TensorFlow Tutorial

Presenter: Wei Li

Advisor: I-Chen Wu

Outline

- Deep Learning Frameworks
  - Deep Learning Frameworks
  - Deep Learning and GPUs
- TensorFlow Basic
  - Quick Strat
  - How to train a Network
  - Magic: TensorBoard
  - Keras: The Python Deep Learning library
  - Calling Python Program from C++
- Distributed Deep Learning
  - Distributed Training
  - Distributed Deep Learning Frameworks
- Distributed TensorFlow
  - Multi-GPUs Training
  - Distributed Training

Reference

- Shi, Shaohuai, and Xiaowen Chu. "Performance Modeling and Evaluation of Distributed Deep Learning Frameworks on GPUs." arXiv preprint arXiv:1711.05979 (2017).
- Canziani, Alfredo, Adam Paszke, and Eugenio Culurciello. "An analysis of deep neural network models for practical applications." arXiv preprint arXiv:1605.07678 (2016).
- CS231n : Lecture8 - Deep Learning Software
- CS231n : Lecture9 - CNN Architectures
- CS224n: TensorFlow Tutorial
- Which GPU(s) to Get for Deep Learning
- A Comparison between GeForce GTX 1080 and Tesla P100 for Deep Learning
- TitanXp vs GTX1080Ti for Machine Learning
- CNN-benchmarks
- How to Train a Very Large and Deep Model on One GPU?
- List of NVIDIA graphics processing units
- Memory usage and computational considerations
- Tensorflow Shared library and C++ example

Deep Learning Frameworks

Deep Learning Frameworks

Deep Learning and GPUs

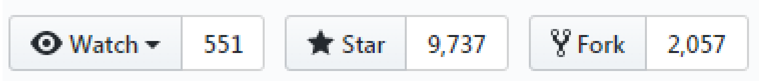
Deep Learning Frameworks

- In the past
  - Caffe (UC Berkeley) — Watch 2,098 | Star 21,536 | Fork 13,245
    - Yang-qing Jia (贾扬清)
      - the author of Caffe & leader of caffe2
      - one of the authors of GoogLeNet
  - Torch (NYU / Facebook) — Watch 681 | Star 7,495 | Fork 2,206
    - C/C++, Lua
    - most used in research(now is PyTorch)
    - be used by DeepMind(now is TensorFlow)
  - Theano(U Montréal) — Watch 565 | Star 7,355 | Fork 2,337
    - Authors: Yoshua Bengio & Ian Goodfellow etc.
    - Start from 2007(has died)

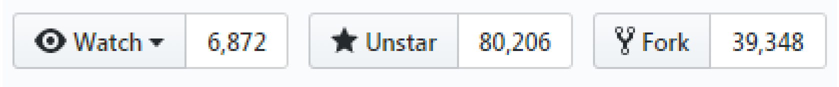
Deep Learning Frameworks(cont.)

- Today
  - Caffe (UC Berkeley) → Caffe2 (Facebook)
    - April 18, 2017

  - Torch (NYU/Facebook) → PyTorch (Facebook)
    - January 18, 2017
    - most used in **research**



  - Theano(U Montréal) → TensorFlow (Google)



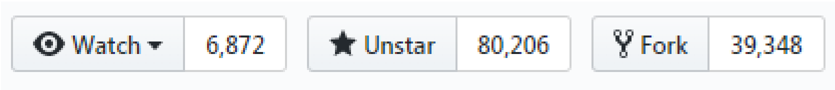
- Google
  - TensorFlow - one framework to rule them all
- Facebook
  - PyTorch - Research
  - Caffe2 - Production

Deep Learning Frameworks(cont.)

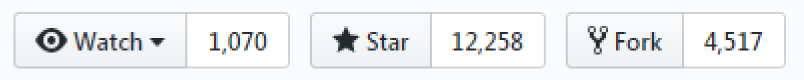
- Today
  - Caffe2 (Facebook)

  - PyTorch (Facebook)

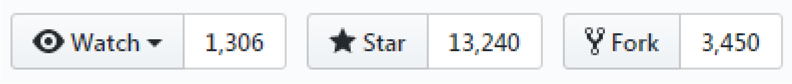
  - TensorFlow (Google)



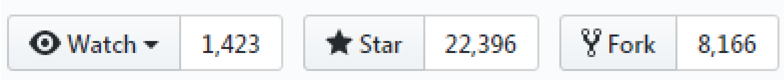
  - MXNet(Amazon)



    - Mu Li(李沐)
  - CNTK(Microsoft)



  - Keras(François Chollet)



    - A Deep Learning library for Python, that is simple, modular, and extensible.

Deep Learning Frameworks(cont.)

- **TensorFlow** is a safe bet for most projects. Not perfect but has huge community, wide usage.
  - Maybe pair with high-level wrapper (**Keras**, **Sonnet**, etc.)
  - Upper hand in distributed training
- **PyTorch** is best for research.
- Consider **Caffe, Caffe2**, or **TensorFlow** for production deployment
- Consider **TensorFlow** or **Caffe2** for mobile

Deep Learning Frameworks(cont.)

