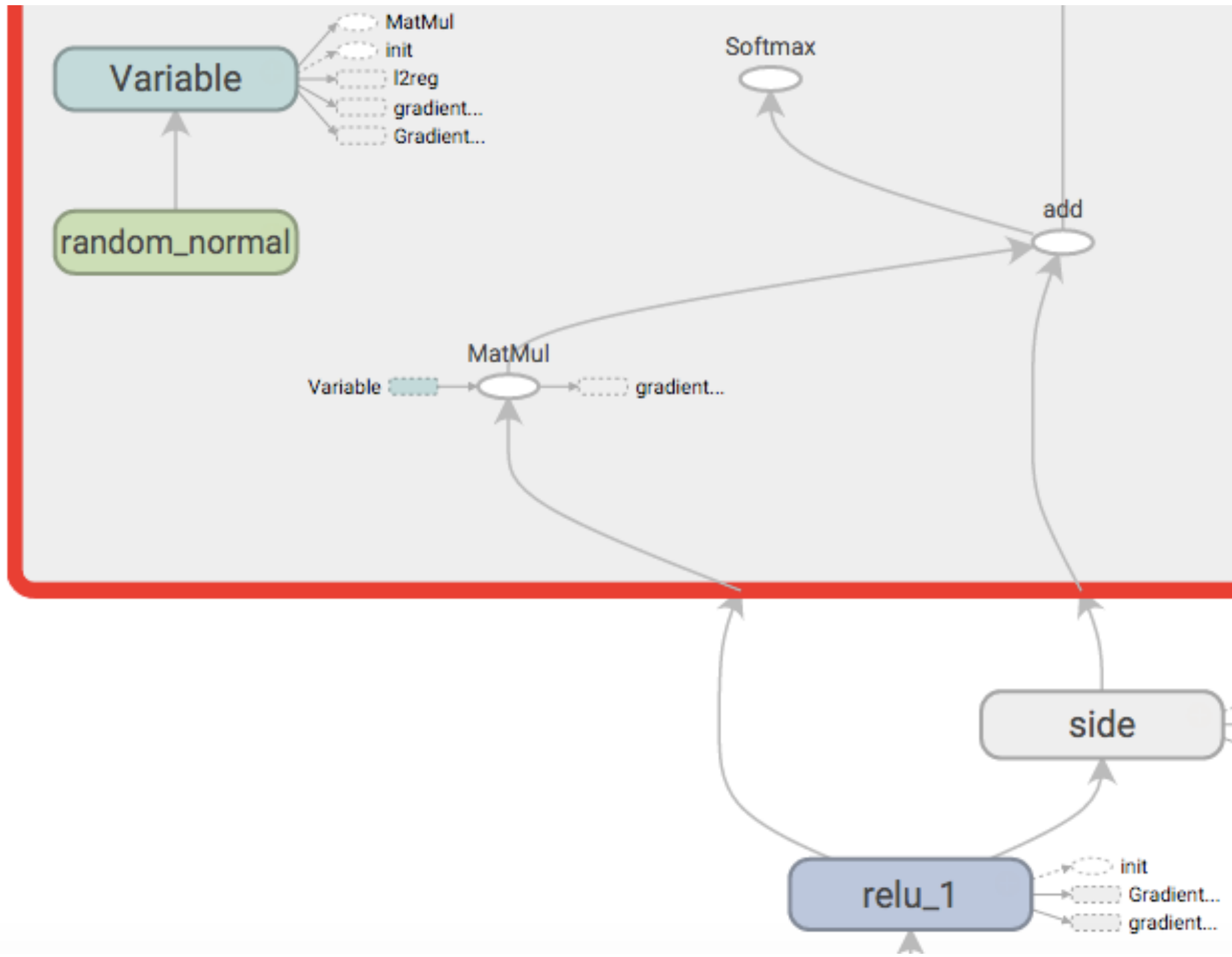
```
...  
l2reg = tf.reduce_sum(tf.square(softmax_w))  
loss = cross_entropy + l2reg  
train_op = optimizer.minimize(loss)  
...  
print sess.run(l2reg)  
...
```


