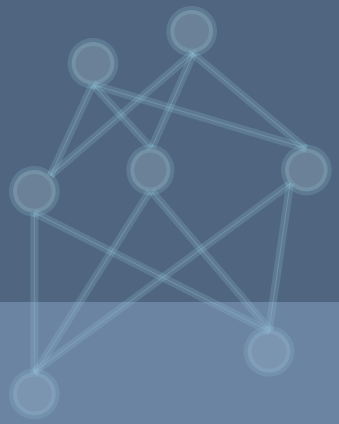
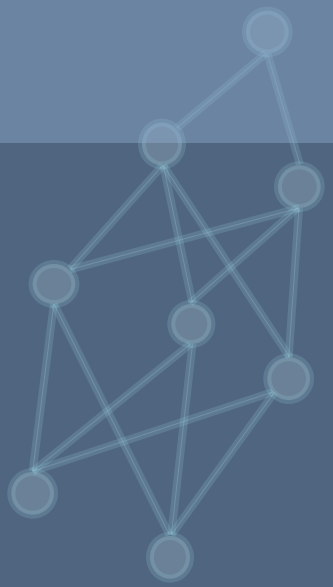


NTU Machine Learning Tensorflow

By Mark Chang



Outlines

- One-layer Perceptron
- Tensorboard
- Convolutional Neural Networks
- Multi-GPU Implementation

Tensorflow

TensorFlow

https://www.tensorflow.org/

How to Install Tensorflow

- Direct step:
  - install tensorflow

```
> sudo pip install tensorflow
```

- Install in virtual environment:
  - install pyenv
  - install anaconda-2.x.x
  - install tensorflow

```
> pip install tensorflow
```

1. install pyenv

- Mac OSX Homebrew

```
> brew update
> brew install pyenv
```

- From Github

(for ubuntu user, please replace .bash_profile with .bashrc)

```
> cd ~
> git clone https://github.com/yyuu/pyenv.git ~/.pyenv
> echo 'export PYENV_ROOT="$HOME/.pyenv"' >> ~/.bash_profile
> echo 'export PATH="$PYENV_ROOT/bin:$PATH"' >> ~/.bash_profile
> echo 'eval "$(pyenv init -)"' >> ~/.bash_profile
> source ~/.bash_profile
```

install anaconda-2.x.x and Tensorflow

- install anaconda-2.4.0

```
> pyenv install anaconda-2.4.0
```

- switch to anaconda 2.4.0

```
> pyenv global anaconda-2.4.0
```

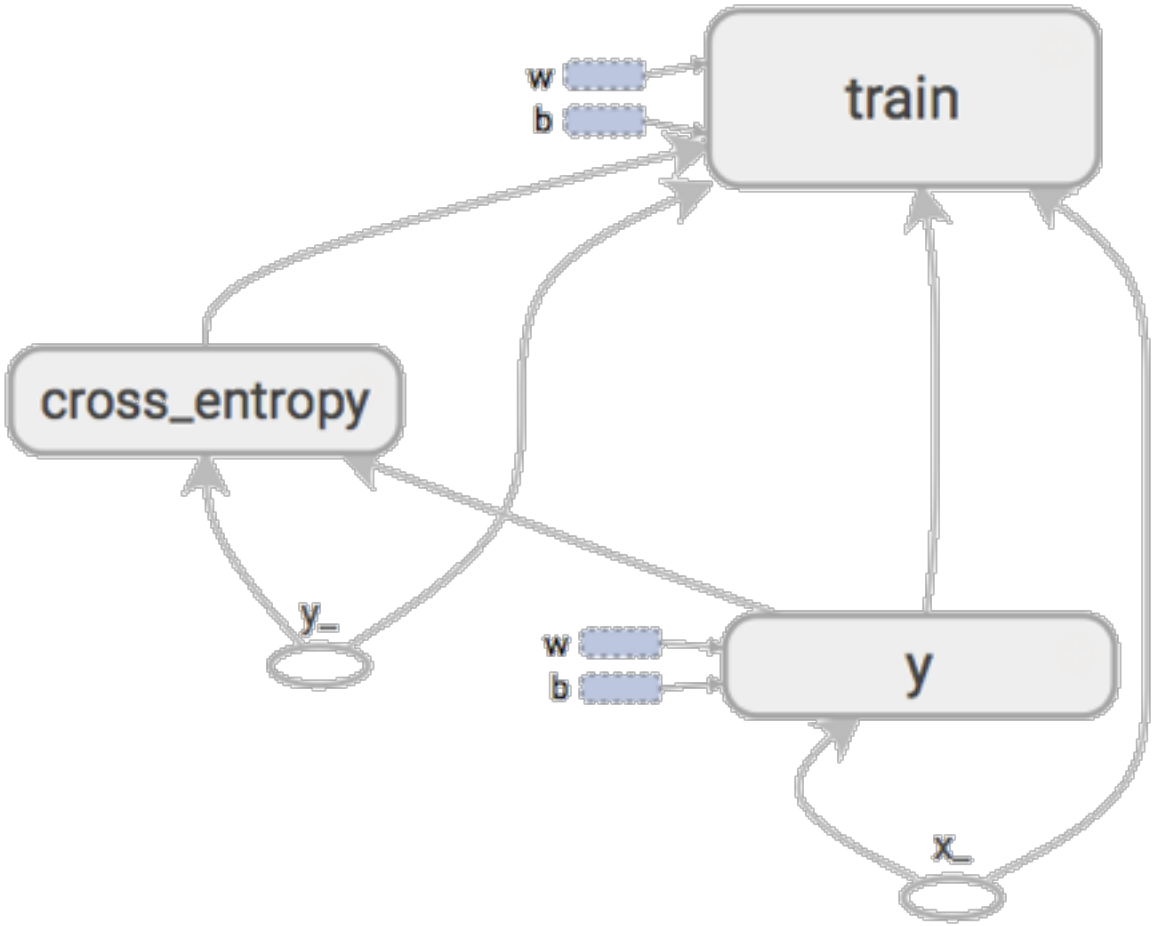
- Install tensorflow

```
> pip install tensorflow
```

Features of Tensorflow

- Computational Graph
  - Define computation
- Session
  - Run computation

Computational Graph

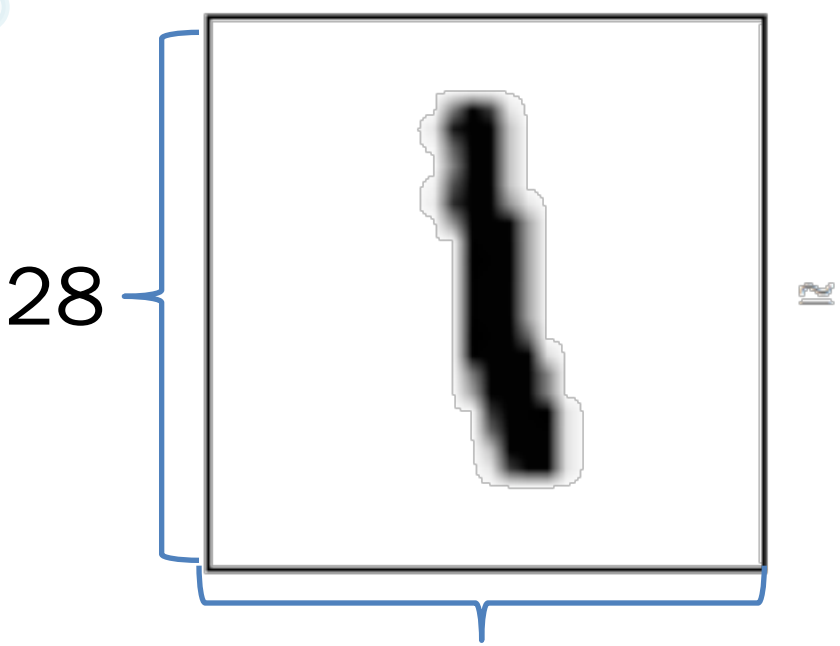


MNIST

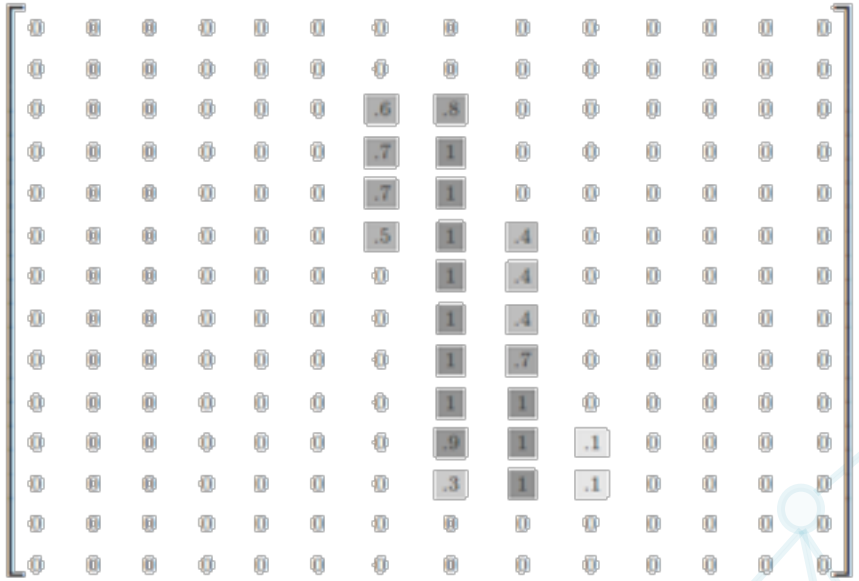
- Image Classification
- Multi-class : 0~9

https://www.tensorflow.org/versions/r0.7/images/MNIST.png

Training Data



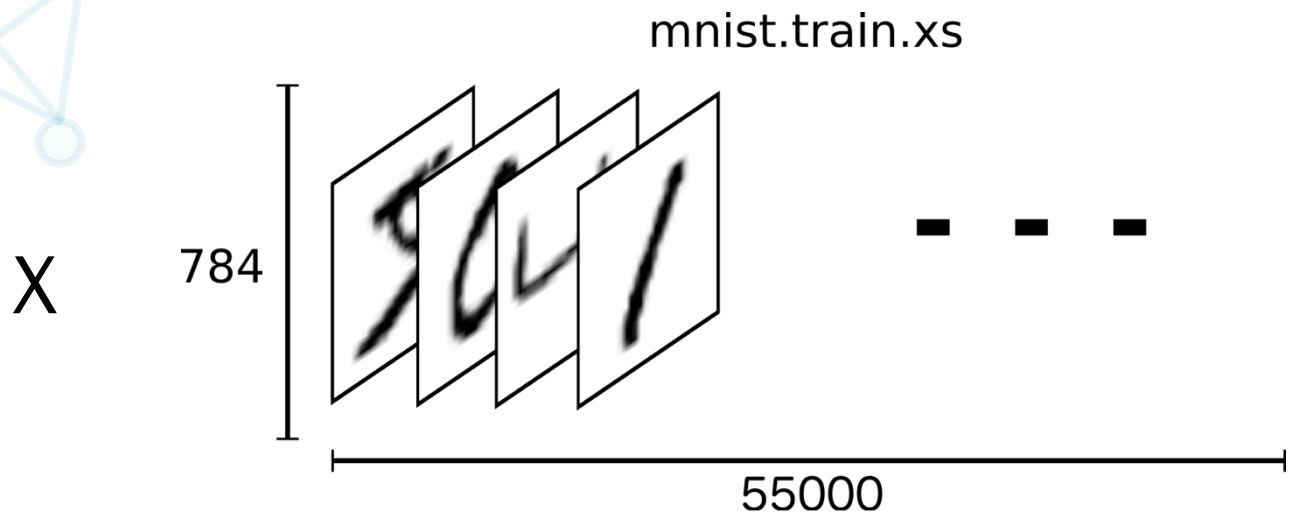
≈



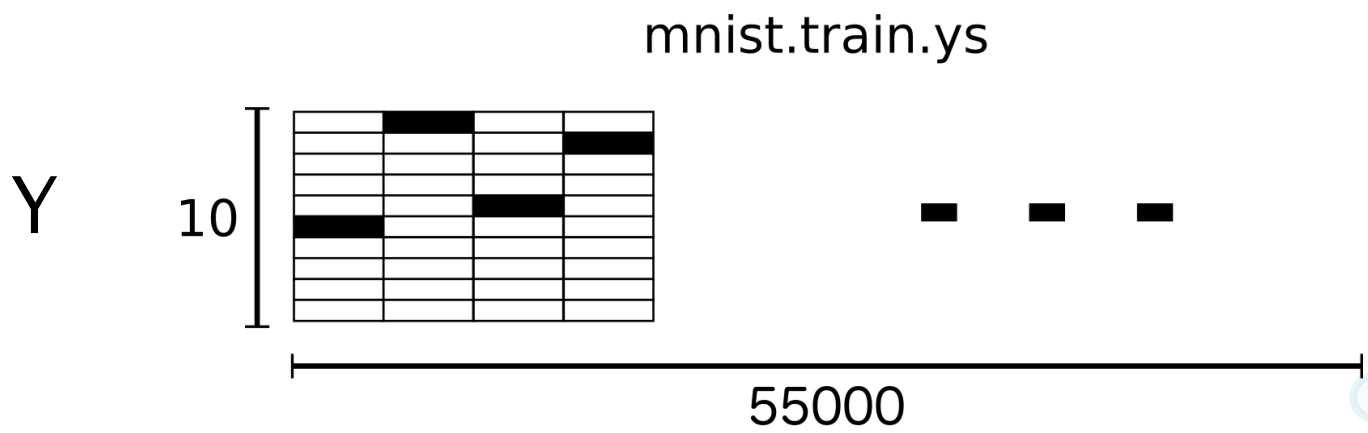
28x28 = 784

https://www.tensorflow.org/versions/r0.7/images/MNIST-Matrix.png

Training Data

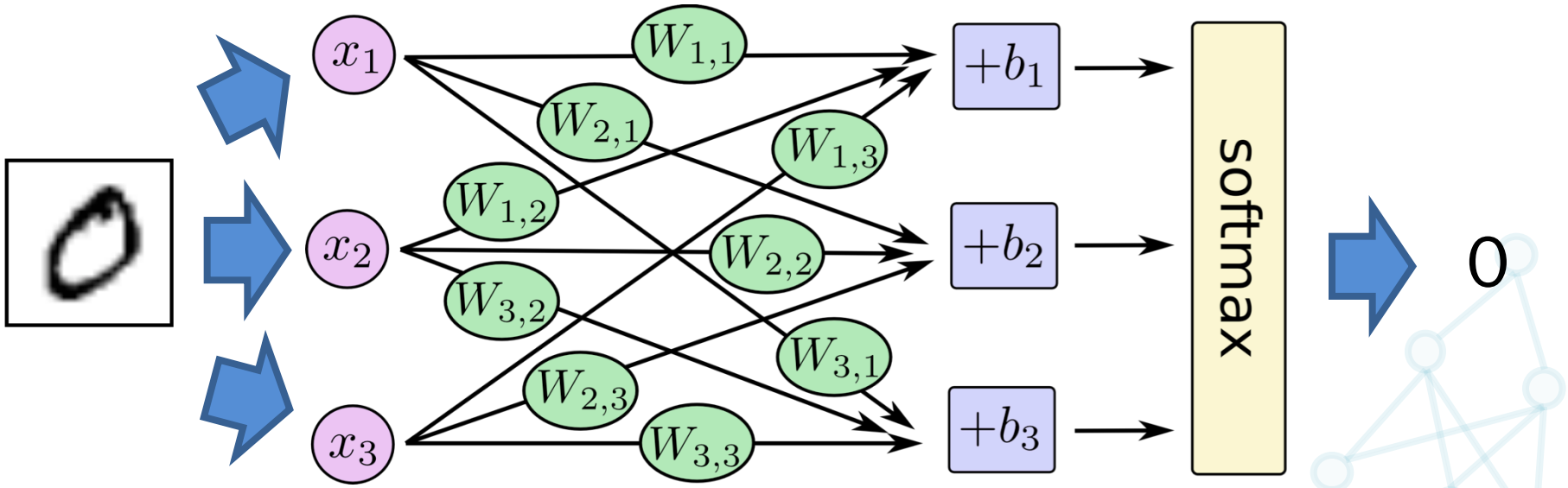


https://www.tensorflow.org/versions/r0.7/images/mnist-train-xs.png



https://www.tensorflow.org/versions/r0.7/images/mnist-train-ys.png

One-layer Perceptron



https://www.tensorflow.org/versions/r0.7/images/softmax-regression-scalargraph.png

One-layer Perceptron

- https://github.com/ckmarkoh/ntu_tensorflow/blob/master/one_layer_train.ipynb

One-layer Perceptron

Computational Graph

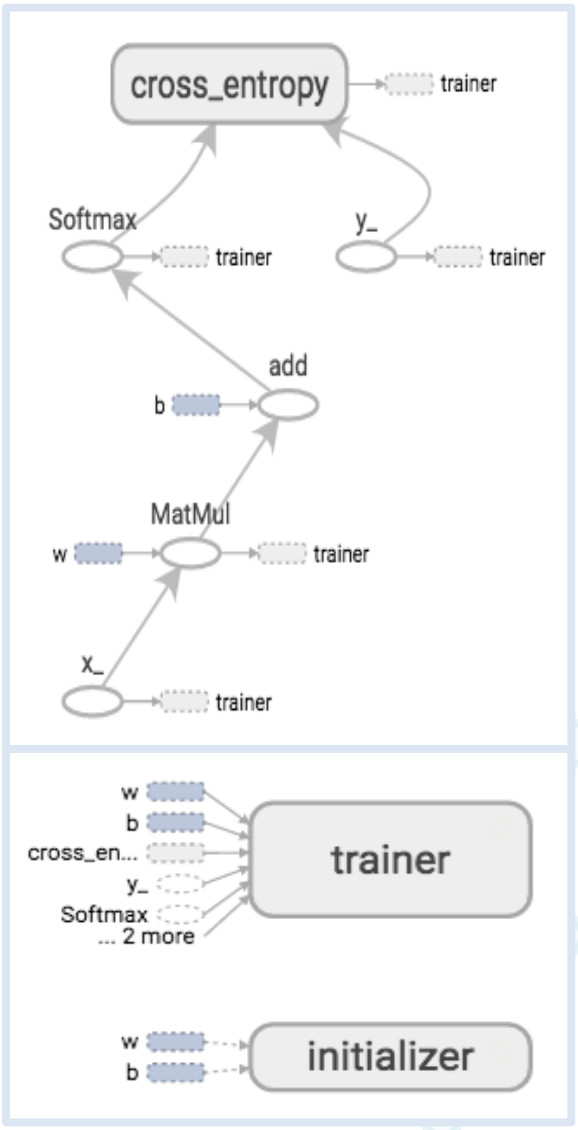
```
x_ = tf.placeholder(tf.float32, [None, 784])
y_ = tf.placeholder(tf.float32, [None, 10])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(x_, W) + b)
cross_entropy = -tf.reduce_mean(y_ * tf.log(y))
optimizer = tf.train.GradientDescentOptimizer(0.01)
trainer = optimizer.minimize(cross_entropy)
init = tf.global_variables_initializer()
```