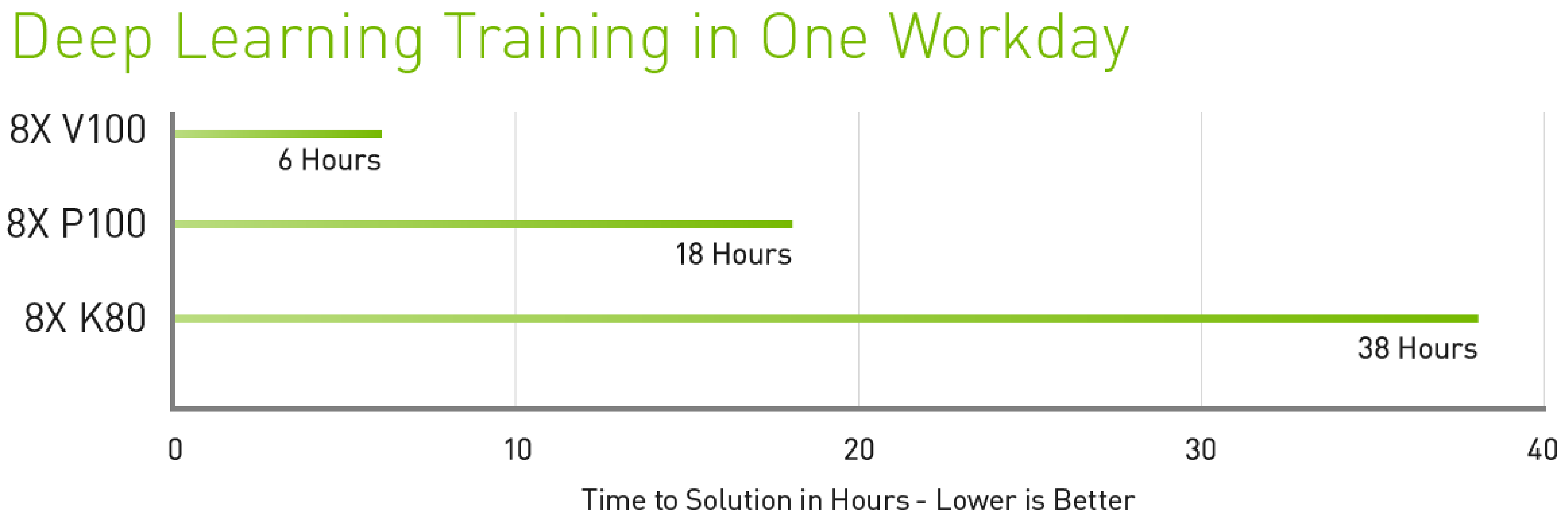
- Today
  - Caffe2 (Facebook) — Watch 471 | Star 6,404 | Fork 1,440
  - PyTorch (Facebook) — Watch 551 | Star 9,737 | Fork 2,057
  - TensorFlow (Google) — Watch 6,872 | Unstar 80,206 | Fork 39,348
  - MXNet(Amazon) — Watch 1,070 | Star 12,258 | Fork 4,517
    - Mu Li(李沐)
  - CNTK(Microsoft) — Watch 1,306 | Star 13,240 | Fork 3,450
  - Keras(**François Chollet**) — Watch 1,423 | Star 22,396 | Fork 8,166

Deep Learning and GPUs

- FLOPS(floating-point operations per second)
  - Single-precision
  - Double-precision(no need)
  - TFLOPS(teraFLOPS)
    - GeForce GTX 1080TI (10.6/11.34 tflops)
    - Tesla P100 SMX2 (10.61 tflops)
    - Tesla V100 for NVLink (15.7 tflops)
- ECC memory(Error-correcting code memory)
  - No need