Add a parallel path



Use activation as bias

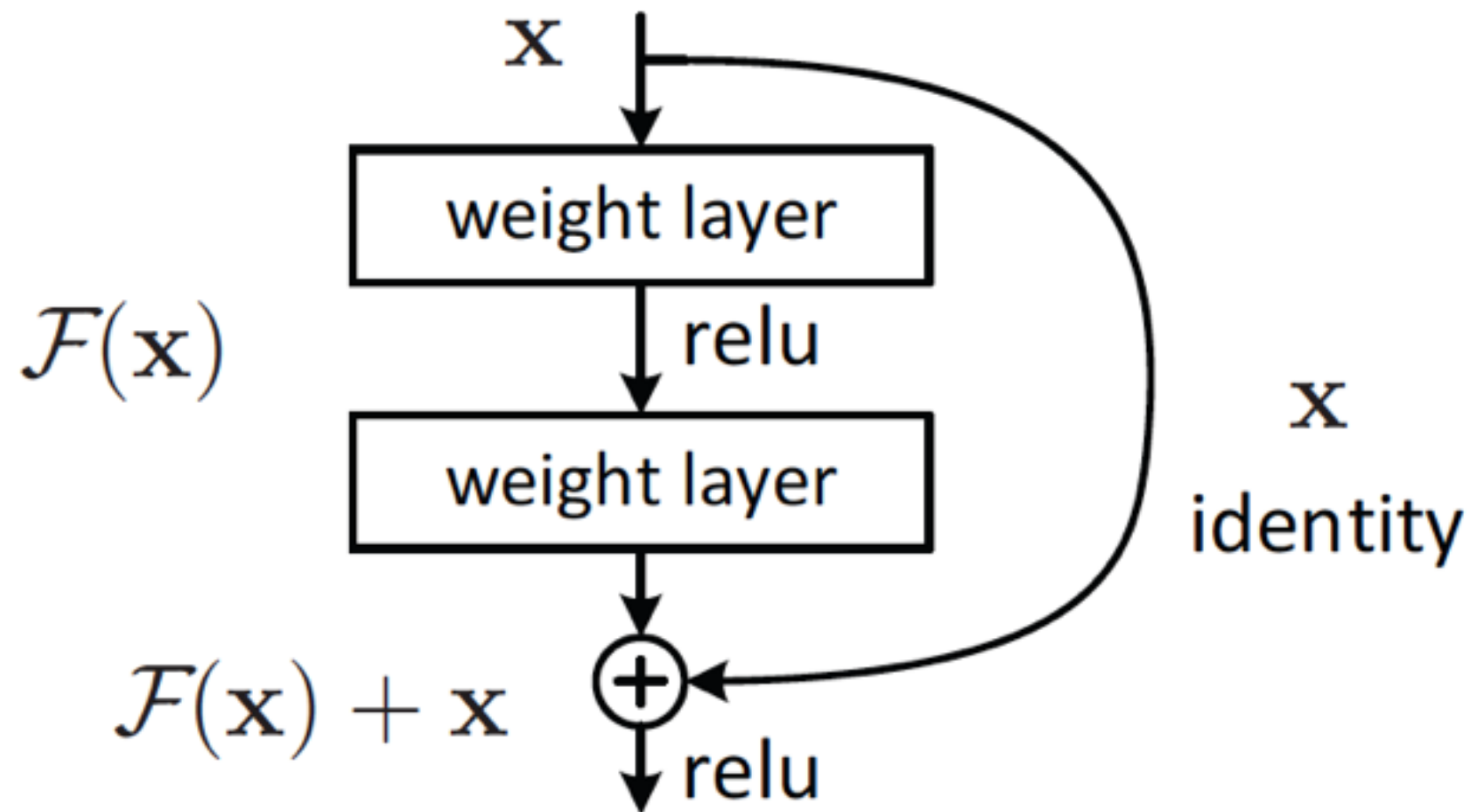
Everything is a tensor



Residual learning

He et al. 2015

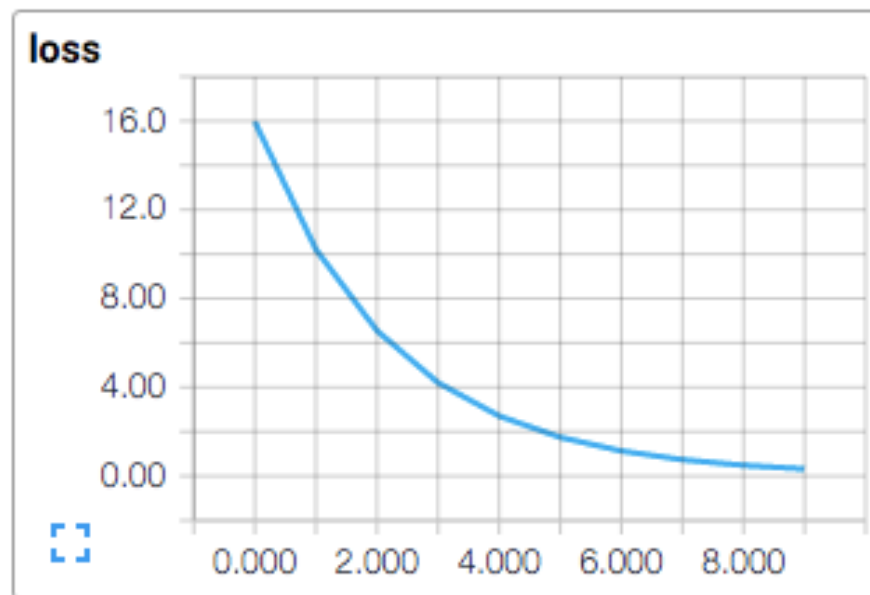
ILSVRC 2015 classification task winner



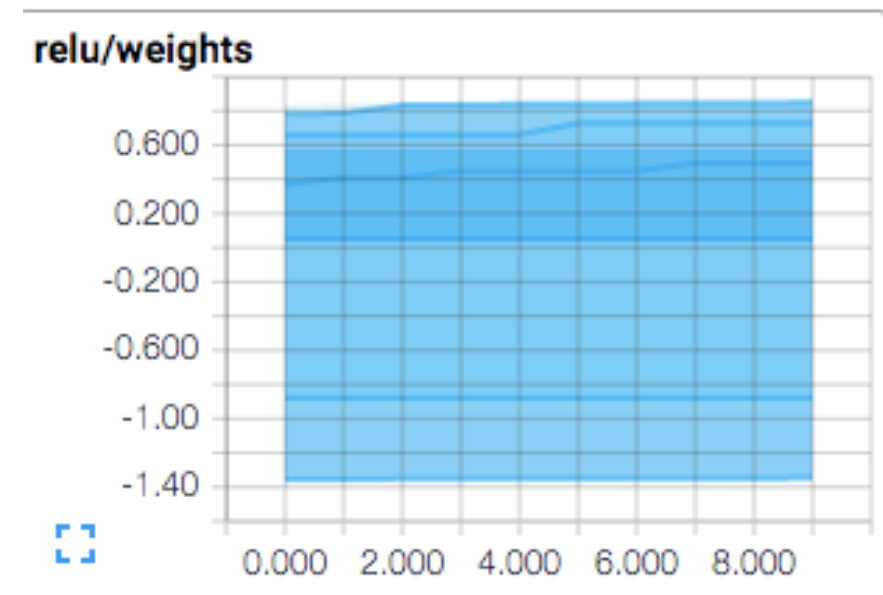
Visualize states

Add summaries

`scalar_summary`



`histogram_summary`



```
merged_summaries = tf.merge_all_summaries()  
results = sess.run([train_op, merged_summaries],  
                    feed_dict=...)  
writer.add_summary(results[1], step)
```

Save and load models

`tf.train.Saver(...)`

Default will associate with all variables
`all_variables()`

`save(sess, save_path, ...)`

`restore(sess, save_path, ...)`

Replace initialization

That's why we need to run initialization
separately

Convolution

```
conv2d(input, filter, strides, padding,  
       use_cudnn_on_gpu=None, name=None)
```