Session

```
sess = tf.Session()
sess.run(init)
for i in range(1000):
        sess.run(train, feed_dict={x_:x_data,y_:y_data})
        print sess.run(cross_entropy, feed_dict={x_:x_data,y_:y_data})
sess.close()
```

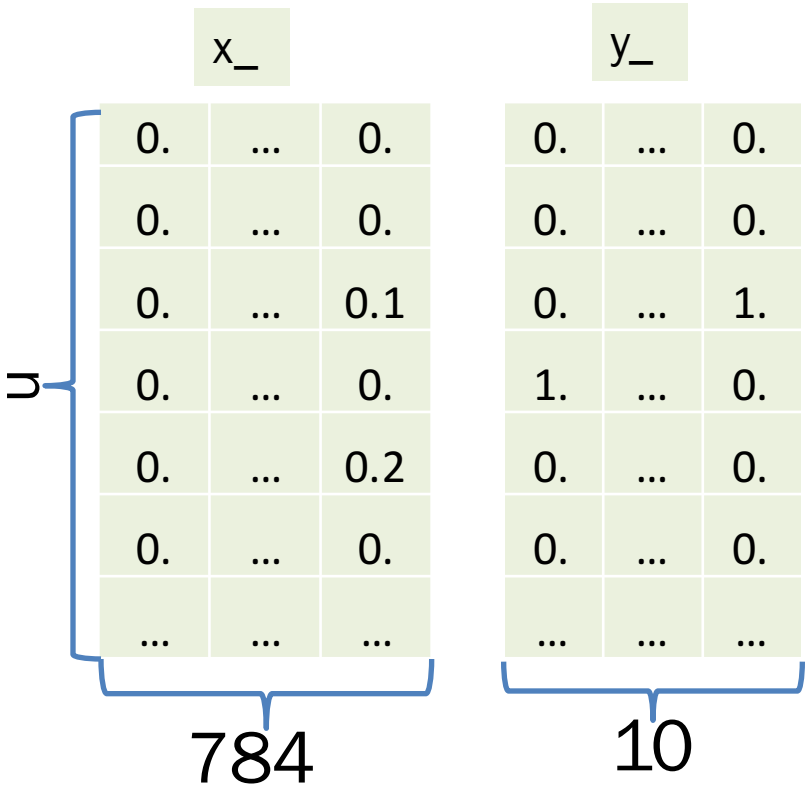
Computation Graph

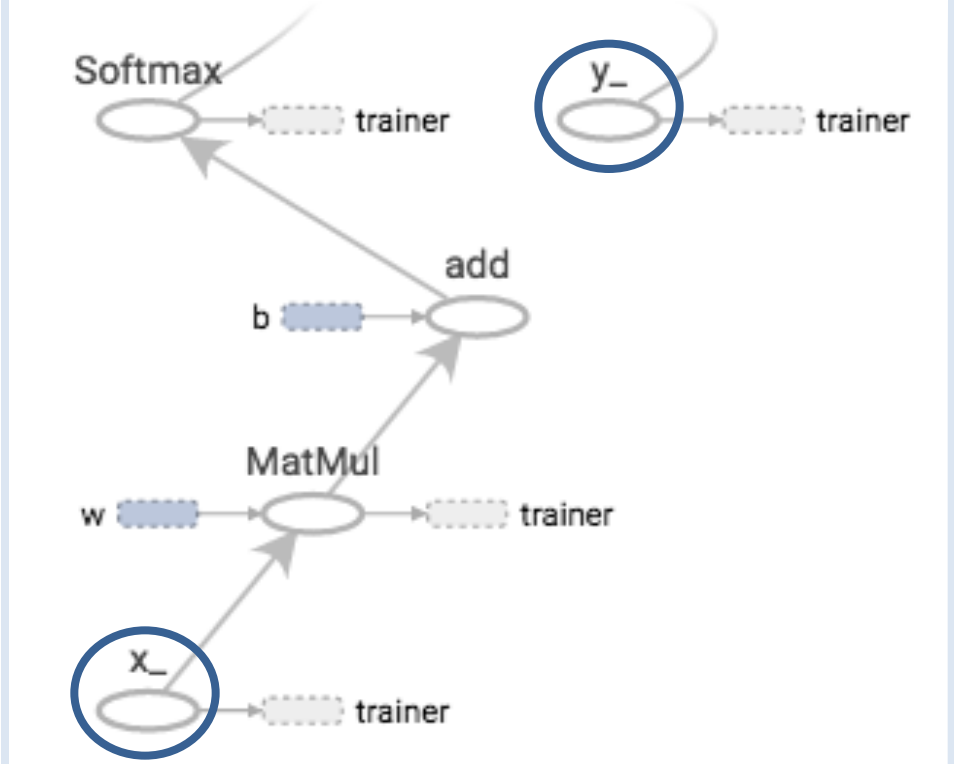
```
# placeholder
x_ = tf.placeholder(tf.float32, [None, 784])
y_ = tf.placeholder(tf.float32, [None, 10])
# variable
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
# operations
y = tf.nn.softmax(tf.matmul(x_, W) + b)
# error function
cross_entropy = -tf.reduce_sum(y_* tf.log(y))
# trainer
optimizer = tf.train.GradientDescentOptimizer(0.01)
trainer = optimizer.minimize(cross_entropy)
# initalizer
init = tf.global_variables_initializer()
```



Placeholder

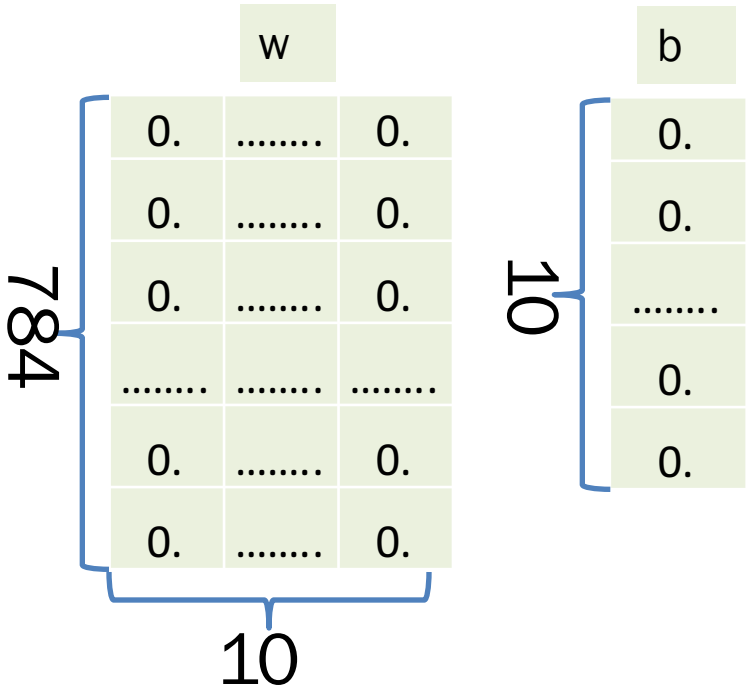
```
x_ = tf.placeholder(tf.float32, [None, 784])
y_ = tf.placeholder(tf.float32, [None, 10])
```

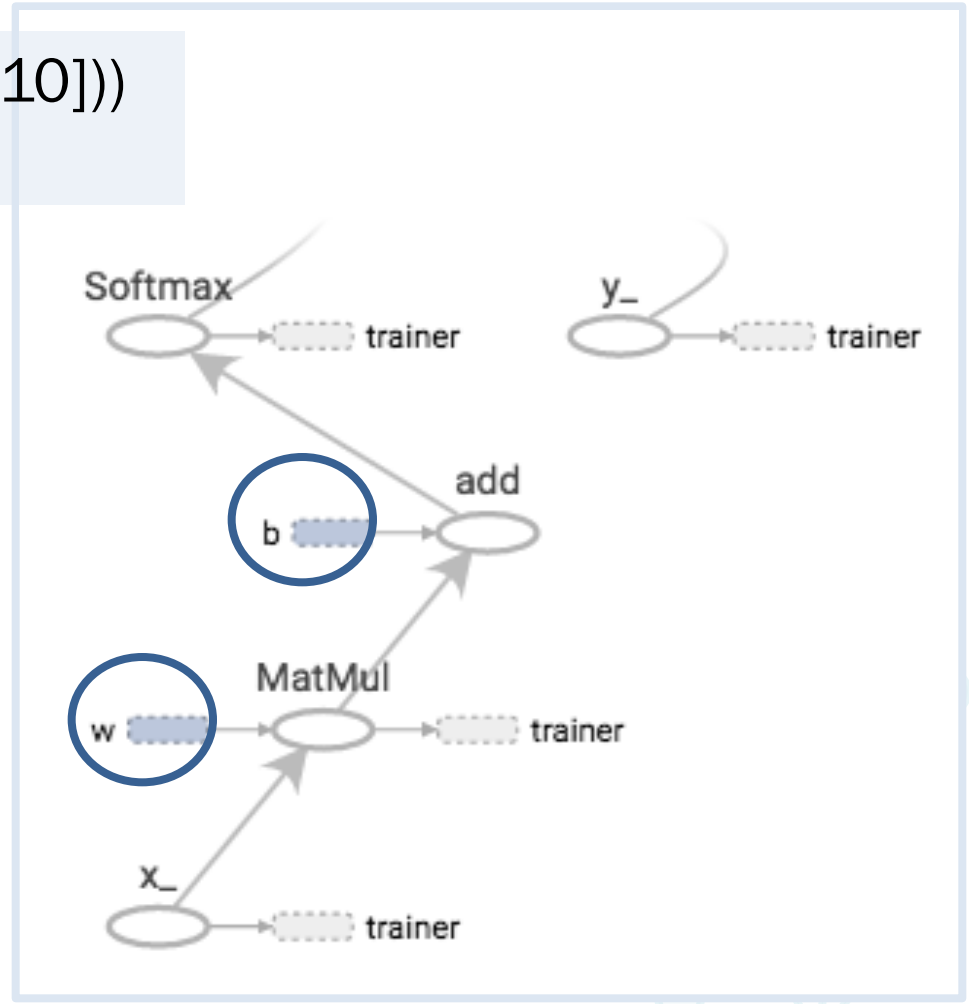




Variable

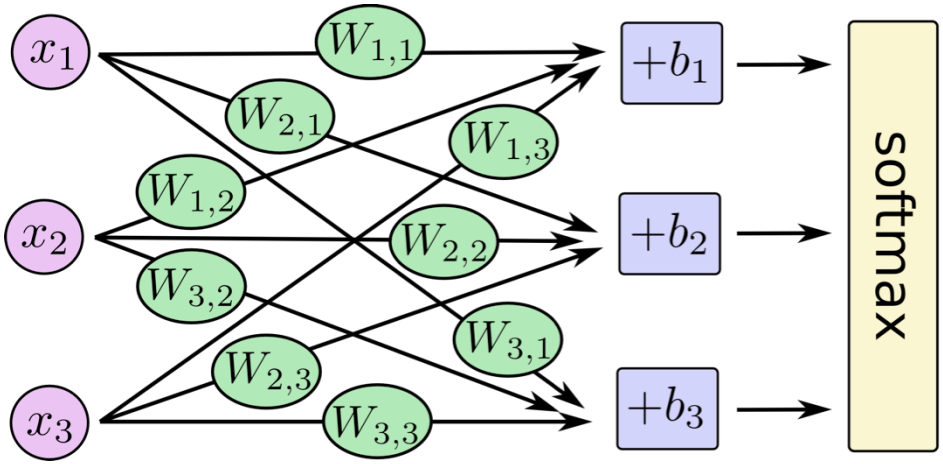
```
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
```





Matrix Multiplication

```
y = tf.nn.softmax(tf.matmul(x_, W) + b)
```



$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{matrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{matrix} \right)$$

https://www.tensorflow.org/versions/r0.8/images/softmax-regression-scalarequation.png

Matrix Multiplication

```
y = tf.nn.softmax(tf.matmul(x_, W) + b)
```

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{matrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{matrix} \right)$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

https://www.tensorflow.org/versions/r0.8/images/softmax-regression-vectorequation.png

Batch

- Improve parallelization
- Ex, batch size = 4 :

$$\mathbf{x}_{-} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} \\ x_1^{(3)} & x_2^{(3)} & x_3^{(3)} \\ x_1^{(4)} & x_2^{(4)} & x_3^{(4)} \end{bmatrix}$$

Matrix Multiplication with Batch

```
y = tf.nn.softmax(tf.matmul(x_, W) + b)
```

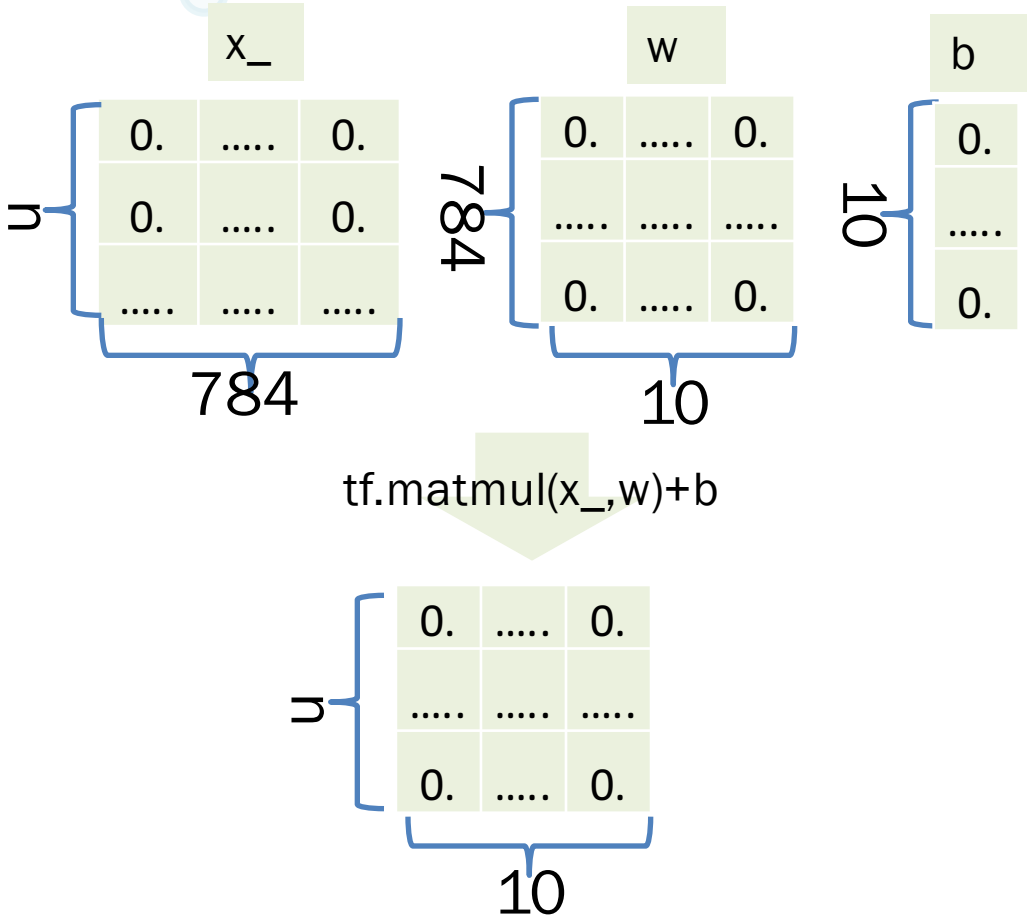
$$\mathbf{x\_} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} \\ x_1^{(3)} & x_2^{(3)} & x_3^{(3)} \\ x_1^{(4)} & x_2^{(4)} & x_3^{(4)} \end{bmatrix} \quad \mathbf{W} = \begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