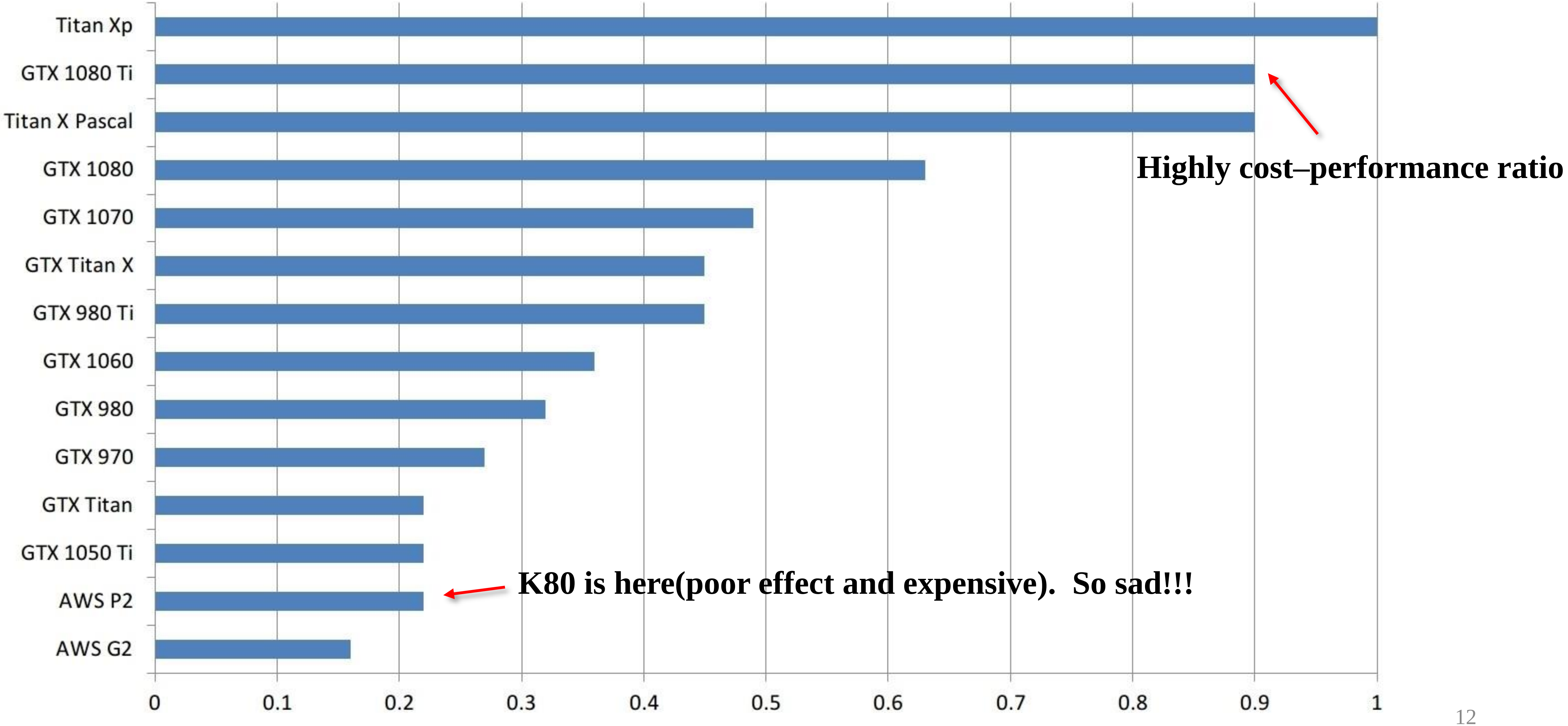
Deep Learning and GPUs

- Training time on caffe2(ResNet-50,90 epoch)



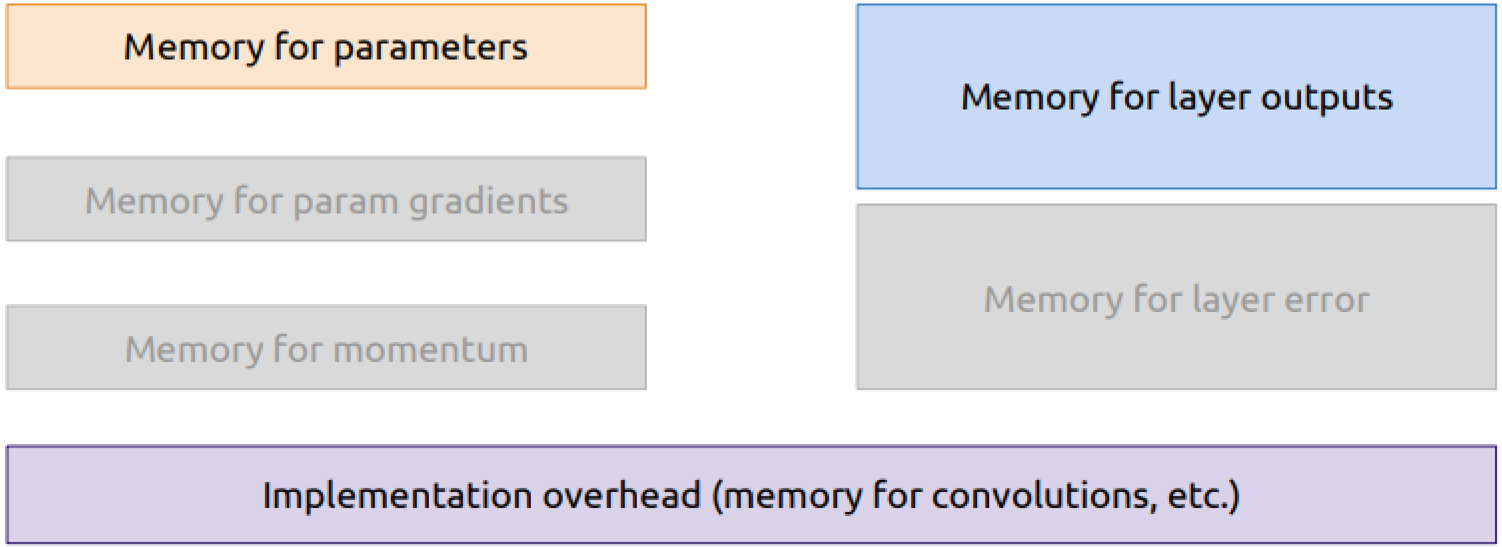
Server Config: Dual Xeon E5-2699 v4, 2.6GHz | 8X Tesla K80, Tesla P100 or Tesla V100 | ResNet-50 Training on Caffe2 for 90 Epochs with 1.28M ImageNet dataset