LSTM

BasicLSTMCell

$$i_t = W_{ix}x_t + W_{ih}h_{t-1} + b_i$$

$$j_t = W_{jx}x_t + W_{jh}h_{t-1} + b_j$$

$$f_t = W_{fx}x_t + W_{fh}h_{t-1} + b_f$$

$$o_t = W_{ox}x_t + W_{oh}h_{t-1} + b_o$$

$$c_t = \sigma(f_t) \odot c_{t-1} + \sigma(i_t) \odot \tanh(j_t)$$

$$h_t = \sigma(o_t) \odot \tanh(c_t)$$

Parameters of gates are concatenated into one multiply for efficiency.

```
c, h = array_ops.split(1, 2, state)
```

```
concat = linear([inputs, h], 4 * self._num_units, True)
```

```
# i = input_gate, j = new_input, f = forget_gate, o = output_gate
```

```
i, j, f, o = array_ops.split(1, 4, concat)
```

```
new_c = c * sigmoid(f + self._forget_bias) + sigmoid(i) * tanh(j)
```

```
new_h = tanh(new_c) * sigmoid(o)
```

Word2Vec with TensorFlow

```
# Look up embeddings for inputs.
embeddings = tf.Variable(
    tf.random_uniform([vocabulary_size, embedding_size], -1.0, 1.0))
embed = tf.nn.embedding_lookup(embeddings, train_inputs)
# Construct the variables for the NCE loss
nce_weights = tf.Variable(
    tf.truncated_normal([vocabulary_size, embedding_size],
                        stddev=1.0 / math.sqrt(embedding_size)))
nce_biases = tf.Variable(tf.zeros([vocabulary_size]))
# Compute the average NCE loss for the batch.
# tf.nce_loss automatically draws a new sample of the negative labels each
# time we evaluate the loss.
loss = tf.reduce_mean(
    tf.nn.nce_loss(nce_weights, nce_biases, embed, train_labels,
                  num_sampled, vocabulary_size))
```