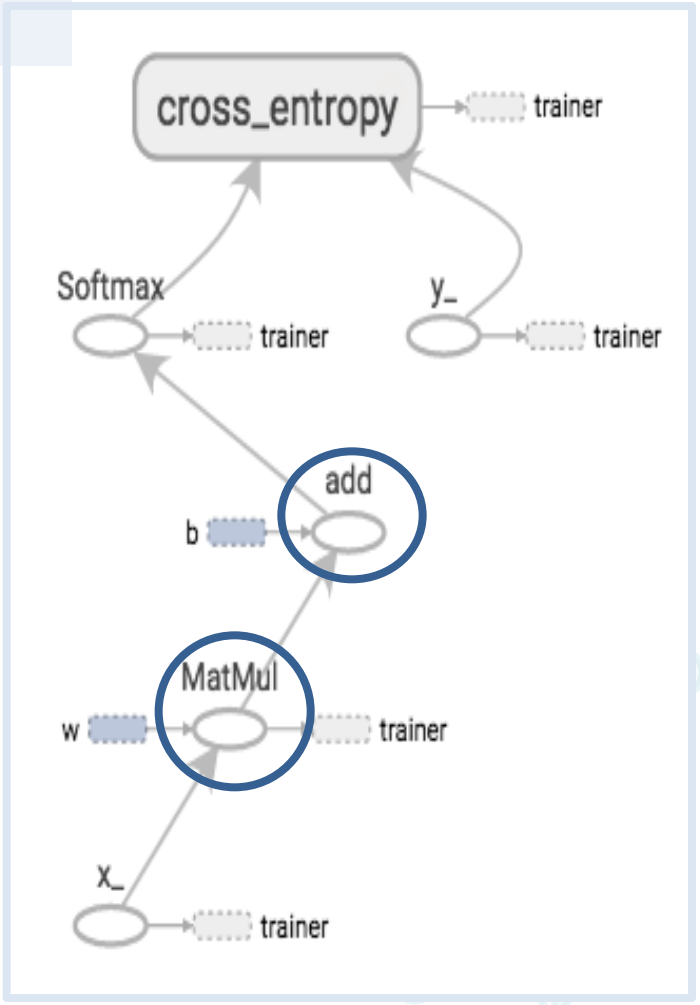
$$\mathrm{matmul}(\mathbf{x\_}, \mathbf{W}) + \mathbf{b}$$

$$= \begin{bmatrix} x_1^{(1)}W_{1,1} + x_2^{(1)}W_{2,1} + x_3^{(1)}W_{3,1} + b_1 & \cdots & x_1^{(1)}W_{1,3} + x_2^{(1)}W_{2,3} + x_3^{(1)}W_{3,3} + b_3 \\ x_1^{(2)}W_{1,1} + x_2^{(2)}W_{2,1} + x_3^{(2)}W_{3,1} + b_1 & \cdots & x_1^{(2)}W_{1,3} + x_2^{(2)}W_{2,3} + x_3^{(2)}W_{3,3} + b_3 \\ x_1^{(3)}W_{1,1} + x_2^{(3)}W_{2,1} + x_3^{(3)}W_{3,1} + b_1 & \cdots & x_1^{(3)}W_{1,3} + x_2^{(3)}W_{2,3} + x_3^{(3)}W_{3,3} + b_3 \\ x_1^{(4)}W_{1,1} + x_2^{(4)}W_{2,1} + x_3^{(4)}W_{3,1} + b_1 & \cdots & x_1^{(4)}W_{1,3} + x_2^{(4)}W_{2,3} + x_3^{(4)}W_{3,3} + b_3 \end{bmatrix}$$

Matrix Multiplication with Batch

y = tf.nn.softmax(tf.matmul(x_, W) + b)





Softmax

y = tf.nn.softmax(X)

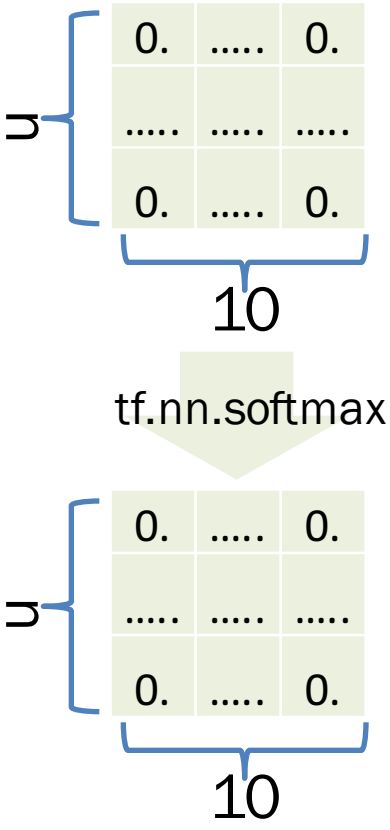
$$\mathbf{X} = \begin{bmatrix} X_1^{(1)} & X_2^{(1)} & X_3^{(1)} \\ X_1^{(2)} & X_2^{(2)} & X_3^{(2)} \\ X_1^{(3)} & X_2^{(3)} & X_3^{(3)} \\ X_1^{(4)} & X_2^{(4)} & X_3^{(4)} \end{bmatrix}$$

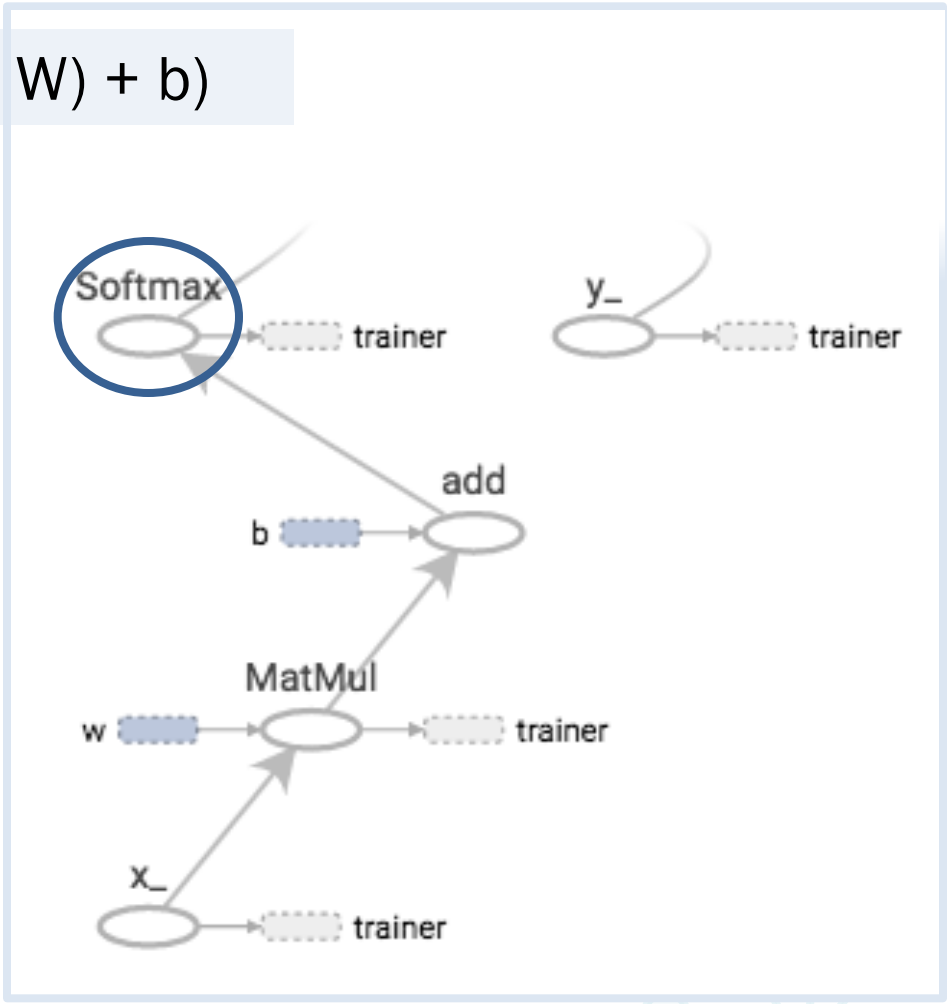
$$\mathbf{y} = \mathrm{softmax}(\mathbf{X})$$

$$= \begin{bmatrix} \frac{e^{X_1^{(1)}}}{e^{X_1^{(1)}}+e^{X_2^{(1)}}+e^{X_3^{(1)}}} & \frac{e^{X_2^{(1)}}}{e^{X_1^{(1)}}+e^{X_2^{(1)}}+e^{X_3^{(1)}}} & \frac{e^{X_3^{(1)}}}{e^{X_1^{(1)}}+e^{X_2^{(1)}}+e^{X_3^{(1)}}} \\ \frac{e^{X_1^{(2)}}}{e^{X_1^{(2)}}+e^{X_2^{(2)}}+e^{X_3^{(2)}}} & \frac{e^{X_2^{(2)}}}{e^{X_1^{(2)}}+e^{X_2^{(2)}}+e^{X_3^{(2)}}} & \frac{e^{X_3^{(2)}}}{e^{X_1^{(2)}}+e^{X_2^{(2)}}+e^{X_3^{(2)}}} \\ \frac{e^{X_1^{(3)}}}{e^{X_1^{(3)}}+e^{X_2^{(3)}}+e^{X_3^{(3)}}} & \frac{e^{X_2^{(3)}}}{e^{X_1^{(3)}}+e^{X_2^{(3)}}+e^{X_3^{(3)}}} & \frac{e^{X_3^{(3)}}}{e^{X_1^{(3)}}+e^{X_2^{(3)}}+e^{X_3^{(3)}}} \\ \frac{e^{X_1^{(4)}}}{e^{X_1^{(4)}}+e^{X_2^{(4)}}+e^{X_3^{(4)}}} & \frac{e^{X_2^{(4)}}}{e^{X_1^{(4)}}+e^{X_2^{(4)}}+e^{X_3^{(4)}}} & \frac{e^{X_3^{(4)}}}{e^{X_1^{(4)}}+e^{X_2^{(4)}}+e^{X_3^{(4)}}} \end{bmatrix}$$

Softmax

```
y = tf.nn.softmax(tf.matmul(x_, W) + b)
```





Error Function

```
cross_entropy = -tf.reduce_mean(y_* tf.log(y))
```

$$\mathbf{y\_} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \log(\mathbf{y}) = \begin{bmatrix} \log(y_1^{(1)}) & \log(y_2^{(1)}) & \log(y_3^{(1)}) \\ \log(y_1^{(2)}) & \log(y_2^{(2)}) & \log(y_3^{(2)}) \\ \log(y_1^{(3)}) & \log(y_2^{(3)}) & \log(y_3^{(3)}) \\ \log(y_1^{(4)}) & \log(y_2^{(4)}) & \log(y_3^{(4)}) \end{bmatrix}$$

$$\mathbf{y\_} * \log(\mathbf{y}) = \begin{bmatrix} \log(y_1^{(1)}) & 0 & 0 \\ 0 & \log(y_2^{(2)}) & 0 \\ \log(y_1^{(3)}) & 0 & 0 \\ 0 & 0 & \log(y_3^{(4)}) \end{bmatrix}$$

Error Function

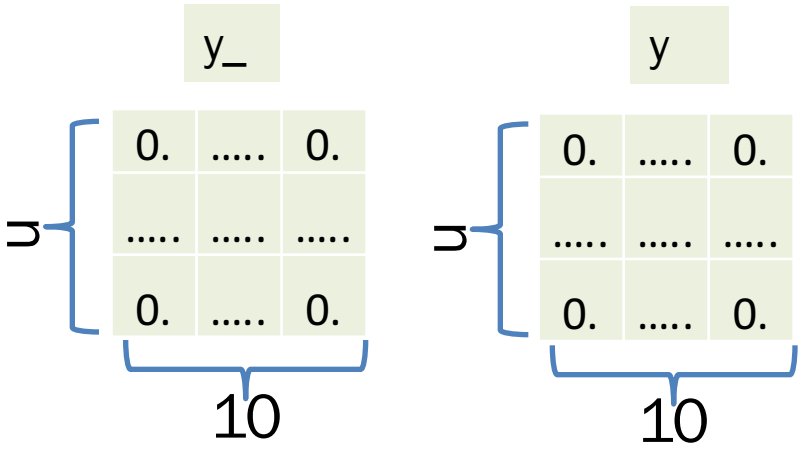
```
cross_entropy = -tf.reduce_mean(y_* tf.log(y))
```

$$\mathbf{y\_} * \log(\mathbf{y}) = \begin{bmatrix} \log(y_1^{(1)}) & 0 & 0 \\ 0 & \log(y_2^{(2)}) & 0 \\ \log(y_1^{(3)}) & 0 & 0 \\ 0 & 0 & \log(y_3^{(4)}) \end{bmatrix}$$

$$\text{reduce\_mean}(\mathbf{y\_} * \log(\mathbf{y})) = \frac{1}{4}\left(\log(y_1^{(1)}) + \log(y_2^{(2)}) + \log(y_1^{(3)}) + \log(y_3^{(4)})\right)$$

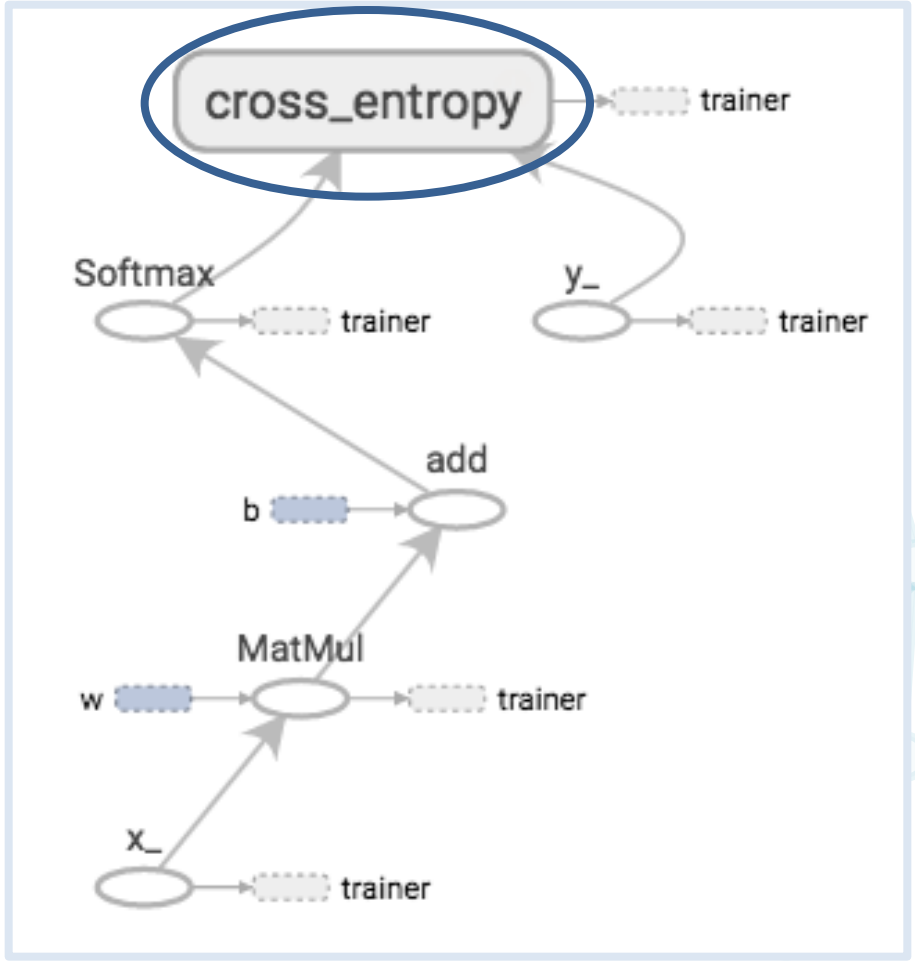
Error Function

```
cross_entropy = -tf.reduce_mean(y_* tf.log(y))
```



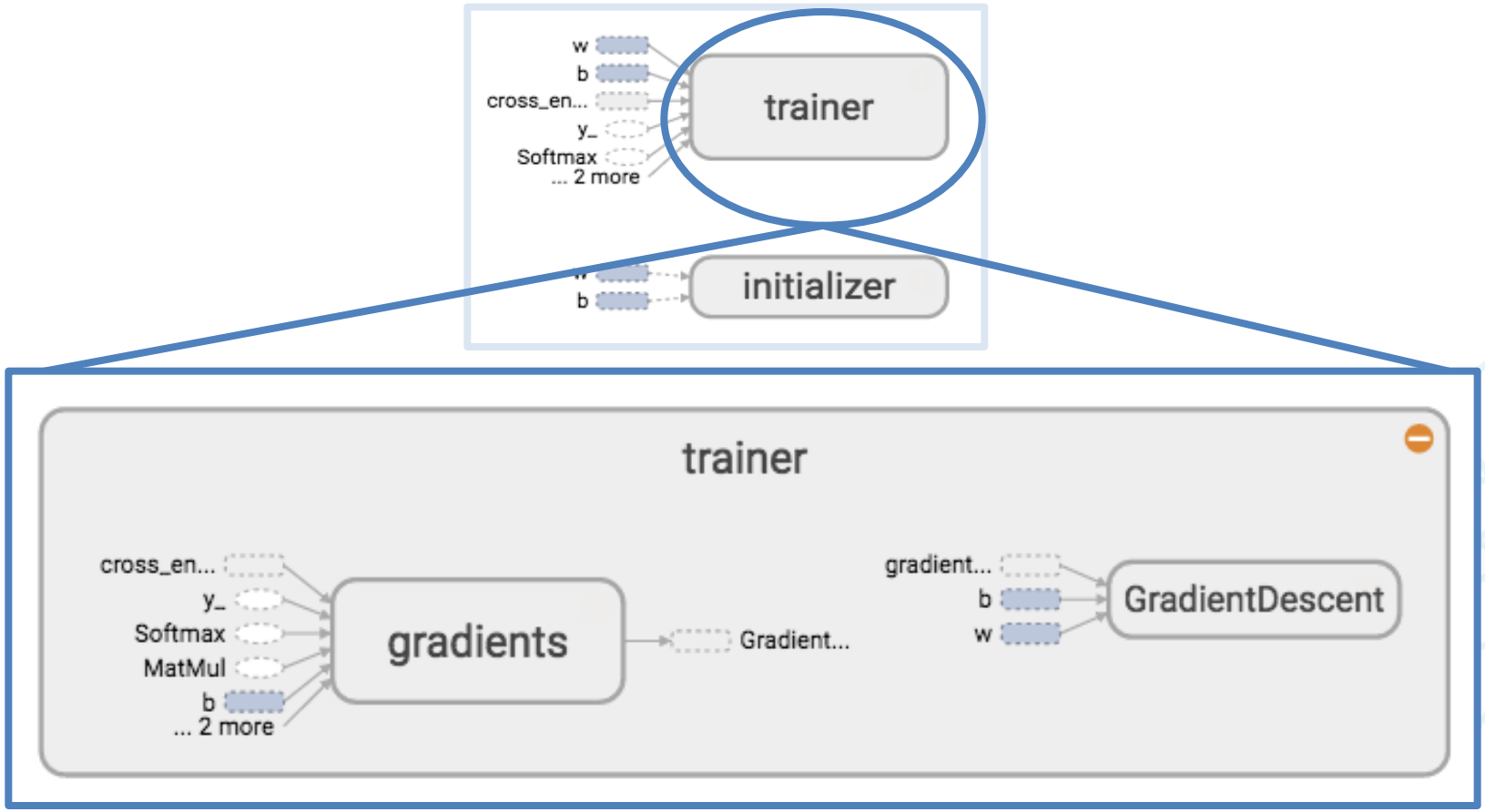
-tf.reduce_mean(y_*tf.log(y))