Deep Learning and GPUs(cont.)



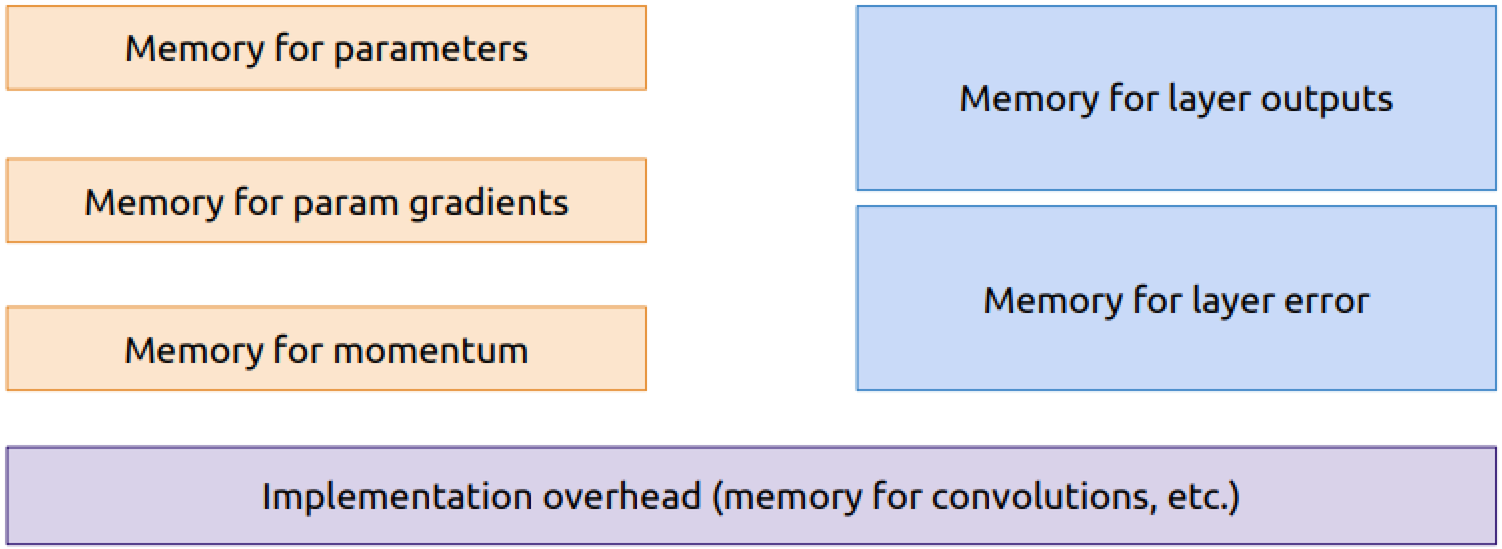
Deep Learning and GPUs(cont.)

- Total GPU memory requirements
  - Memory for model
  - Memory for layer outputs
- Total GPU memory in testing time(forward)



Deep Learning and GPUs(cont.)

- Total GPU memory requirements
  - Memory for model
  - Memory for layer outputs
- Total GPU memory in training time(forward + backward)



Deep Learning and GPUs(cont.)

- Total GPU memory requirements
  - Memory for model
    - params (parameters need to train)
    - $\boldsymbol{C_{in} * C_{out} * K^2}$
  - Memory for layer outputs
    - according to batch size
    - $\boldsymbol{C_{out} * H * W}$

$H *$ W: output shape
C_{in}: input channels
C_{out}: output channels
K: kernel size