Reuse Pre-trained models

Image recognition

Inception-v3

military uniform (866): 0.647296

suit (794): 0.0477196

academic gown (896): 0.0232411

bow tie (817): 0.0157356

bolo tie (940): 0.0145024



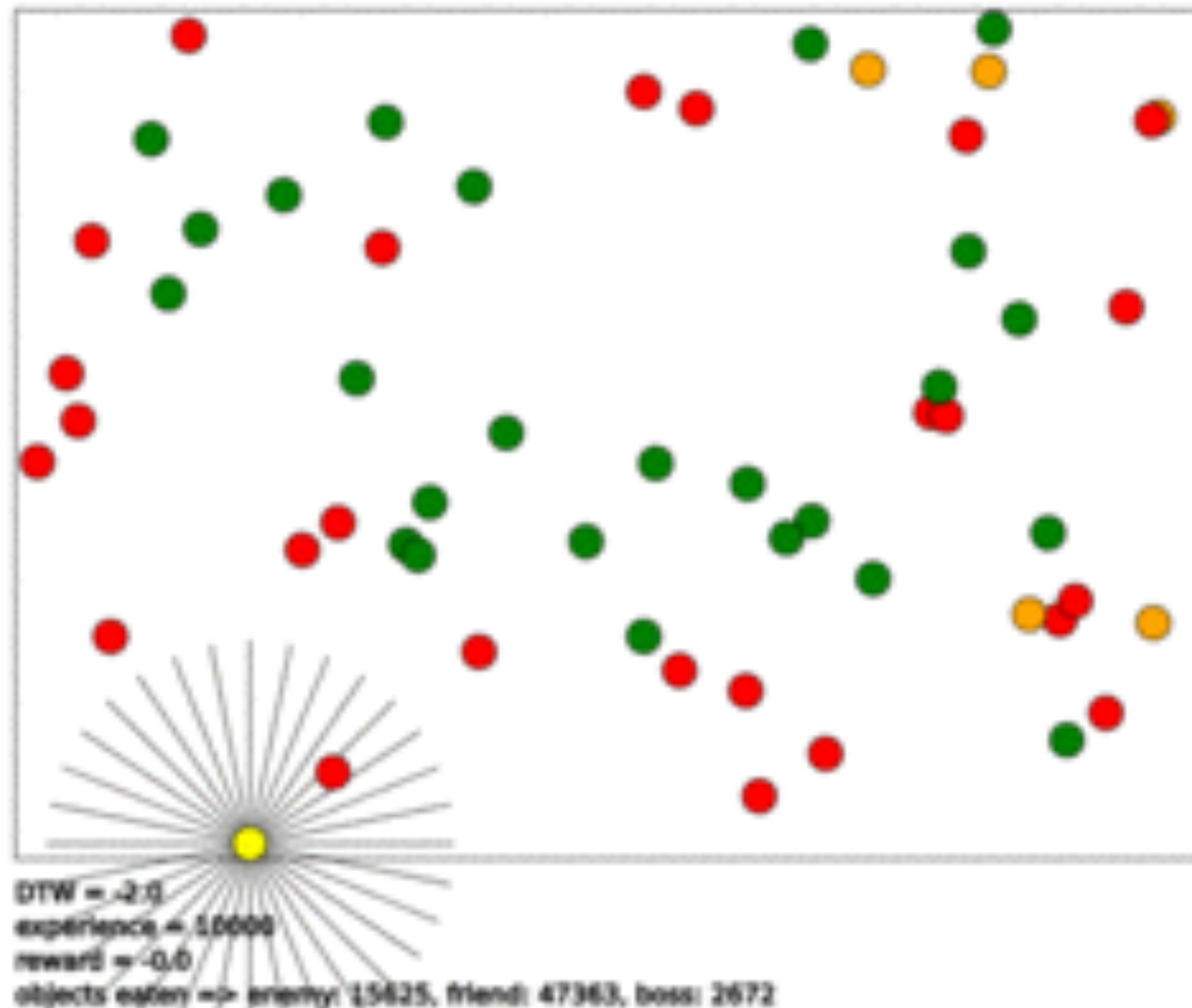
Try it on your Android

Tensorflow Android Camera Demo

Uses a Google Inception model to classify camera frames in real-time, displaying the top results in an overlay on the camera image.

[github.com/tensorflow/tensorflow/tree/master/tensorflow/
examples/android](https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/android)