1.4331052

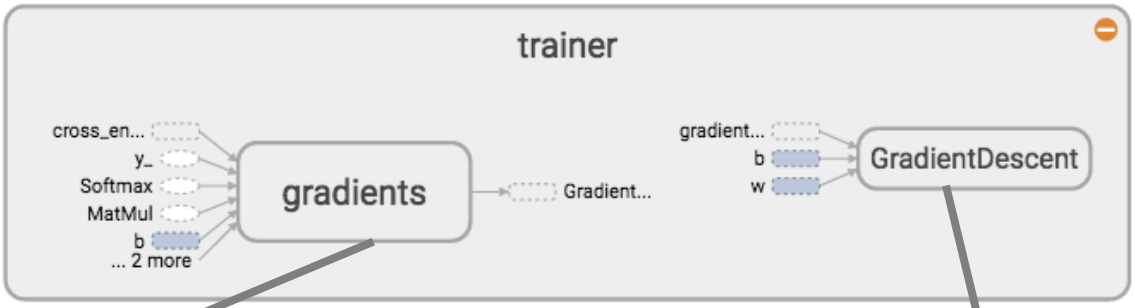


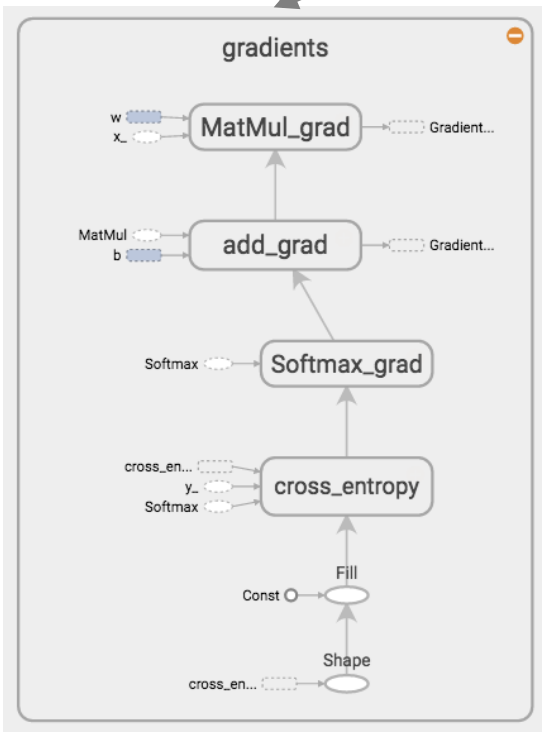
Trainer

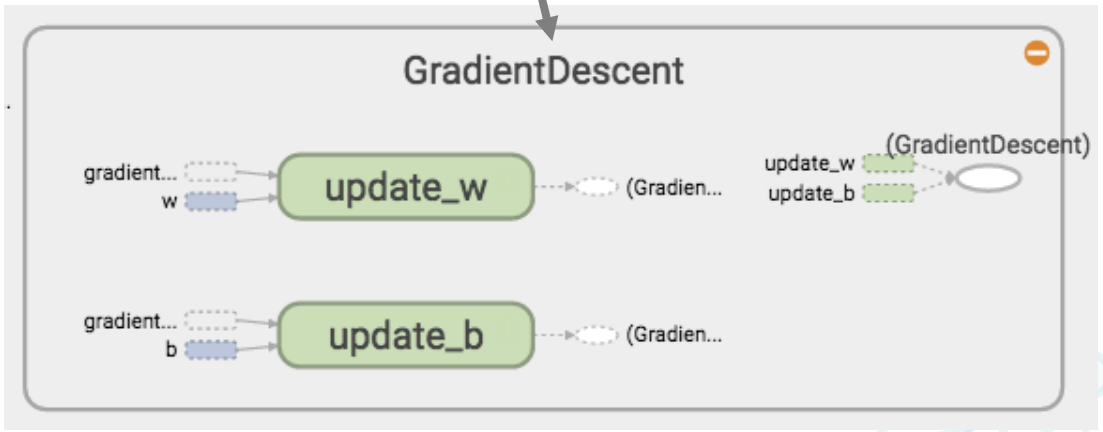
```
optimizer = tf.train.GradientDescentOptimizer(0.1)
train = optimizer.minimize(cross_entropy)
```



Trainer







$$w \leftarrow w - \eta \frac{\partial E(h(X), Y)}{\partial w}$$

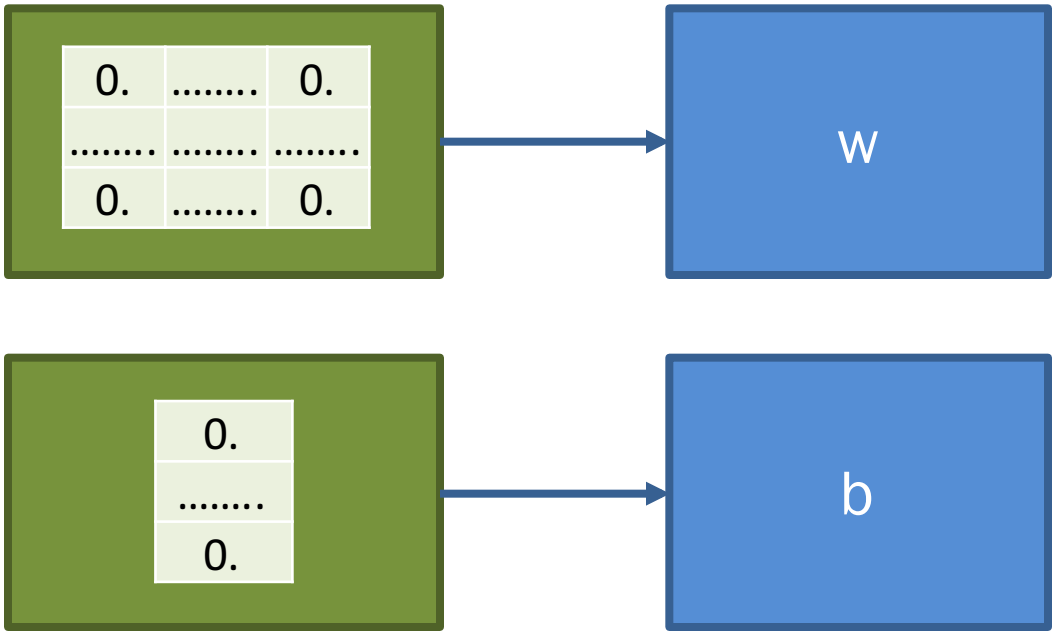
$$b \leftarrow b - \eta \frac{\partial E(h(X), Y)}{\partial b}$$

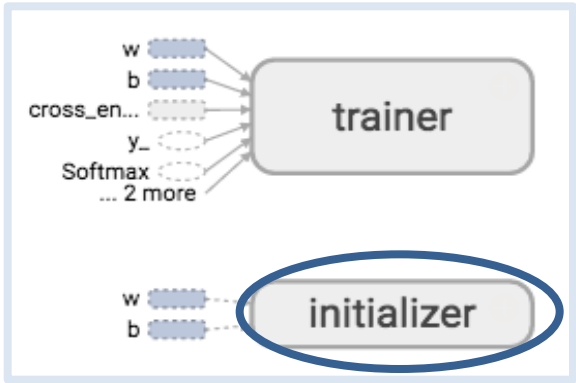
Computation Graph

- Initializer

```
init = tf.global_variables_initializer()
```

```
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
```





Session

```
# create session
sess = tf.Session()
# initialize variable
sess.run(init)
# gradient descent
for step in xrange(1000):
  sess.run(train, feed_dict={x_:x_data,y_:y_data})
  # fetch variable
  print sess.run(cross_entropy, feed_dict={x_:x_data,y_:y_data})
# release resource
sess.close()
```

Run Operations

```
sess.run(init)
```

the Node in Computational Graph

Run Operations

```
for step in xrange(1000):
    sess.run(train, feed_dict={x_:x_data,y_:y_data} )
```

the Node in Computational Graph

Input Data

x_data

0.	...	0.
0.	...	0.
0.	...	0.1
0.	...	0.
0.	...	0.2
0.	...	0.
...	...	...

y_data

0.	...	0.
0.	...	0.
0.	...	1.
1.	...	0.
0.	...	0.
0.	...	0.
...	...	...

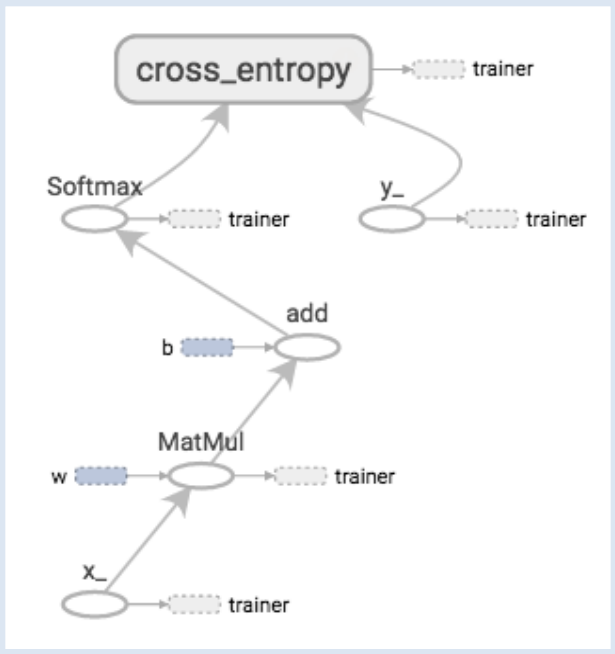
Run Operations

```
print sess.run(cross_entropy, feed_dict={x_:x_data,y_:y_data})
```

Results

2.4564333

the Node in Computational Graph



Input Data

x_data

0.	...	0.
0.	...	0.
0.	...	0.1
0.	...	0.
0.	...	0.2
0.	...	0.
...	...	...

y_data

0.	...	0.
0.	...	0.
0.	...	1.
1.	...	0.
0.	...	0.
0.	...	0.
...	...	...

Evaluation

```
# computational graph
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
# session
result_accurarcy = sess.run(accuracy, feed_dict={x_: mnist.test.images,
y_: mnist.test.labels})
```

Evaluation

```
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
```

$$\mathbf{y} = \begin{bmatrix} 0.8 & 0.2 & 0 \\ 0 & 1.0 & 0.0 \\ 0.4 & 0.5 & 0.1 \\ 0 & 0.1 & 0.9 \end{bmatrix} \qquad \mathbf{y\_} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\text{argmax}(\mathbf{y}) = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 2 \end{bmatrix} \qquad \text{argmax}(\mathbf{y\_}) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 2 \end{bmatrix}$$

$$\text{equal}(\text{argmax}(\mathbf{y}), \text{argmax}(\mathbf{y\_})) = \begin{bmatrix} True \\ True \\ False \\ True \end{bmatrix}$$

Evaluation

```
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

$$correct\_prediction = \begin{bmatrix} True \\ True \\ False \\ True \end{bmatrix}$$

$$\text{cast}(correct\_prediction, \text{float32}) = \begin{bmatrix} 1.0 \\ 1.0 \\ 0.0 \\ 1.0 \end{bmatrix}$$

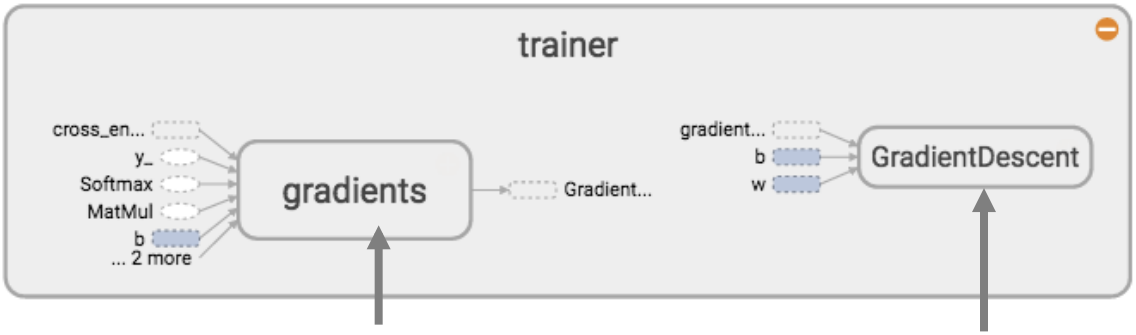
$$\text{reduce\_mean}(\text{cast}(correct\_prediction, \text{float32})) = 0.75$$

Decomposing Gradient Descent

- https://github.com/ckmarkoh/ntu_tensorflow/blob/master/one_layer_compute_gradients.ipynb

Decomposing Gradient Descent

```
grads = optimizer.compute_gradients(cross_entropy)
apply_grads = optimizer.apply_gradients(grads)
```



Compute Gradients

Apply Gradients

$$\frac{\partial E(h(X), Y)}{\partial w}$$

$$\frac{\partial E(h(X), Y)}{\partial b}$$

$$w \leftarrow w - \eta \frac{\partial E(h(X), Y)}{\partial w}$$

$$b \leftarrow b - \eta \frac{\partial E(h(X), Y)}{\partial b}$$

Save and Load Trained Model

- https://github.com/ckmarkoh/ntu_tensorflow/blob/master/one_layer_load.ipynb

Save and Load Trained Model

- Save Trained Model

```
saver = tf.train.Saver()
saver.save(sess, "model.ckpt")
```

- Load Trained Model