Feature map Size		params	memory of layers
1x1x1000	fc,1000	4096*1000*1*1 = 4096000	1*1*1000 = 1000
1x1x4096	fc,4096	4096*4096*1*1 = 16777216	1*1*4096 = 4096
1x1x4096	fc,4096	512*4096*7*7 = 102760448	1*1*4096 = 4096
7x7x512	pooling	0	7*7*512 = 25088
14x14x512	3x3 conv,512	512*512*3*3 = 2359296	14*14*512 = 100352
14x14x512	3x3 conv,512	512*512*3*3 = 2359296	14*14*512 = 100352
14x14x512	3x3 conv,512	512*512*3*3 = 2359296	14*14*512 = 100352
14x14x512	pooling	0	14*14*512 = 100352
28x28x512	3x3 conv,512	512*512*3*3 = 2359296	28*28*512 = 401408
28x28x512	3x3 conv,512	512*512*3*3 = 2359296	28*28*512 = 401408
28x28x512	3x3 conv,512	256*512*3*3 = 1179648	28*28*512 = 401408
28x28x256	pooling	0	28*28*256 = 200704
56x56x256	3x3 conv,256	256*256*3*3 = 589824	56*56*256 = 802816
56x56x256	3x3 conv,256	256*256*3*3 = 589824	56*56*256 = 802816
56x56x256	3x3 conv,256	128*256*3*3 = 294912	56*56*256 = 802816
56x56x128	pooling	0	56*56*128 = 401408
112x112x128	3x3 conv,128	128*128*3*3 = 147456	112*112*128 = 1605632
112x112x128	3x3 conv,128	64*128*3*3 = 73728	112*112*128 = 1605632
112x112x64	pooling	0	112*112*64 = 802816
224x224x64	3x3 conv,64	64*64*3*3 = 36864	224*224*64 = 3211264
224x224x64	3x3 conv,64	3*64*3*3 = 1728	224*224*64 = 3211264
224x224x3	input	0	224*224*3 = 150528

- Number of params :
  - 138,344,128
  - **138,357,544(add bias)**
  - about 138M
- Memory of params:
  - float32 occupies 4 bytes
  - **138357544** * 4 = **553430176 byte**
  - **553430176/1024/1024 = 527.79MB**
  - about **528MB**

- Number of (memory of layers):
  - 15,237,608(15.2M)
- Memory of (memory of layers):
  - 15237608 * 4 = **60950432 byte**
  - about **58.12MB / image**

- Memory of (memory of layers):
  - about **58.12MB / image**

- **When training:**
  - **SGD + momentum**
  - **Batch size = 128**
  - **Memory for model:**
    - $\mathbf{528\ MB} * \mathbf{3} \approx \mathbf{1.54\ GB}$
    - **1 for params, 1 for SGD , 1 for momentum**
    - **If use Adam, need to x 4**
  - **Memory for outputs:**
    - $\mathbf{128 * 58.12\ MB} * \mathbf{2} = \mathbf{14878.72\ MB} \approx \mathbf{14.53GB}$
  - **Total memory:**
    - $\mathbf{1.54}\boldsymbol{GB} + \mathbf{14.53}\boldsymbol{GB} = \mathbf{16.07}\boldsymbol{GB}$

  - **So we need about 16 GB memory to train VGG16 Net with 128 batch size.**
  - **Assume we using GTX 1080(8GB) to train this Network**
    - **at least 2 GPUs**
    - **or reduce batch size**

Deep Learning and GPUs(cont.)

- Total GPU memory requirements
  - Memory for model
  - Memory for layer outputs
- Estimate computational complexity
  - FLOPs (float operations need to calculate)
    - In conv layer: $\underbrace{\boldsymbol{H * W * C_{out}}}_{\text{\# of output points}} * \underbrace{\boldsymbol{C_{in} * K^2}}_{\text{op of each points}}$
    - In fc layer: $\boldsymbol{M * N}$
    - In pooling layer: $\boldsymbol{H * W * C_{out} * K^2}$
    - In ReLU layer: $\boldsymbol{H * W * C_{out}}$

$H * \mathrm{W}$: output shape
C_{in}: input channels
C_{out}: output channels
K: kernel size
M: input shape
N: output shape

Feature map Size		FLOPs	
1x1x1000	fc,1000	1*1*4096*1000*1*1	= 4096000
1x1x4096	fc,4096	1*1*4096*4096*1*1	= 16777216
1x1x4096	fc,4096	1*1*512*4096*7*7	= 102760448
7x7x512	pooling		
14x14x512	3x3 conv,512	14*14*512*512*3*3	= 462422016
14x14x512	3x3 conv,512	14*14*512*512*3*3	= 462422016
14x14x512	3x3 conv,512	14*14*512*512*3*3	= 462422016
14x14x512	pooling		
28x28x512	3x3 conv,512	28*28*512*512*3*3	= 1849688064
28x28x512	3x3 conv,512	28*28*512*512*3*3	= 1849688064
28x28x512	3x3 conv,512	28*28*256*512*3*3	= 924844032
28x28x256	pooling		
56x56x256	3x3 conv,256	56*56*256*256*3*3	= 1849688064
56x56x256	3x3 conv,256	56*56*256*256*3*3	= 1849688064
56x56x256	3x3 conv,256	56*56*128*256*3*3	= 924844032
56x56x128	pooling		
112x112x128	3x3 conv,128	112*112*128*128*3*3	= 1849688064
112x112x128	3x3 conv,128	112*112*64*128*3*3	= 924844032
112x112x64	pooling		
224x224x64	3x3 conv,64	224*224*64*64*3*3	= 1849688064
224x224x64	3x3 conv,64	224*224*3*64*3*3	= 86704128
224x224x3	input		