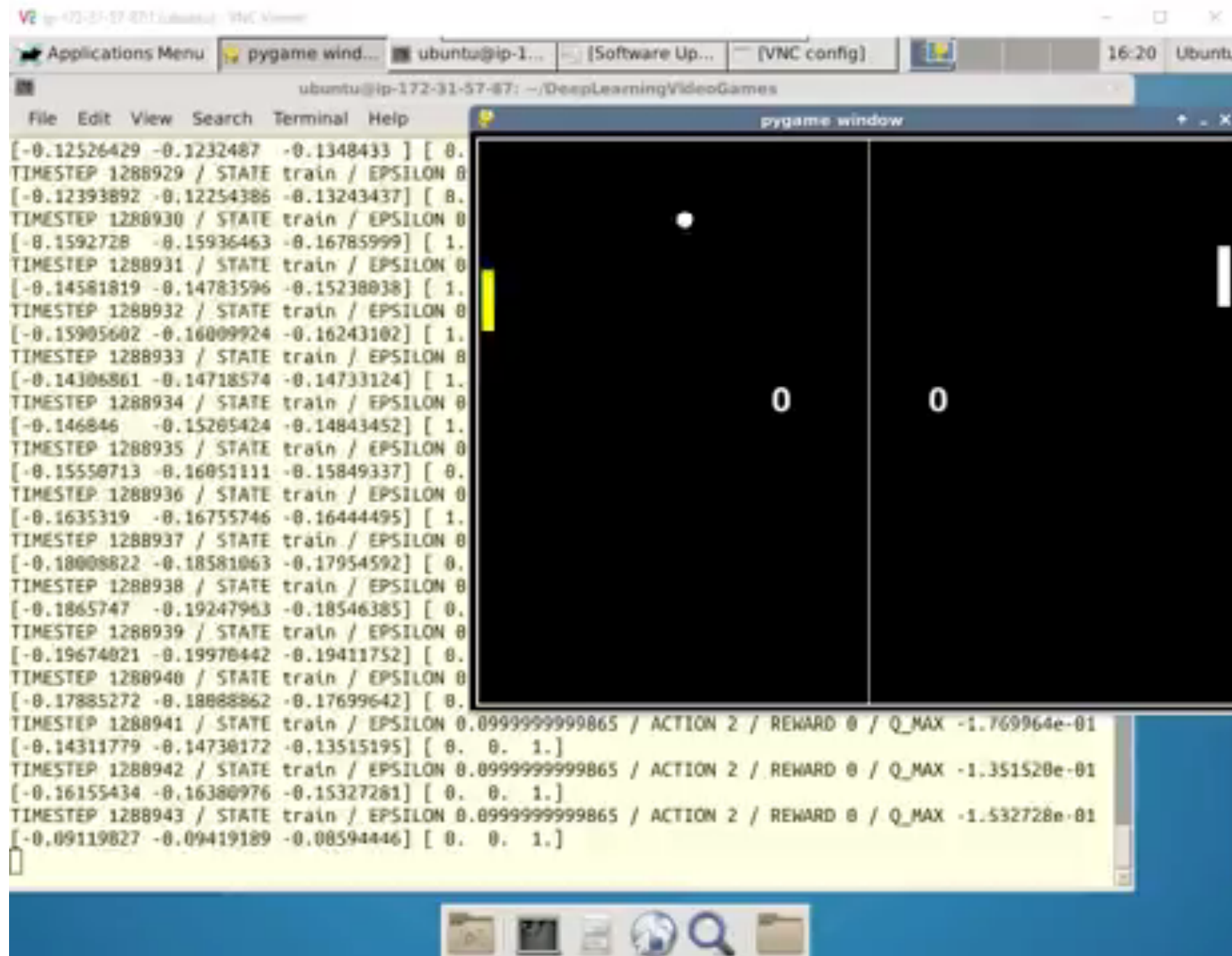
Reinforcement Learning using Tensor Flow



github.com/nivwusquorum/tensorflow-deepq

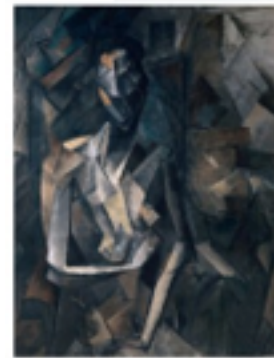


Using Deep Q Networks to Learn Video Game Strategies



github.com/asrivatl/DeepLearningVideoGames

Neural art





char-rnn-tensorflow

Multi-layer Recurrent Neural Networks (LSTM, RNN) for character-level language models in Python using Tensorflow.

Inspired from Andrej Karpathy's [char-rnn](#).

Requirements

- [Tensorflow](#)

Basic Usage

To train with default parameters on the tinyshakespeare corpus, run `python train.py`.

To sample from a checkpointed model, `python sample.py`.

github.com/sherjilozair/char-rnn-tensorflow



Keras: Deep Learning library for Theano and TensorFlow

You have just found Keras.

Keras is a minimalist, highly modular neural networks library, written in Python and capable of running either on top of either [TensorFlow](#) or [Theano](#). It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

Use Keras if you need a deep learning library that:

- allows for easy and fast prototyping (through total modularity, minimalism, and extensibility).
- supports both convolutional networks and recurrent networks, as well as combinations of the two.
- supports arbitrary connectivity schemes (including multi-input and multi-output training).
- runs seamlessly on CPU and GPU.

Read the documentation at [Keras.io](#).