```
saver = tf.train.Saver()
saver.restore(sess, "model.ckpt")
```

Save and Load Trained Model

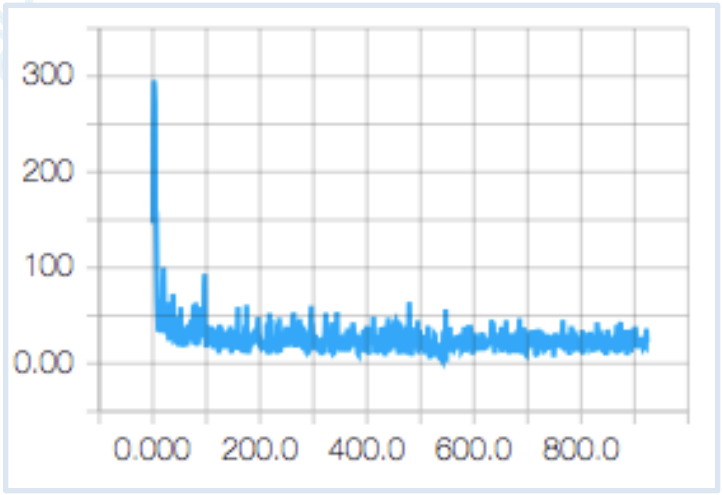
- .ckpt.data : values of all variables
- .ckpt.meta : structure of the computational graph
- How to save and load computational graph?
  - https://www.tensorflow.org/versions/r0.12/how_tos/meta_graph/

Tensorboard

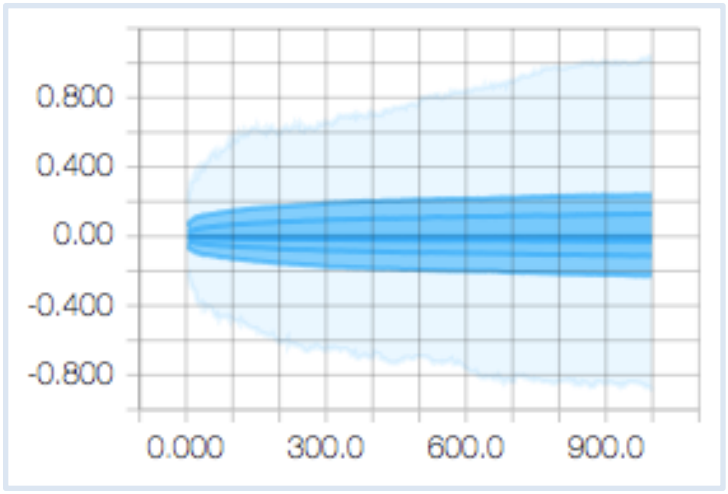
- https://github.com/ckmarkoh/ntu_tensorflow/blob/master/one_layer_board.ipynb

Tensorboard

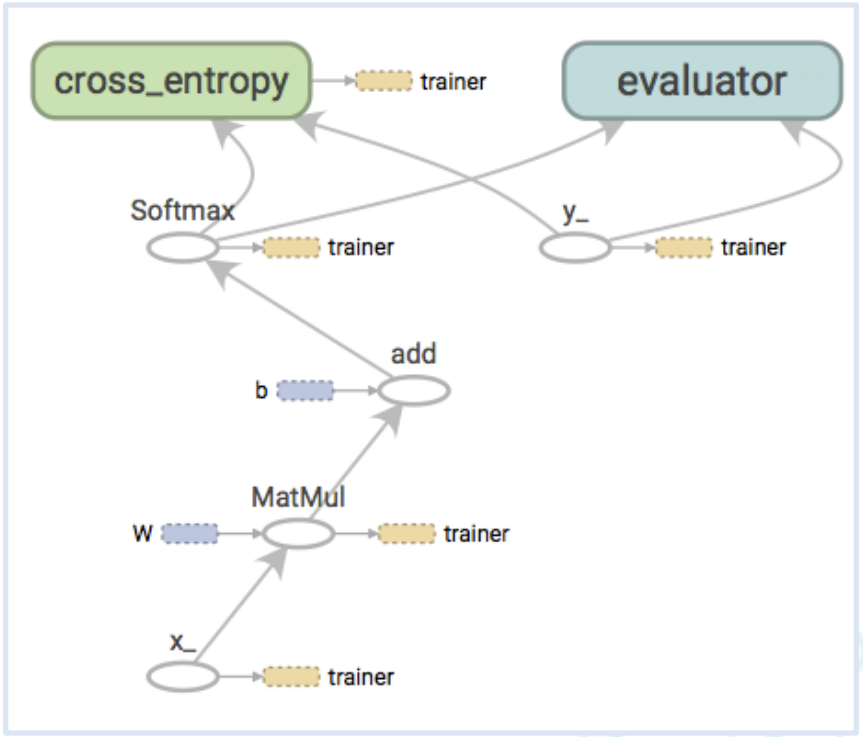
Scalar Summary



Histogram Summary

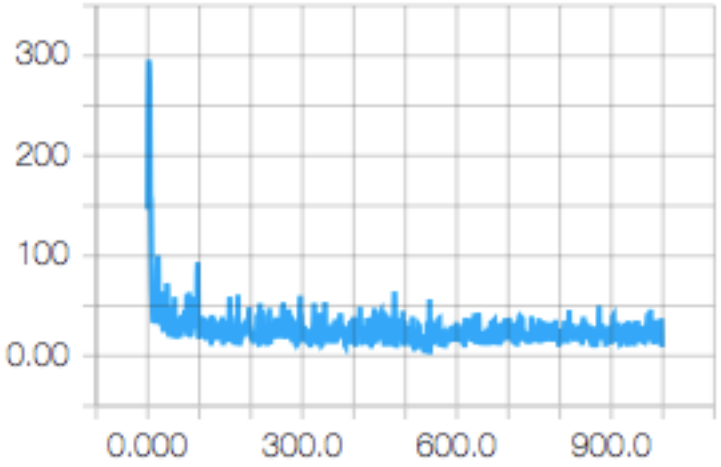


Computational Graph

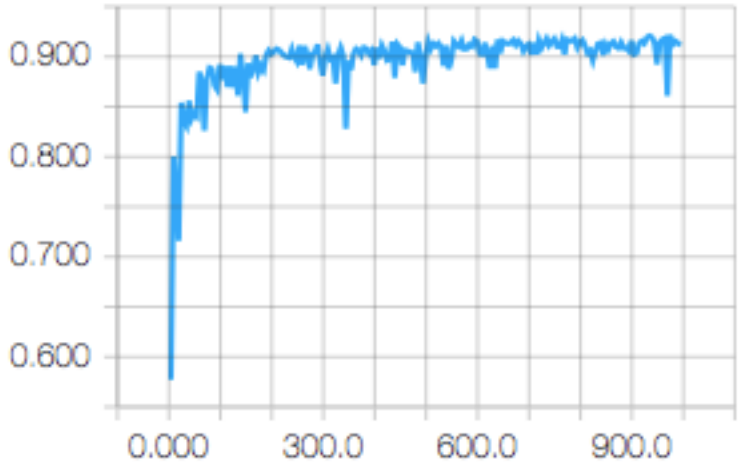


Scalar Summary

cross_entropy



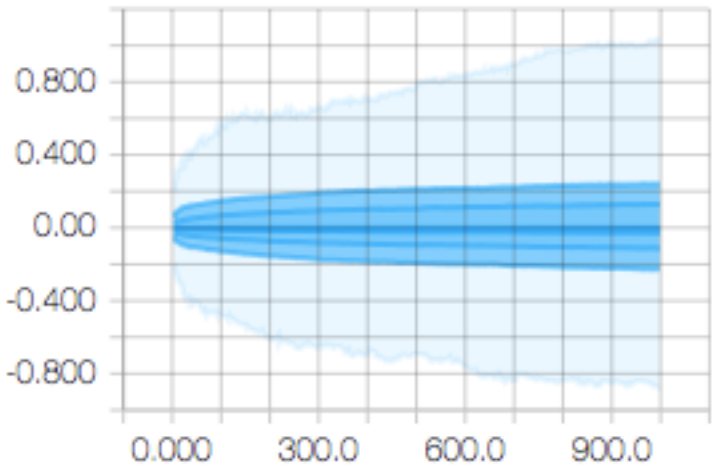
accurarcy



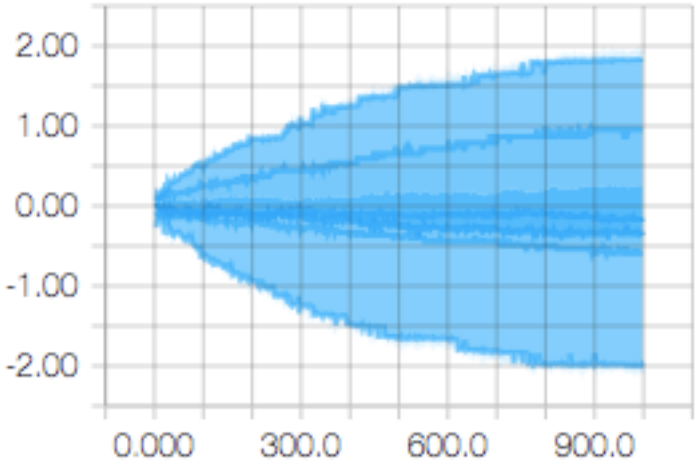

```
summ_ce = tf.summary.scalar("cross_entropy", cross_entropy)
summ_acc = tf.summary.scalar("accuracy", accuracy)
```

Histogram Summary

weights



biases



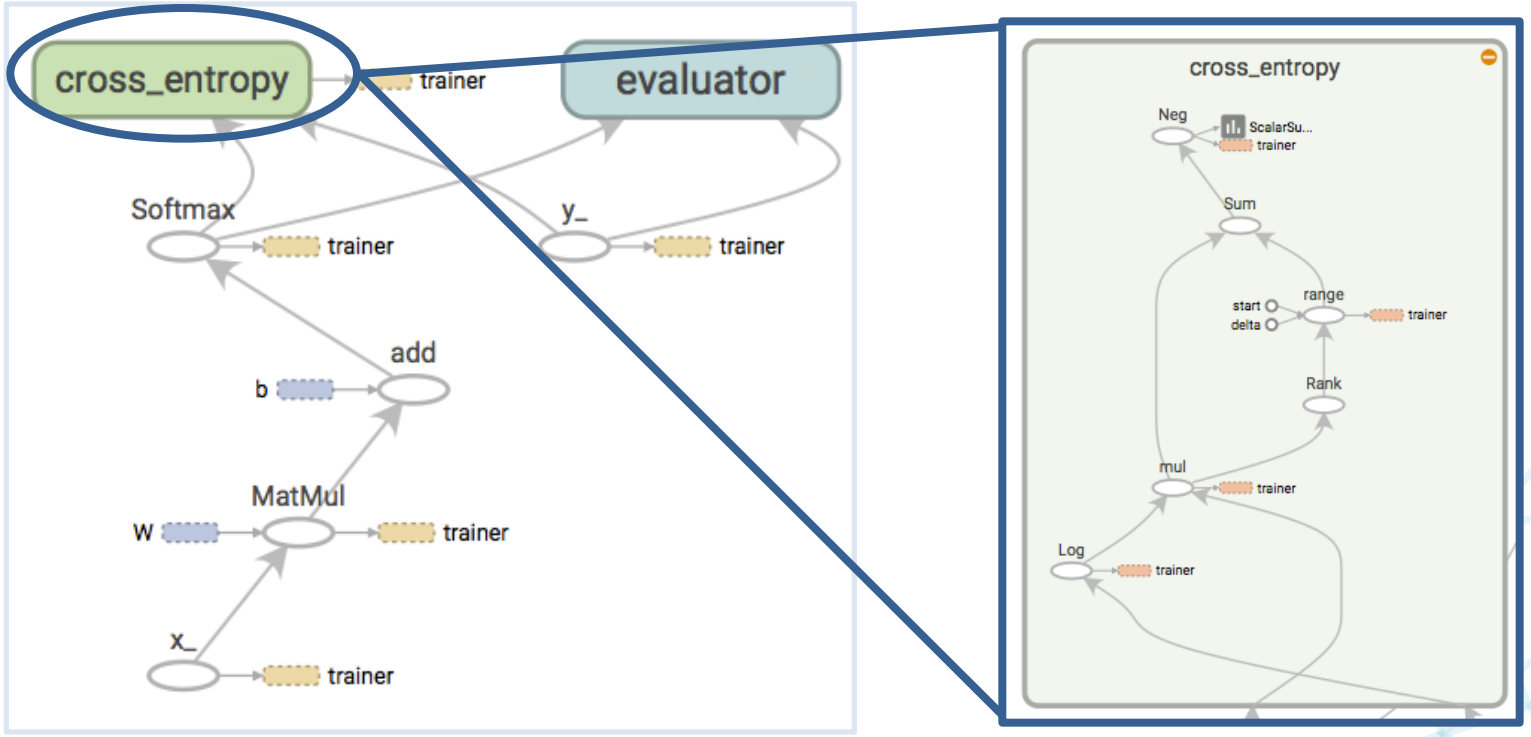

```
summ_W = tf.summary.histogram("weights", W)
summ_b = tf.summary.histogram("biases", b)
```

Summary Writer

```
summ_merged = tf.summary.merge([summ_W, summ_b, summ_ce])

writer = tf.summary.FileWriter("./", sess.graph_def)
for i in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    result1 = sess.run(tf.log(y),  feed_dict={x_: batch_xs, y_: batch_ys})
    result2 = sess.run(y_ * tf.log(y),  feed_dict={x_: batch_xs, y_: batch_ys})
    sess.run(trainer,  feed_dict={x_: batch_xs, y_: batch_ys})
    summ_str = sess.run(summ_merged,feed_dict={x_:batch_xs,y_:batch_ys})
    writer.add_summary(summ_str, i)
    if (i+1)%5 == 0:
        summary_str = sess.run(summ_acc,feed_dict={x_:mnist.test.images,
            y_:mnist.test.labels})
        writer.add_summary(summ_str, i)
```

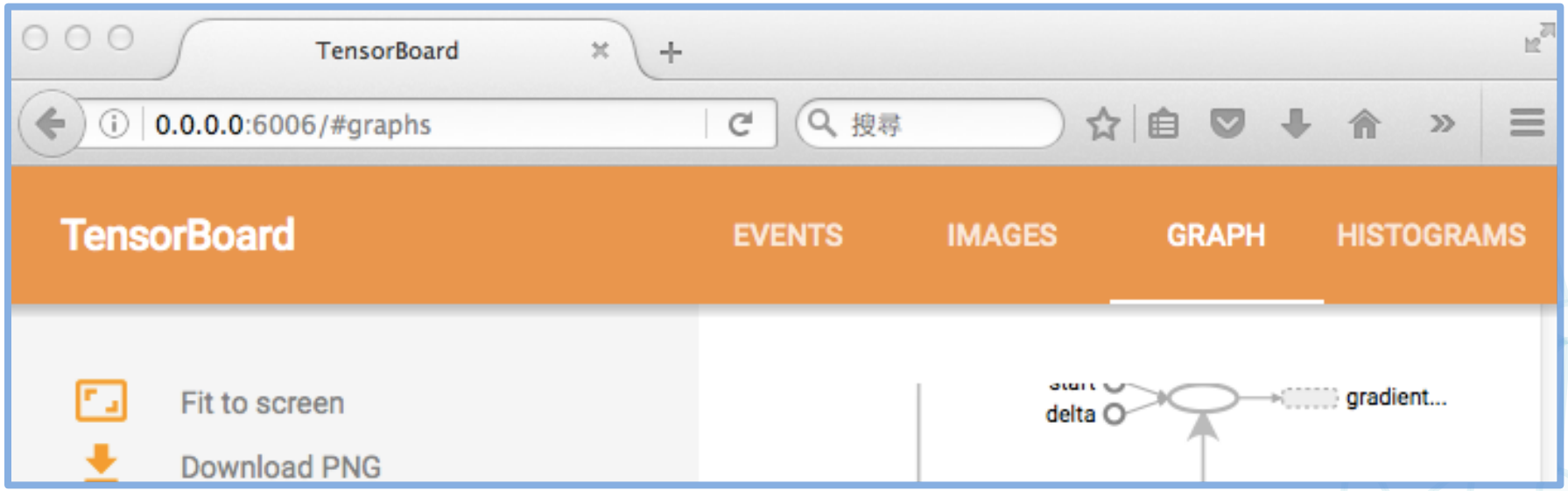
name_scope

```
with tf.name_scope("cross_entropy") as scope:
    cross_entropy = -tf.reduce_mean(y_ * tf.log(y))
```



Launch Tensorboard

```
> tensorboard --logdir=./
Starting TensorBoard  on port 6006
(You can navigate to http://0.0.0.0:6006)
```



Viewing Computational Graph without Tensorboard

- https://github.com/ckmarkoh/ntu_tensorflow/blob/master/computational_graph_py.ipynb

Viewing Computational Graph without Tensorboard

- tf.get_default_graph()
  - get the computational graph
- tf.get_default_graph().get_operations()
  - get all nodes in computational graph
- tf.get_default_graph().get_tensor_by_name ("name:0")
  - get the node by name
- tf.global_variables()
  - get all variables
  - What is a local variable in tensorflow?
    - https://stackoverflow.com/questions/38910198/what-is-a-local-variable-in-tensorflow

Convolutional Neural Networks

- https://github.com/ckmarkoh/ntu_tensorflow/blob/master/convnet_train.ipynb

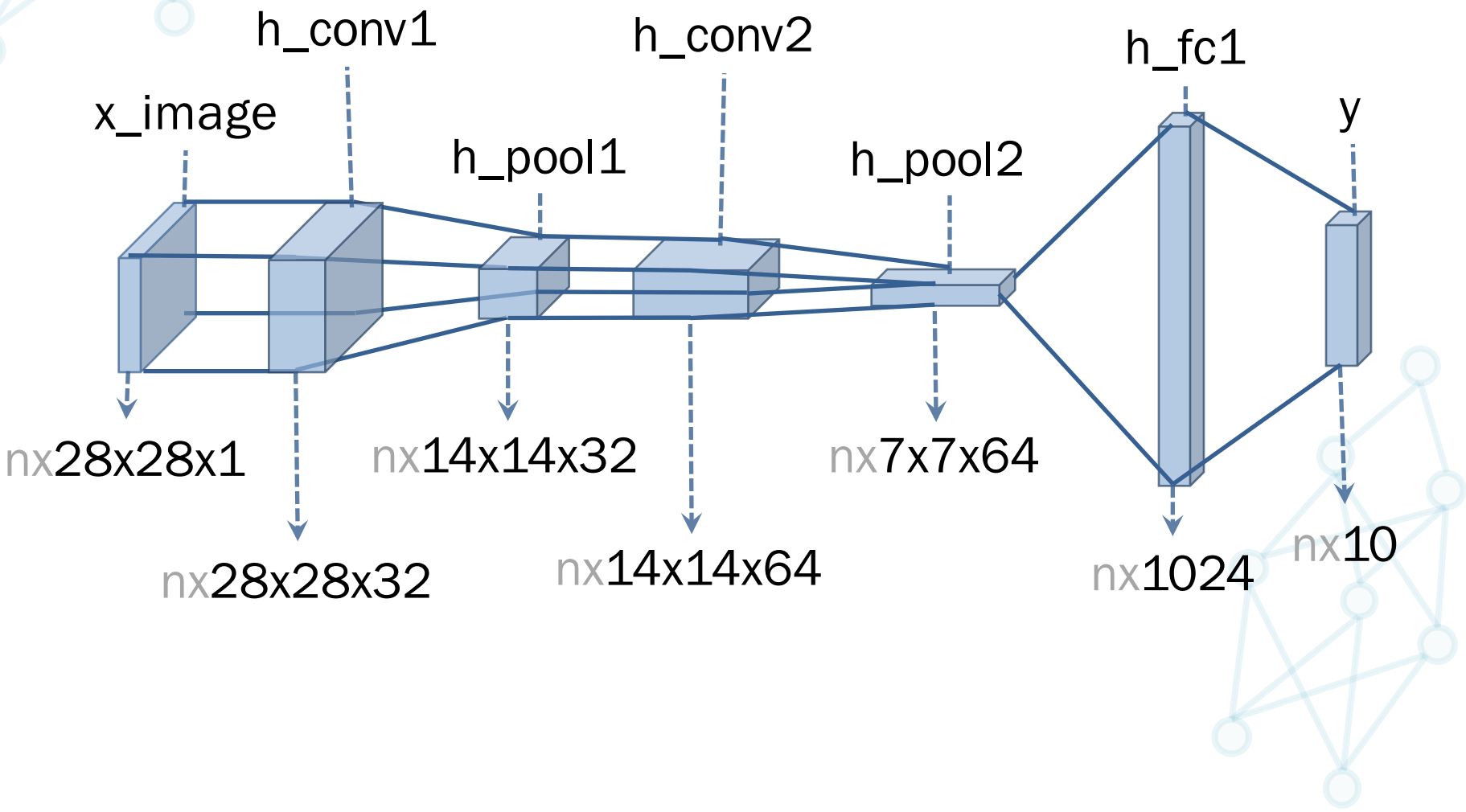
Create Variables & Operators