- Number of FLOPs :
  - 15470264320
  - About **15.4 GFLOPS**

- **About training time:**
  - **Depend on your GPUs**
  - **Depend on your framework**
  - **Depend on your code implement**
  - **etc.**

Deep Learning and GPUs(cont.)

network	GPU	params	batch size	epoch	training time	accuracy(%)
Lecun-Network	GTX1080TI	62k	128	200	30 min	76.25
Network-in-Network	GTX1080TI	0.97M	128	200	1 h 40 min	91.63
Vgg19-Network	GTX1080TI	39M	128	200	1 h 53 min	93.53
Residual-Network20	GTX1080TI	0.27M	128	200	47 min	92.16
Residual-Network32	GTX1080TI	0.47M	128	200	1 h 13 min	92.86
Residual-Network110	GTX1080TI	1.7M	128	200	4 h 30 min	94.44
Wide-resnet 16x8	GTX1080TI	11.3M	128	200	5 h 1 min	95.13
DenseNet-100x12	GTX1080TI	0.85M	64	250	17 h 20 min	94.91
DenseNet-100x24	GTX1080TI	3.3M	64	250	22 h 27 min	95.30
DenseNet-160x24	1080 x 2	7.5M	64	250	50 h 20 min	95.90
ResNeXt-4x64d	GTX1080TI	20M	120	250	21 h 3 min	95.19
SENet(ResNeXt-4x64d)	GTX1080TI	20M	120	250	21 h 57 min	95.60

TensorFlow Basic

Quick Strat

How to train a Network

Magic: TensorBoard

Keras: The Python Deep Learning library

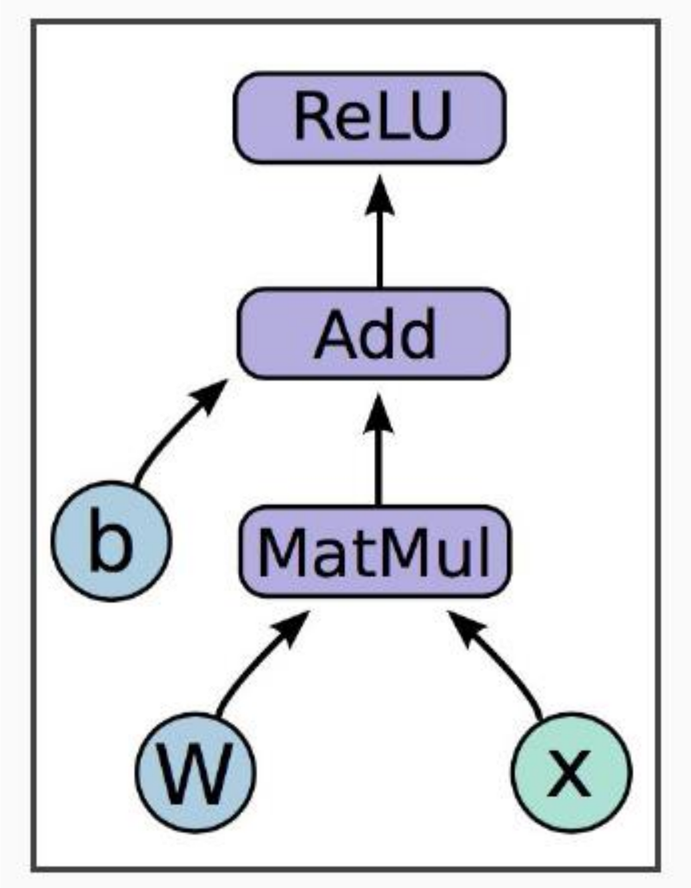
Calling TensorFlow Python Program from C++

Preparation

- Installation:
  - Official Setup
  - NVIDIA Driver & PyTorch(TensorFlow) installation
- Tutorials:
  - TensorFlow: Getting Started
  - Stanford CS 20SI: Tensorflow for Deep Learning Research
  - TensorFlow-Examples