Keras is compatible with: - **Python 2.7-3.5** with the Theano backend - **Python 2.7** with the TensorFlow backend

github.com/fchollet/keras



Neural Caption Generator

- Implementation of "Show and Tell" <http://arxiv.org/abs/1411.4555>
 - Borrowed some code and ideas from Andrej Karpathy's NeuralTalk.
- You need flickr30k data (images and annotations)

Code

- `make_flickr_dataset.py` : Extracting feats of flickr30k images, and save them in './data/feats.npy'
- `model_tensorflow.py` : TensorFlow Version
- `model_theano.py` : Theano Version

Usage

- Flickr30k Dataset Download
- Extract VGG Features of Flickr30k images (`make_flickr_dataset.py`)
- Train: run `train()` in `model_tensorflow.py` or `model_theano.py`
- Test: run `test()` in `model_tensorflow.py` or `model_theano.py`.
 - parameters: VGG FC7 feature of test image, trained model path

github.com/jazzsaxmafia/show_and_tell.tensorflow



你正在阅读的项目可能会比 **Android** 系统更加深远地影响着世界！

缘起

2015年11月9日，Google发布人工智能系统TensorFlow并宣布开源，同日，极客学院组织在线TensorFlow中文文档翻译。

机器学习作为人工智能的一种类型，可以让软件根据大量的数据来对未来的情况进行阐述或预判。如今，领先的科技巨头无不在机器学习下予以极大投入。Facebook、苹果、微软，甚至国内的百度。Google 自然也在其中。「TensorFlow」是 Google 多年以来内部的机器学习系统。如今，Google 正在将此系统成为开源系统，并将此系统的参数公布给业界工程师、学者和拥有大量编程能力的技术人员，这意味着什么呢？

github.com/jikexueyuanwiki/tensorflow-zh



Google Brain Residency Program

New one year immersion program in deep learning research

Learn to conduct deep learning research w/experts in our team

Fixed one-year employment with salary, benefits, ...

Goal after one year is to have conducted several research projects

Interesting problems, TensorFlow, and access to computational resources



Google Brain Residency Program

Who should apply?

People with BSc, MSc or PhD, ideally in CS, mathematics or statistics

Completed coursework in calculus, linear algebra, and probability, or equiv.

Programming experience

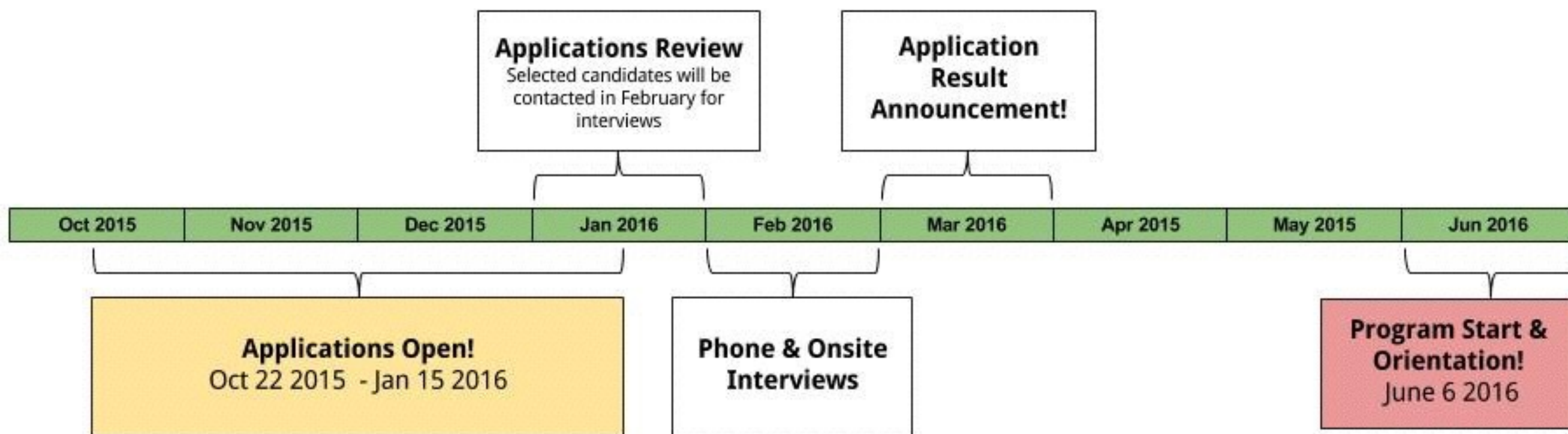
Motivated, hard working, and have a strong interest in deep learning



Google Brain Residency Program

Program Application & Timeline

DEADLINE: January 15, 2016



Thanks for your attention!