```
def weight_variable(shape):
    return tf.Variable(tf.truncated_normal(shape, stddev=0.1))

def bias_variable(shape):
    return tf.Variable(tf.constant(0, shape=shape))

def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
        strides=[1, 2, 2, 1], padding='SAME')
```

Computational Graph

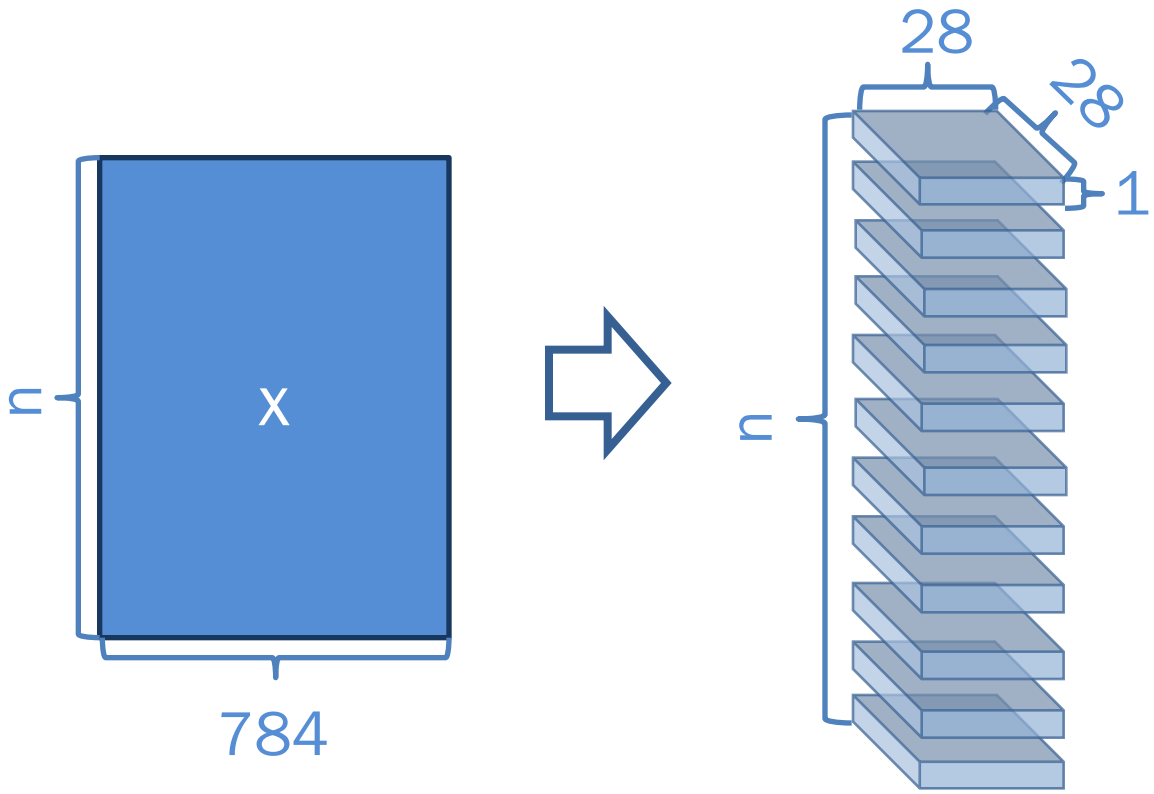
```
x_ = tf.placeholder(tf.float32, [None, 784], name="x_")
y_ = tf.placeholder(tf.float32, [None, 10], name="y_”)
x_image = tf.reshape(x_, [-1,28,28,1])
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)
W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])
h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)
W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])
h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])
y= tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)
```

Convolutional Neural Networks



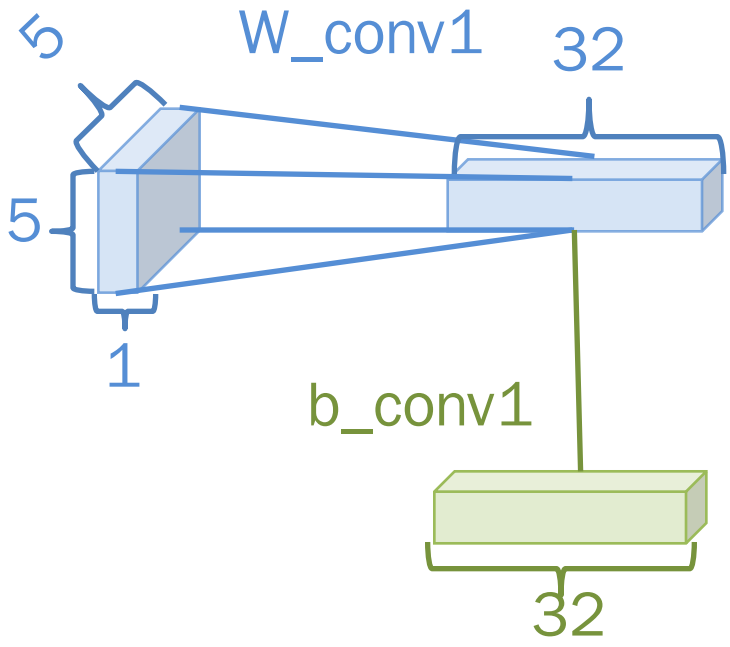
Reshape

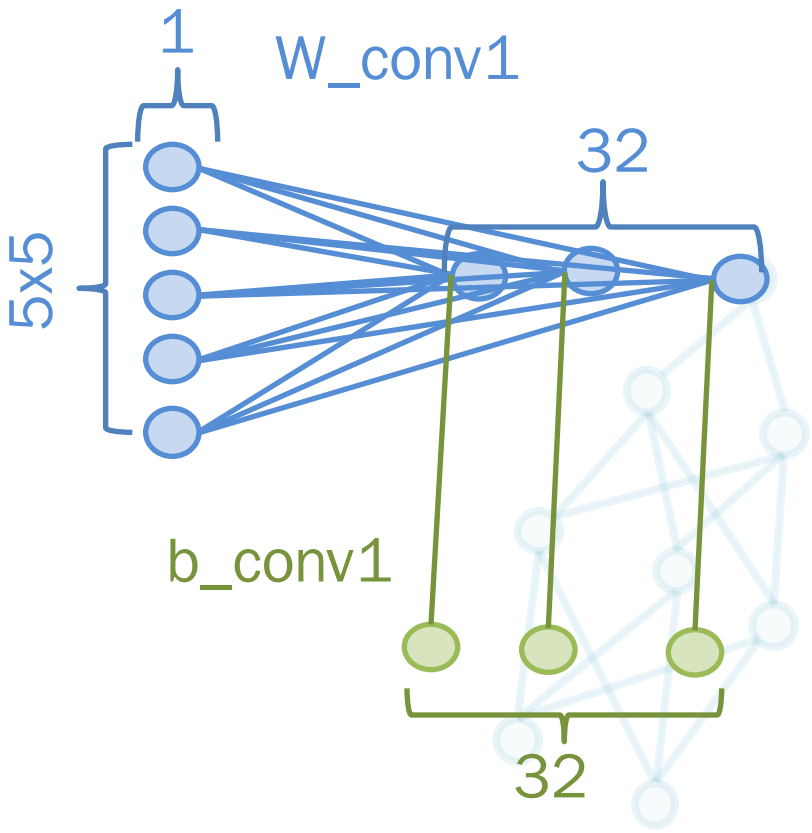
```
x_image = tf.reshape(x_, [-1,28,28,1])
```



Convolutional Layer

```
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])
```

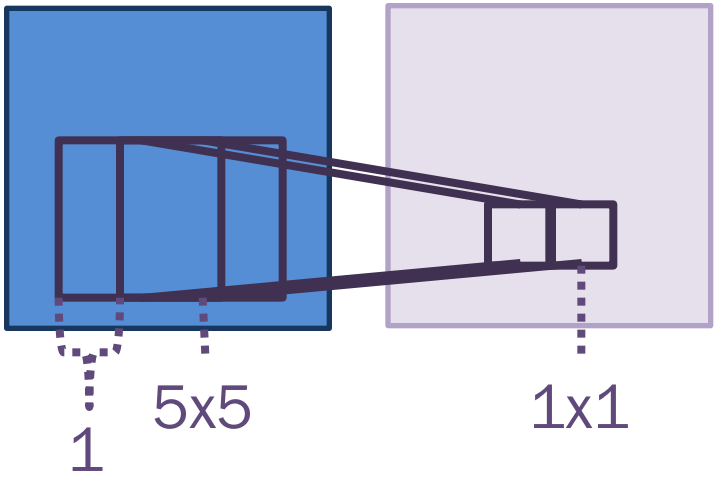




Convolutional Layer

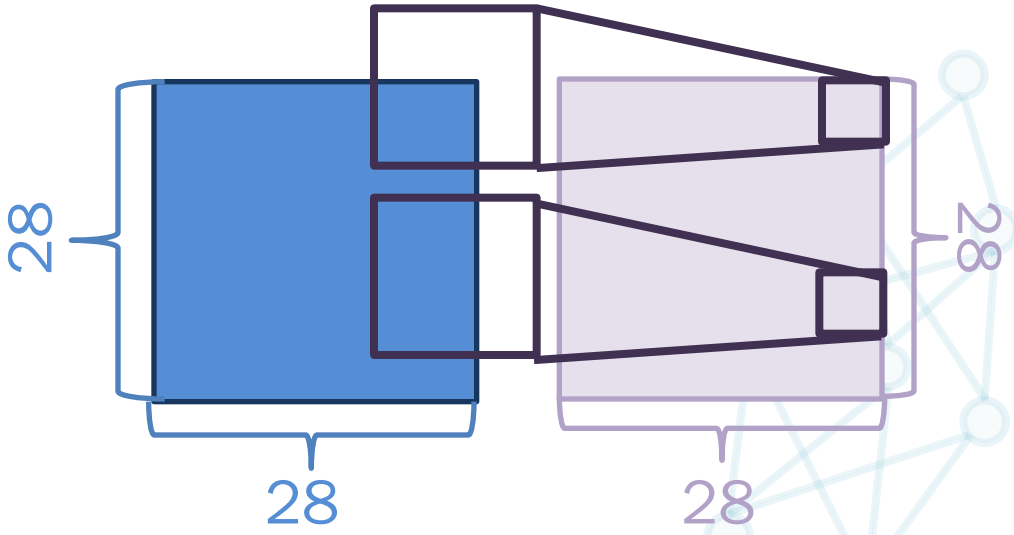
```
tf.nn.conv2d(x, W , strides=[1, 1, 1, 1], padding='SAME')+b
```

[batch, in_height, in_width, in_channels]



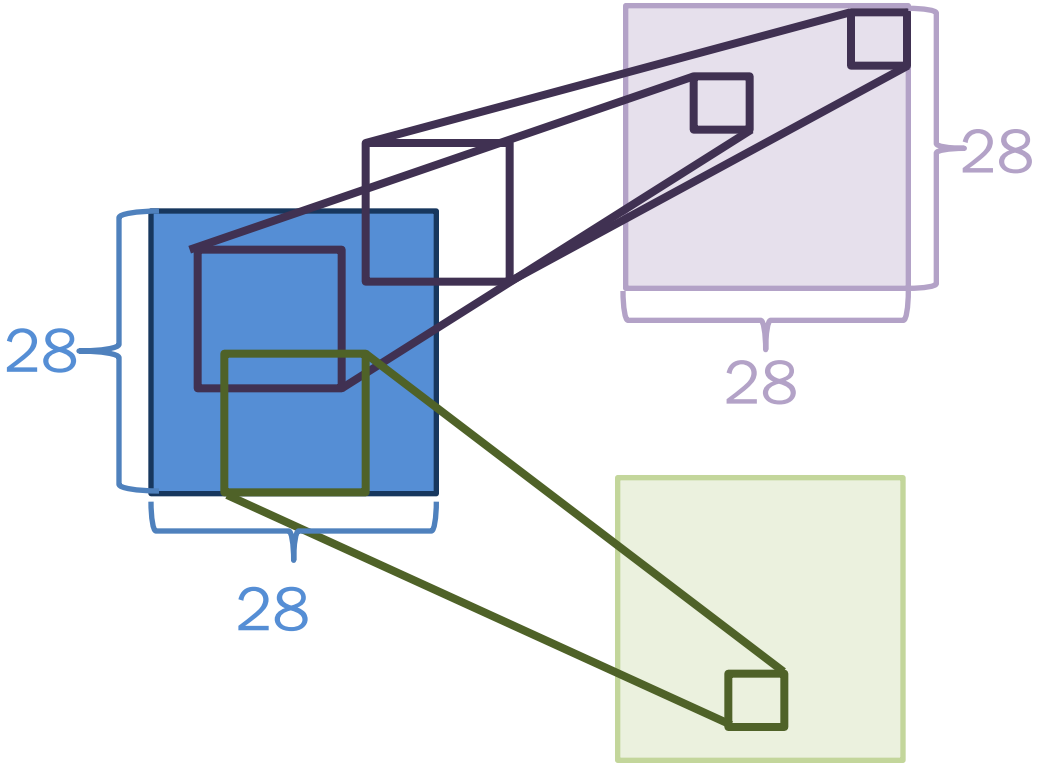
strides=1

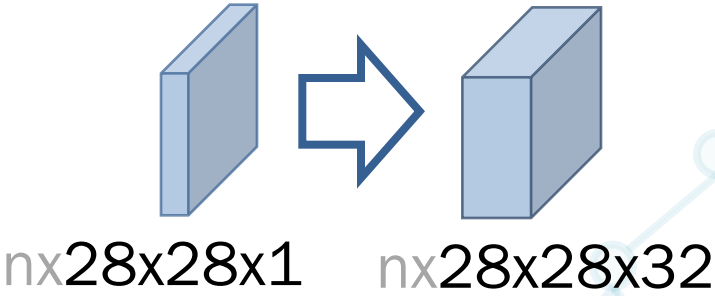
padding='SAME'



Convolutional Layer

```
tf.nn.conv2d(x, W , strides=[1, 1, 1, 1], padding='SAME')+b
```

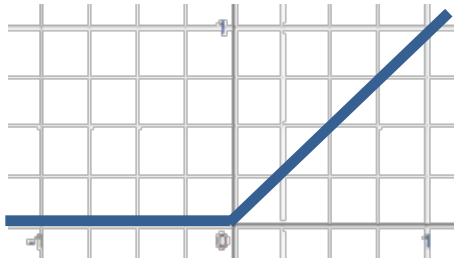




ReLU

```
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
```

ReLU: $\begin{cases} n_{in} \text{ if } n_{in} > 0 \\ 0 \text{ otherwise} \end{cases}$

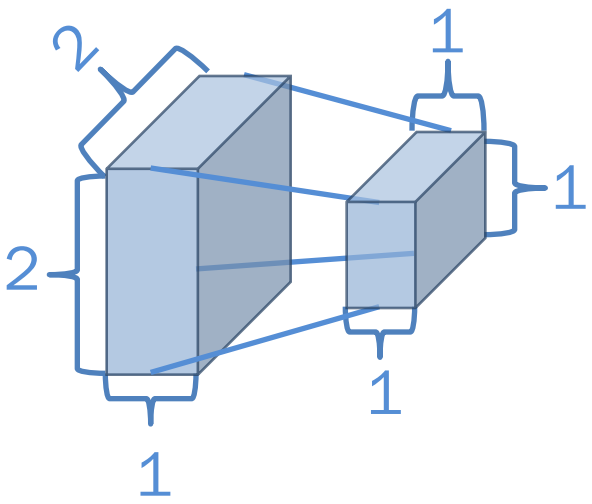


-0.5	0.2	0.3	-0.1
0.2	-0.3	-0.4	-1.1
2.1	-2.1	0.1	1.2
0.2	3.0	-0.3	0.5

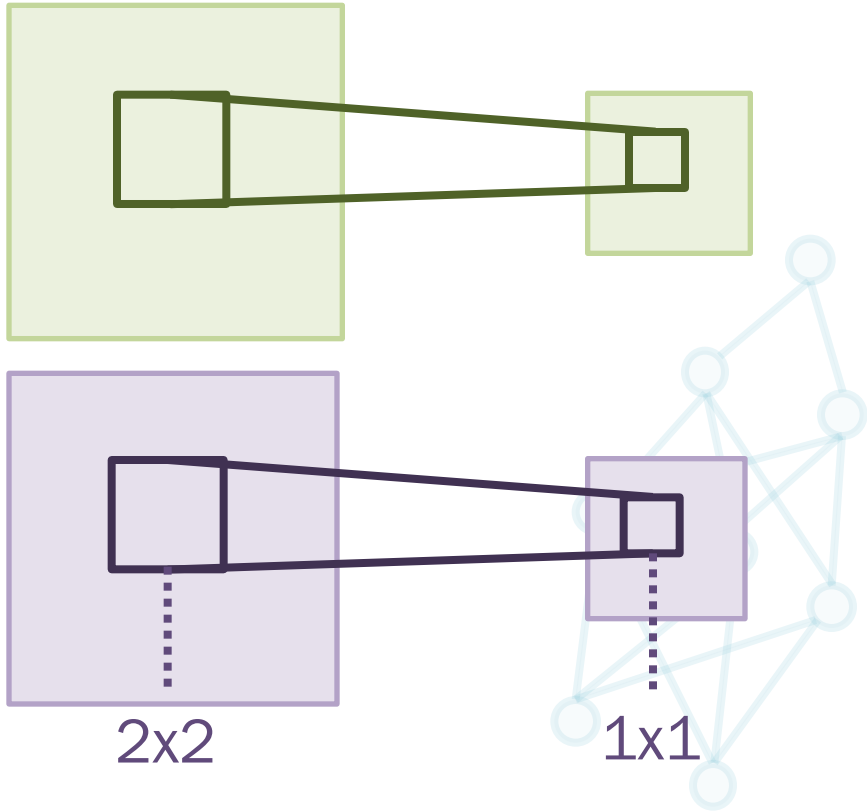
0	0.2	0.3	0
0.2	0	0	0
2.1	0	0.1	1.2
0.2	3.0	0	0.5

Pooling Layer