Quick Start

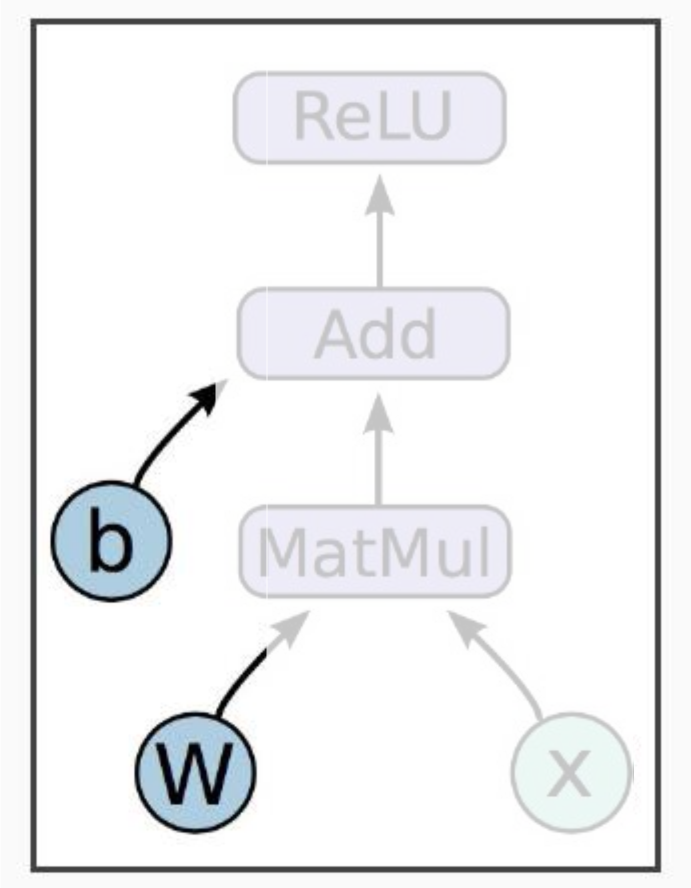
- Computation graph
- Variables(mostly parameters)
- Placeholders(inputs, labels, …)
- Mathematical operations



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

Quick Start(cont.)

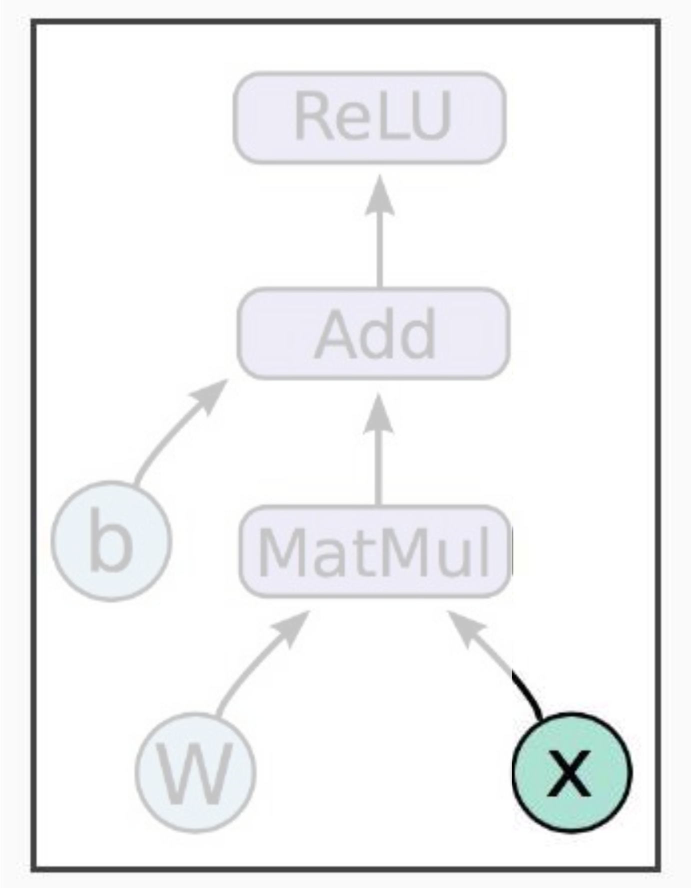
- Computation graph
- Variables(mostly parameters)
- Placeholders(inputs, labels, …)
- Mathematical operations



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

Quick Start(cont.)

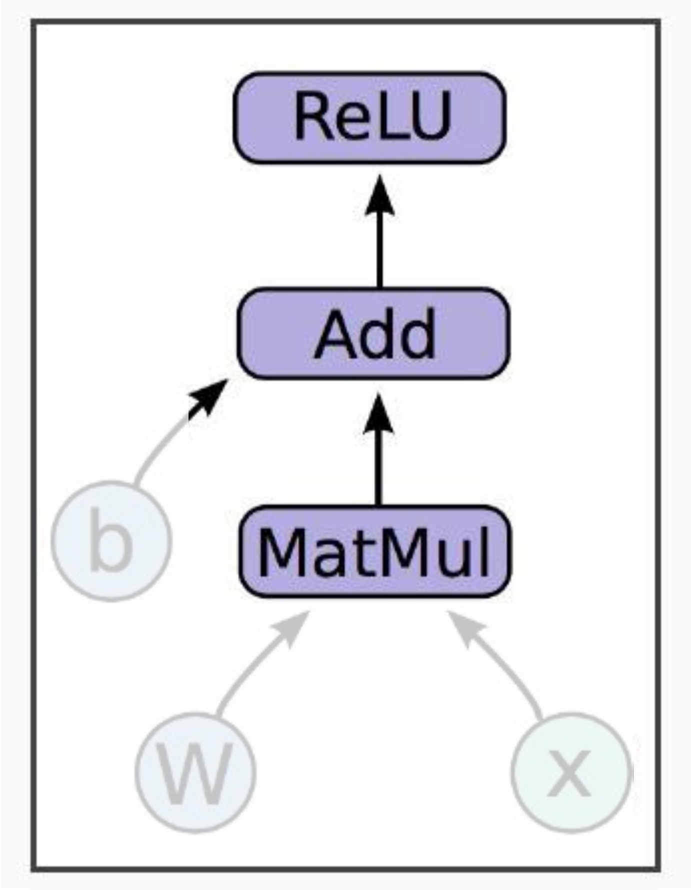
- Computation graph
- Variables(mostly parameters)
- Placeholders(inputs, labels, …)
- Mathematical operations



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

Quick Start(cont.)