```
tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                  strides=[1, 2, 2, 1], padding='SAME')
```

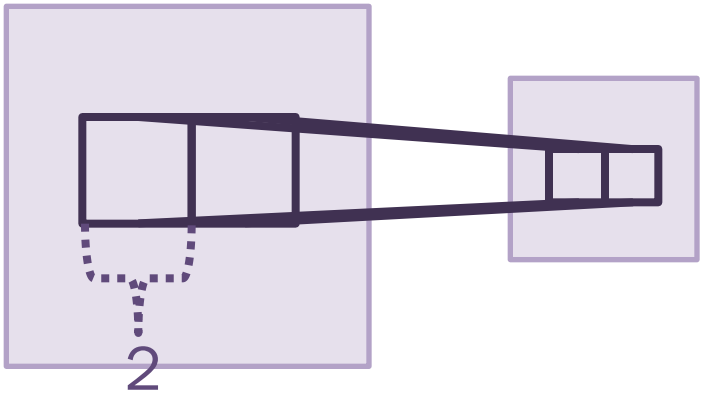


1x2x2x1



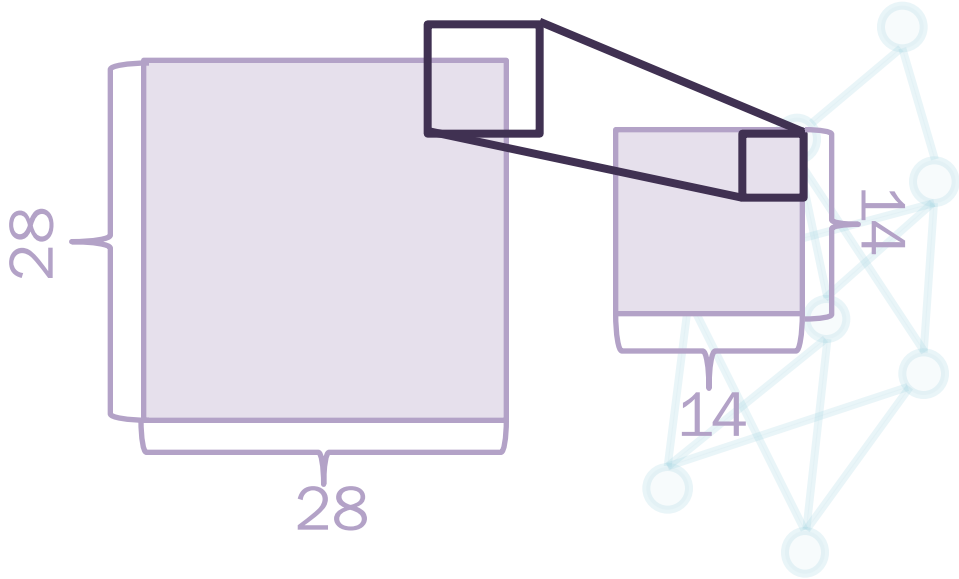
Pooling Layer

```
tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                  strides=[1, 2, 2, 1], padding='SAME')
```



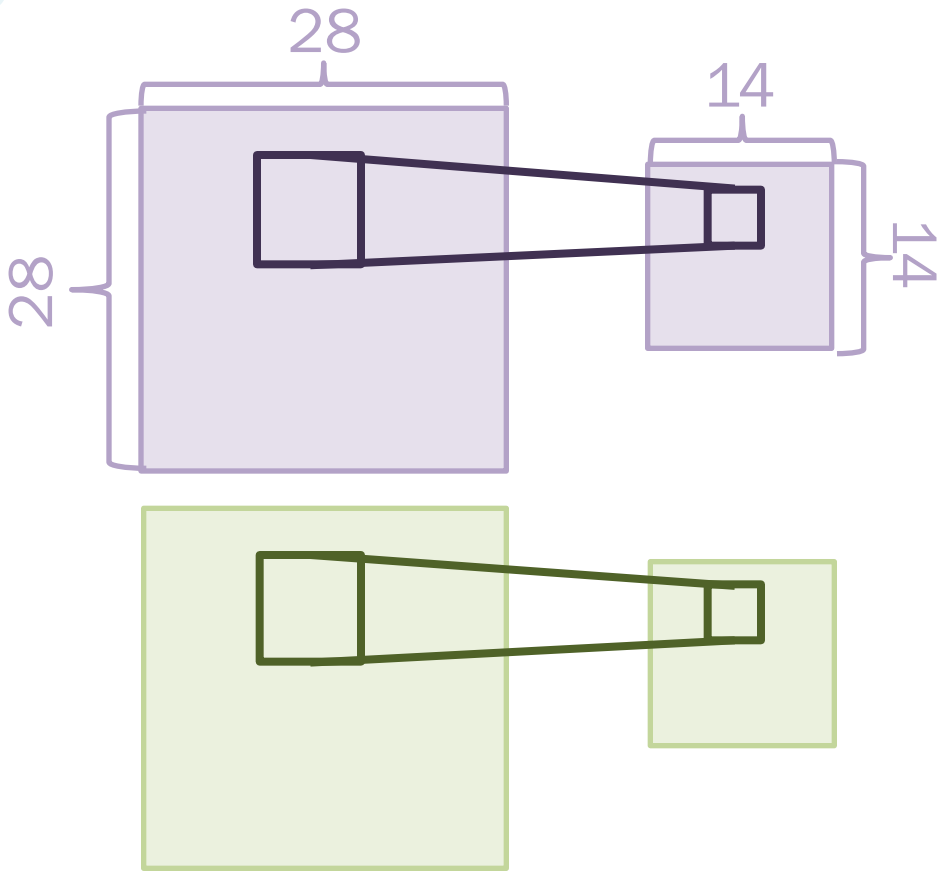
strides=2

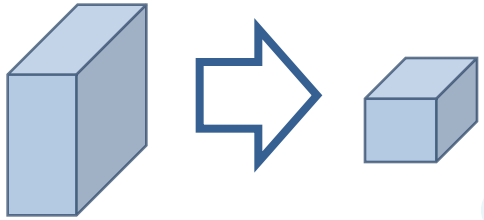
padding='SAME'



Pooling Layer

```
h_pool1 = max_pool_2x2(h_conv1)
```

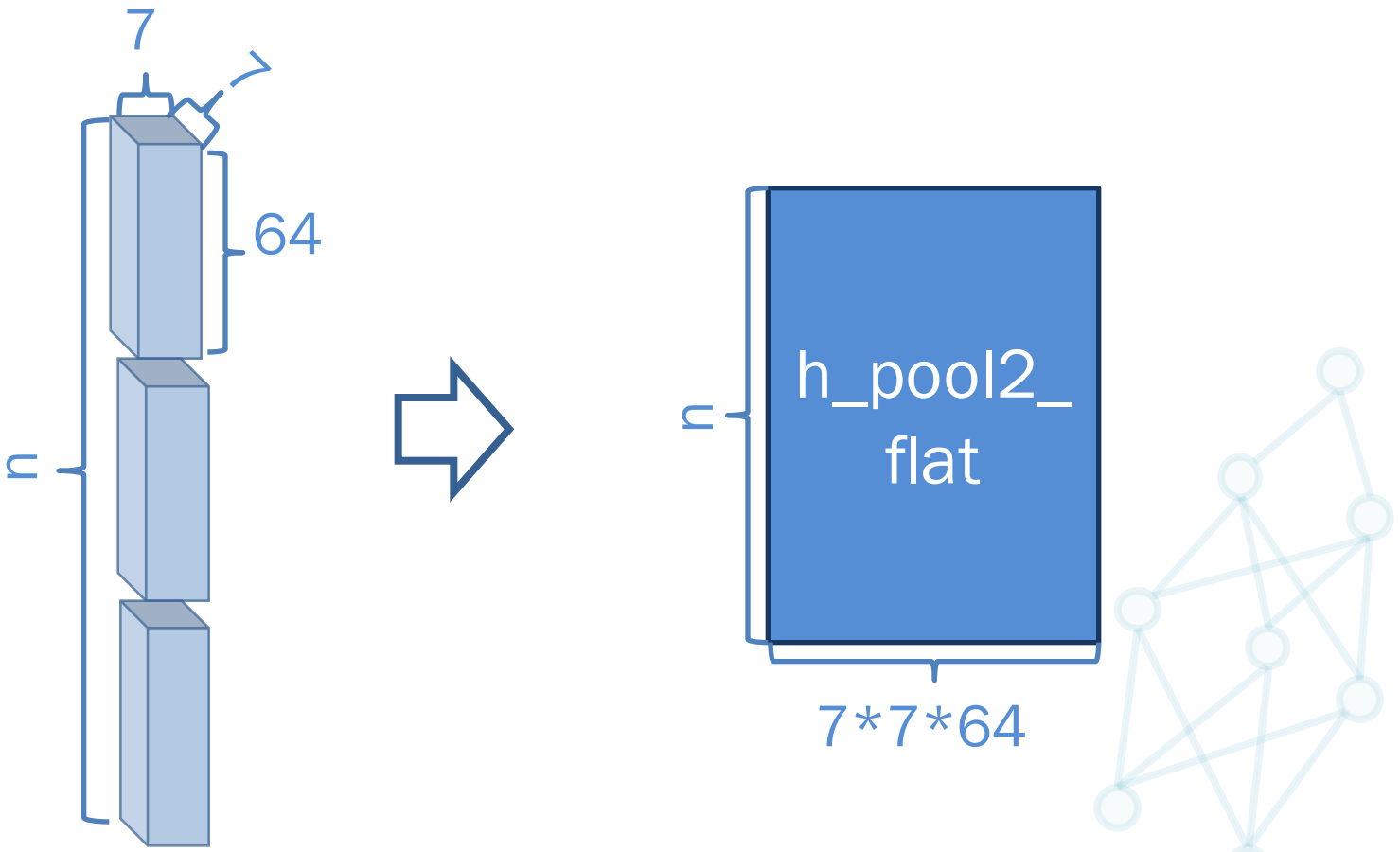




nx28x28x32 nx14x14x32

Reshape

```
h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
```

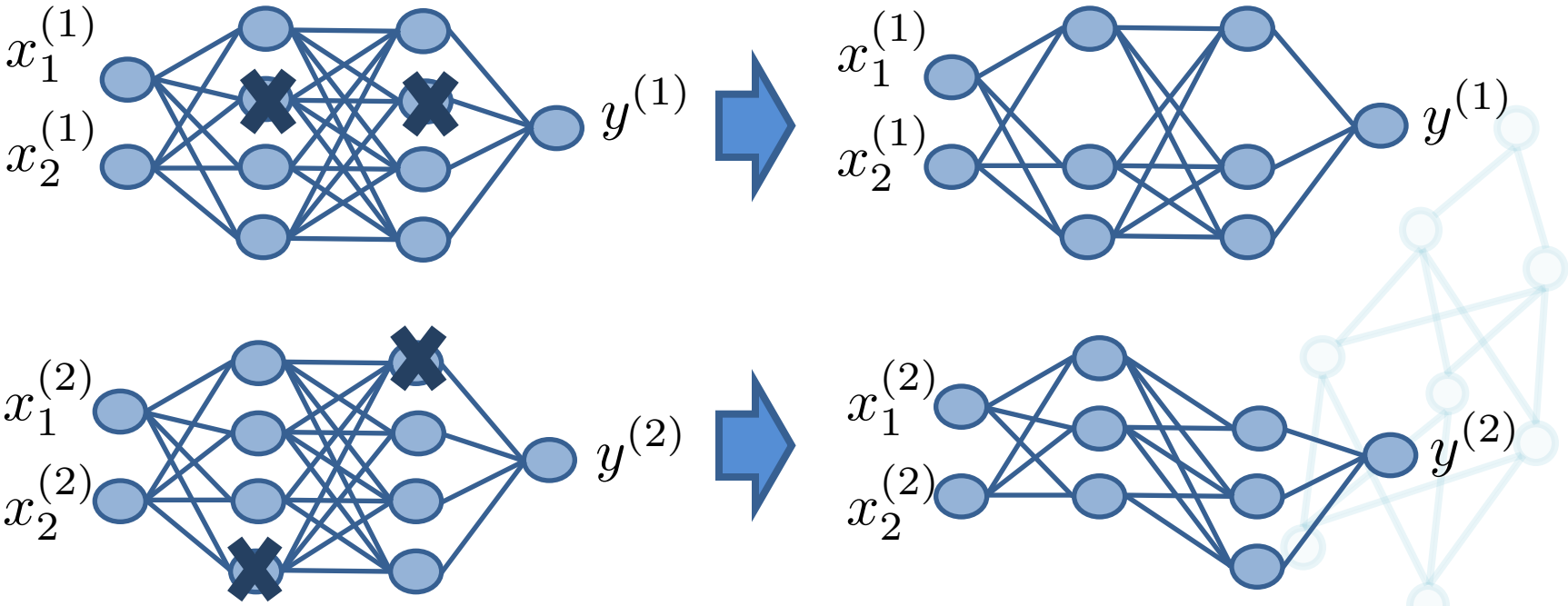


Preventing Overfitting

- Dropout
- Early Stop

Dropout

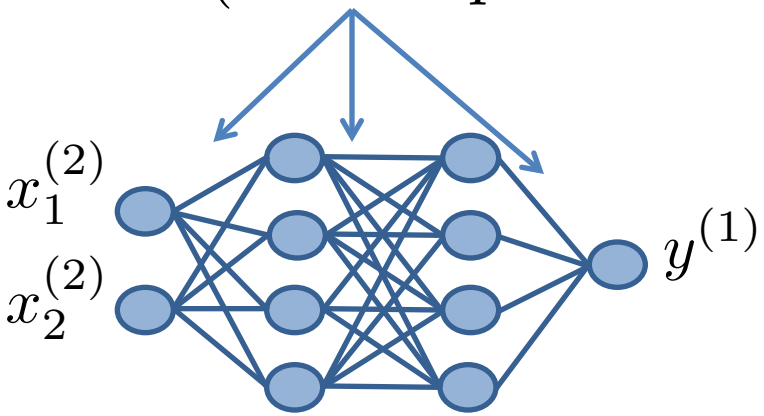
- Randomly remove neurons in hidden layers
- ex: 25%的Dropout Rate



Dropout

- During testing, all weights should be multiply by **(1 – dropout_rate)**

$$w \leftarrow w(1 - dropout\_rate)$$

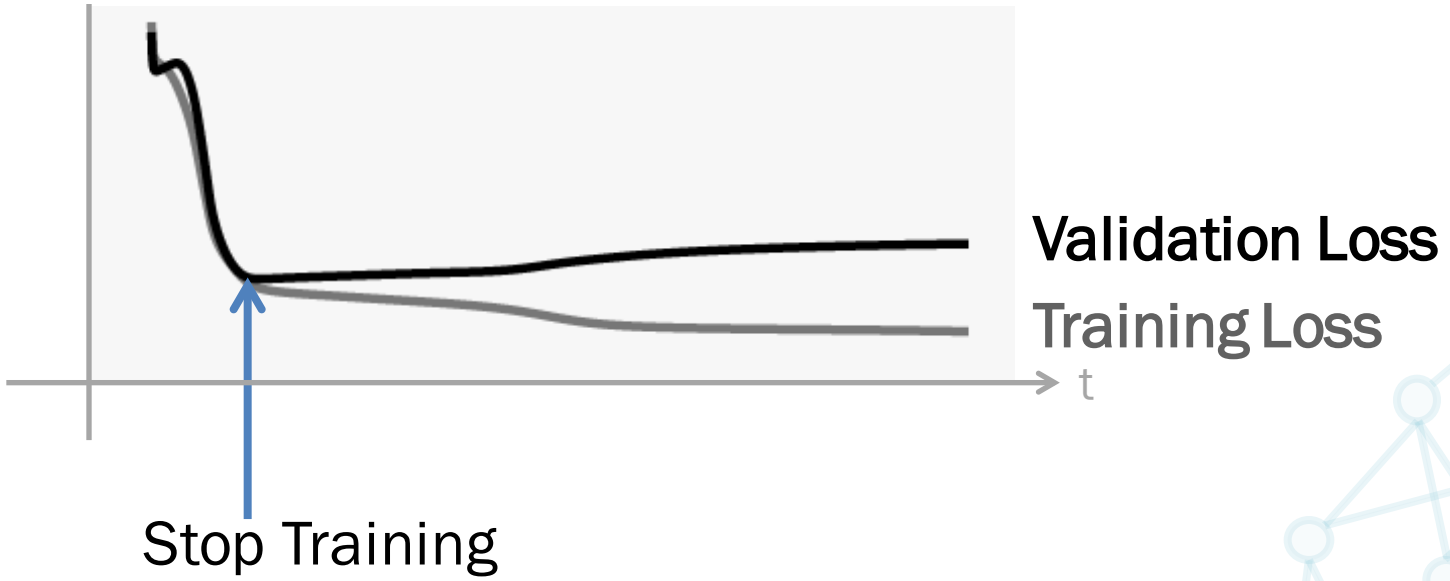


Dropout

- keep_prob = 1- dropout_rate

```
W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])
h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])
y= tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)
```

Early Stop



Early Stop

```
patience = 20
best_accurarcy = 0
i = 0
while True:
    i += 1
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(trainer,  feed_dict={x_: batch_xs, y_: batch_ys, keep_prob:0.5})
    if i%100== 0:
        valid_accurarcy = sess.run(accuracy,
            feed_dict={x_: mnist.validation.images, y_: mnist.validation.labels, keep_prob:1})
        print "%s, valid_accurarcy:%s" %(I, valid_accurarcy)
        if valid_accurarcy > best_accurarcy:
            patience = 20
            best_accurarcy = valid_accurarcy
            print "save model"
            saver.save(sess, "model_conv.ckpt")
        else:
            patience -= 1
            if patience == 0:
                print "early stop"
                break
```

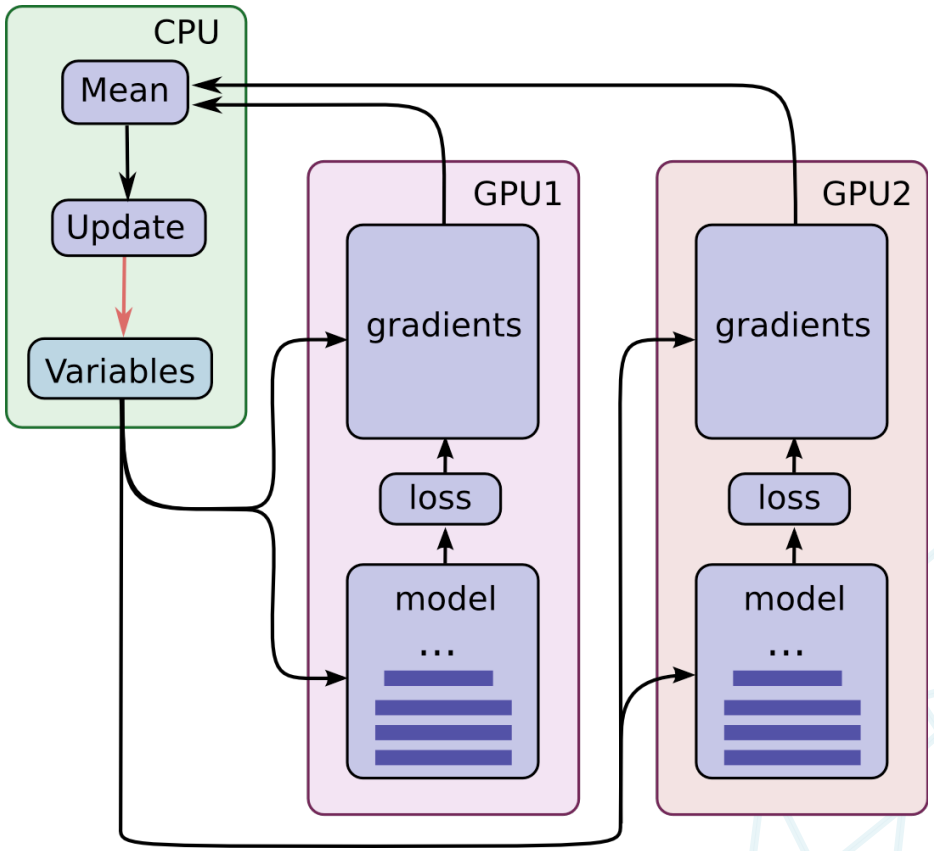
Multi-GPU Implementation

- https://github.com/ckmarkoh/ntu_tensorflow/blob/master/multi_gpu.py

Multi-GPU Implementation

- Variables:
  - stored in CPU
  - shared by GPUs
  - updated by CPU
- Gradients:
  - computed by GPUs
  - averaged by CPU

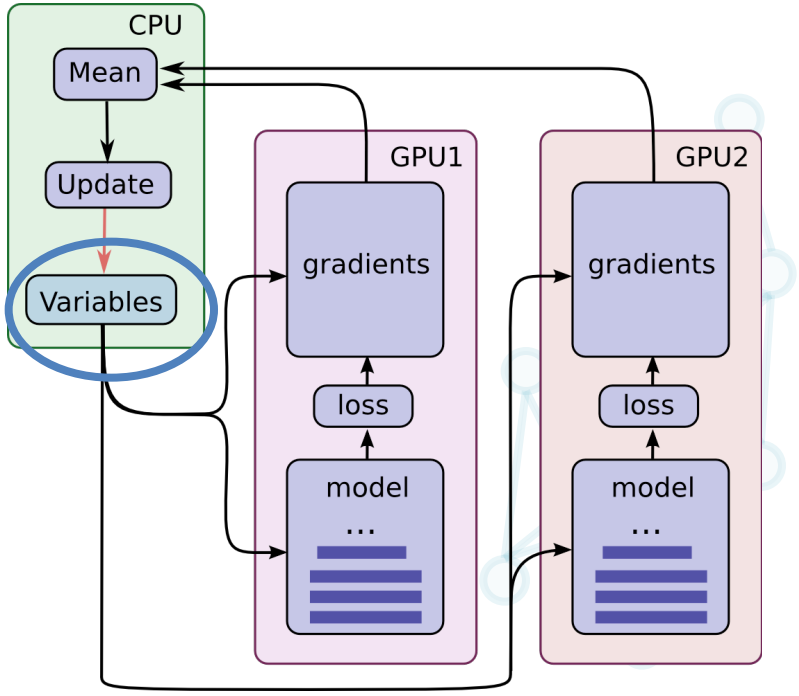
https://www.tensorflow.org/images/Parallelism.png



Multi-GPU Implementation

- create variables stored in CPU

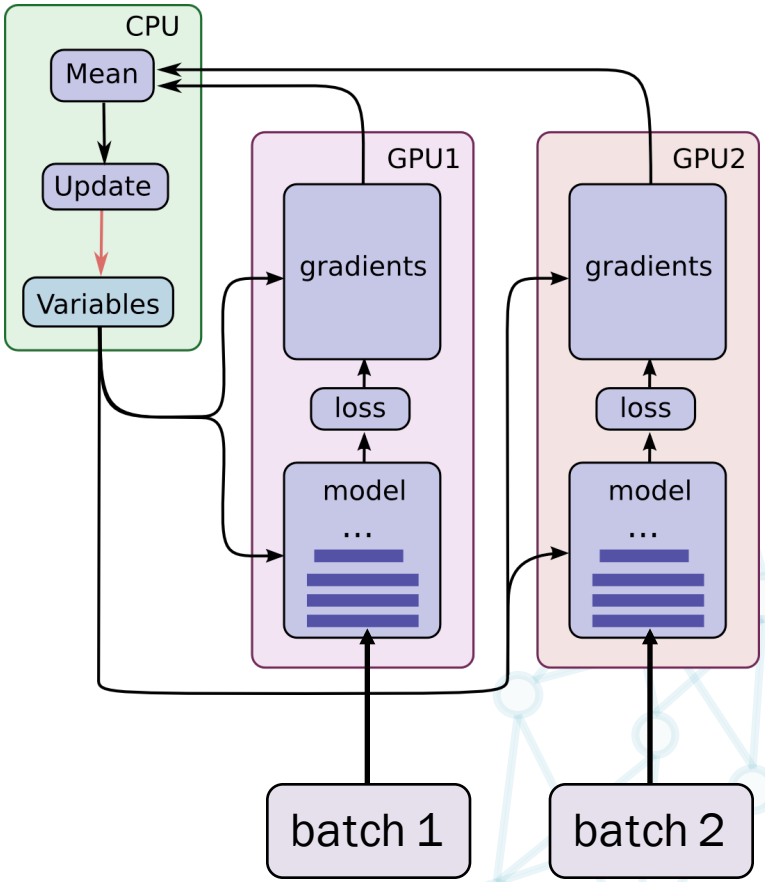
```
def create_weight_and_bias(weight_shape, bias_shape):
    with tf.device('/cpu:0'):
        weight = tf.get_variable("weight", initializer=tf.truncated_normal(weight_shape))
        bias = tf.get_variable("bias", initializer= tf.constant(0, shape=bias_shape))
        return weight, bias
```



Multi-GPU Implementation

- input data for GPUs

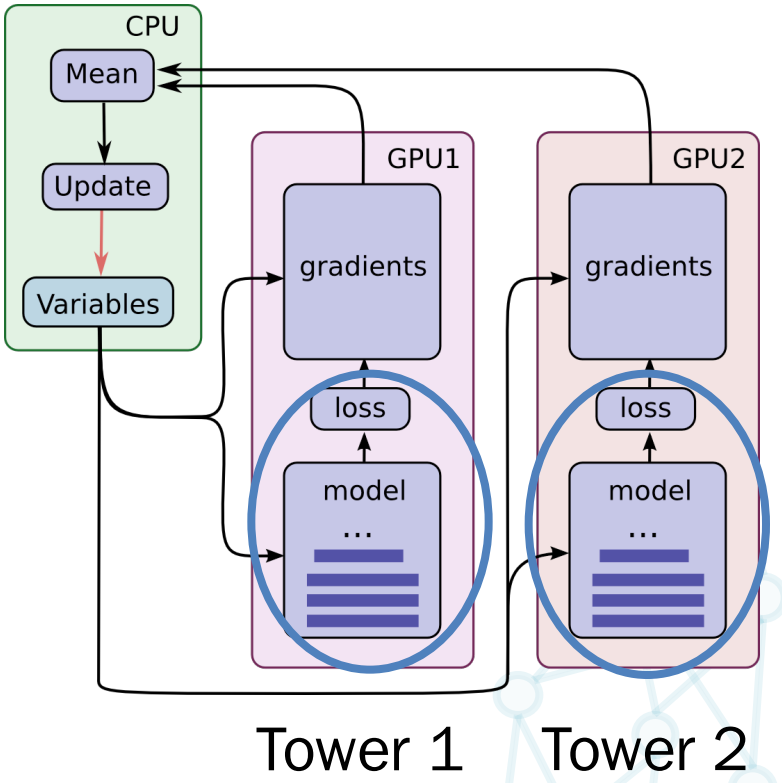
```
for i in range(num_gpus):
  with tf.device('/gpu:%d' % i):
    with tf.name_scope('tower_%d' % (i)) as scope:
      if i > 0:
        reuse = True
      else:
        reuse = False
      x_next = x_all[i * batch_size:(i + 1) * batch_size, :]
      y_next = y_all[i * batch_size:(i + 1) * batch_size, :]
      loss, acc = create_cnn(x_next, y_next, keep_prob,
                  reuse=reuse)
      grads = optimizer.compute_gradients(loss)
      tower_grads.append(grads)
      tower_acc.append(acc)
```



Multi-GPU Implementation

- create model and loss in GPUs

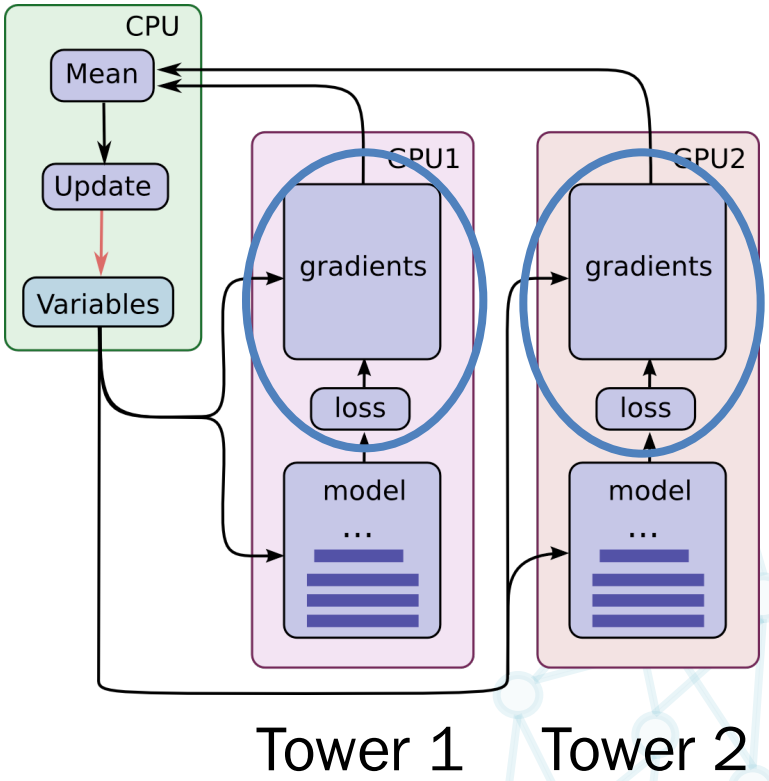
```
for i in range(num_gpus):
  with tf.device('/gpu:%d' % i):
    with tf.name_scope('tower_%d' % (i)) as scope:
      if i > 0:
        reuse = True
      else:
        reuse = False
      x_next = x_all[i * batch_size:(i + 1) * batch_size, :]
      y_next = y_all[i * batch_size:(i + 1) * batch_size, :]
      loss, acc = create_cnn(x_next, y_next, keep_prob,
                  reuse=reuse)
      grads = optimizer.compute_gradients(loss)
      tower_grads.append(grads)
      tower_acc.append(acc)
```



Multi-GPU Implementation

- compute gradients in GPUs

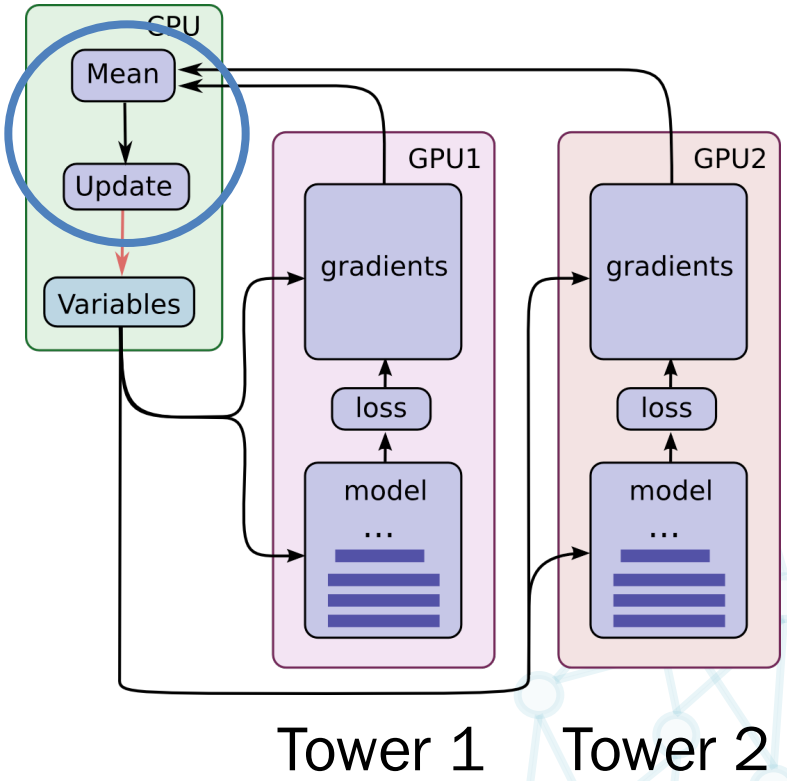
```
tower_grads = []
tower_acc = []
for i in range(num_gpus):
  with tf.device('/gpu:%d' % i):
    with tf.name_scope('tower_%d' % (i)) as scope:
      if i > 0:
        reuse = True
      else:
        reuse = False
      x_next = x_all[i * batch_size:(i + 1) * batch_size, :]
      y_next = y_all[i * batch_size:(i + 1) * batch_size, :]
      loss, acc = create_cnn(x_next, y_next, keep_prob,
                  reuse=reuse)
      grads = optimizer.compute_gradients(loss)
      tower_grads.append(grads)
      tower_acc.append(acc)
```



Multi-GPU Implementation

- average and update gradients in CPU

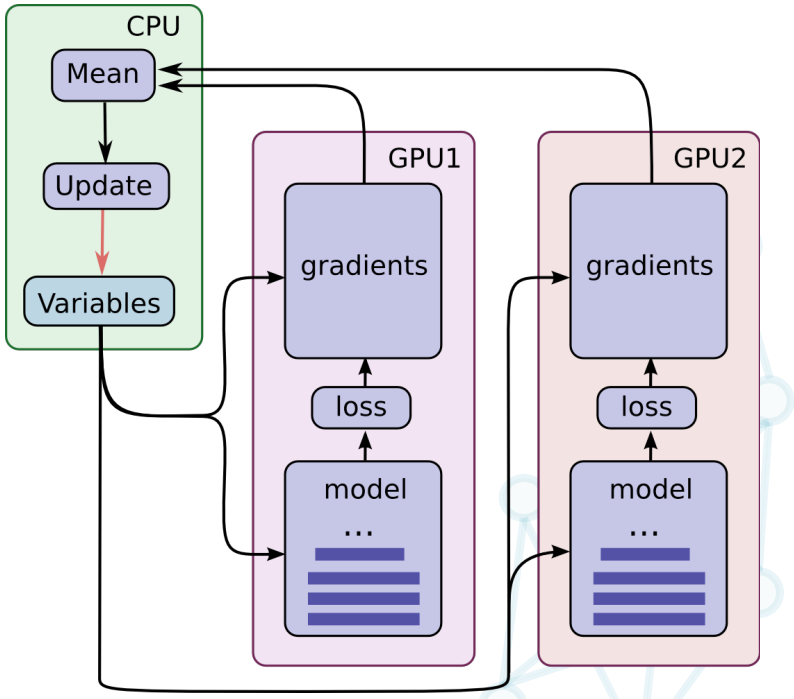
```
with graph.as_default(), tf.device('/cpu:0'):
    for i in range(num_gpus):
        with tf.device('/gpu:%d' % i):
            with tf.name_scope('tower_%d' % (i)) as scope:
                ...
                tower_grads.append(grads)
                tower_acc.append(acc)
    avg_grads = create_average_gradients(tower_grads)
    avg_acc = tf.reduce_mean(tower_acc)
    trainer = optimizer.apply_gradients(avg_grads)
```



Multi-GPU Implementation

- train Multi-GPU Implementation

```
for i in range(3000):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size * num_gpus)
        sess.run(trainer, feed_dict={x_all: batch_xs, y_all: batch_ys, keep_prob: 0.5})
```



講師資訊

Mark Chang

HTC Research & Healthcare

- Email: ckmarkoh at gmail dot com
- Blog: http://cpmarkchang.logdown.com
- Github: https://github.com/ckmarkoh
- Facebook: https://www.facebook.com/ckmarkoh.chang
- Slideshare: http://www.slideshare.net/ckmarkohchang