- Computation graph
- Variables(mostly parameters)
- Placeholders(inputs, labels, …)
- Mathematical operations



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

How to train a Network

- Define a graph
  - build a graph using variables and placeholders
- Define the loss
  - use placeholder for labels
  - build loss node using labels and prediction
- Create operations
  - train op, evaluate op etc.
- Create a session
  - deployed the graph onto a session, which is the execution environment
- Train the Model
  - also include testing

- Define a graph
- Define the loss
- Create the operations
- Create a session
- Train the model

```python
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf

def main(_):
  # Import data
  mnist = input_data.read_data_sets("/tmp/input_data", one_hot=True)

  # Create the model
  x = tf.placeholder(tf.float32, [None, 784])
  W = tf.Variable(tf.zeros([784, 10]))
  b = tf.Variable(tf.zeros([10]))
  y = tf.matmul(x, W) + b

  # Define loss
  y_ = tf.placeholder(tf.float32, [None, 10])
  cross_entropy = tf.reduce_mean(
                tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y))
  # Define train_step & evaluate_step
  train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
  correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
  accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

  # Create a session
  sess = tf.Session()
  sess.run(tf.global_variables_initializer())

  # Train
  for it in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

  # Test trained model
  print(sess.run(accuracy, feed_dict={x: mnist.test.images,y_: mnist.test.labels}))

if __name__ == '__main__':
  tf.app.run(main=main)
```

- Define a graph
- Define the loss
- Create the operations
- Create a session
- Train the model

```python
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf

def main(_):
  # Import data
  mnist = input_data.read_data_sets("/tmp/input_data", one_hot=True)

  # Create the model
  x = tf.placeholder(tf.float32, [None, 784])
  W = tf.Variable(tf.zeros([784, 10]))
  b = tf.Variable(tf.zeros([10]))
  y = tf.matmul(x, W) + b

  # Define loss
  y_ = tf.placeholder(tf.float32, [None, 10])
  cross_entropy = tf.reduce_mean(
              tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y))
  # Define train_step & evaluate_step
  train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
  correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
  accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

  # Create a session
  sess = tf.Session()
  sess.run(tf.global_variables_initializer())

  # Train
  for it in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

  # Test trained model
  print(sess.run(accuracy, feed_dict={x: mnist.test.images,y_: mnist.test.labels}))

if __name__ == '__main__':
  tf.app.run(main=main)
```

- Define a graph
- Define the loss
- Create the operations
- Create a session
- Train the model

```python
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf

def main(_):
  # Import data
  mnist = input_data.read_data_sets("/tmp/input_data", one_hot=True)

  # Create the model
  x = tf.placeholder(tf.float32, [None, 784])
  W = tf.Variable(tf.zeros([784, 10]))
  b = tf.Variable(tf.zeros([10]))
  y = tf.matmul(x, W) + b

  # Define loss
  y_ = tf.placeholder(tf.float32, [None, 10])
  cross_entropy = tf.reduce_mean(
                tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y))
  # Define train_step & evaluate_step
  train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
  correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
  accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

  # Create a session
  sess = tf.Session()
  sess.run(tf.global_variables_initializer())

  # Train
  for it in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

  # Test trained model
  print(sess.run(accuracy, feed_dict={x: mnist.test.images,y_: mnist.test.labels}))

if __name__ == '__main__':
  tf.app.run(main=main)
```

- Define a graph
- Define the loss
- Create the operations
- Create a session
- Train the model

```
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf

def main(_):
  # Import data
  mnist = input_data.read_data_sets("/tmp/input_data", one_hot=True)

  # Create the model
  x = tf.placeholder(tf.float32, [None, 784])
  W = tf.Variable(tf.zeros([784, 10]))
  b = tf.Variable(tf.zeros([10]))
  y = tf.matmul(x, W) + b

  # Define loss
  y_ = tf.placeholder(tf.float32, [None, 10])
  cross_entropy = tf.reduce_mean(
                tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y))
  # Define train_step & evaluate_step
  train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
  correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
  accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

  # Create a session
  sess = tf.Session()
  sess.run(tf.global_variables_initializer())

  # Train
  for it in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

  # Test trained model
  print(sess.run(accuracy, feed_dict={x: mnist.test.images,y_: mnist.test.labels}))

if __name__ == '__main__':
  tf.app.run(main=main)
```

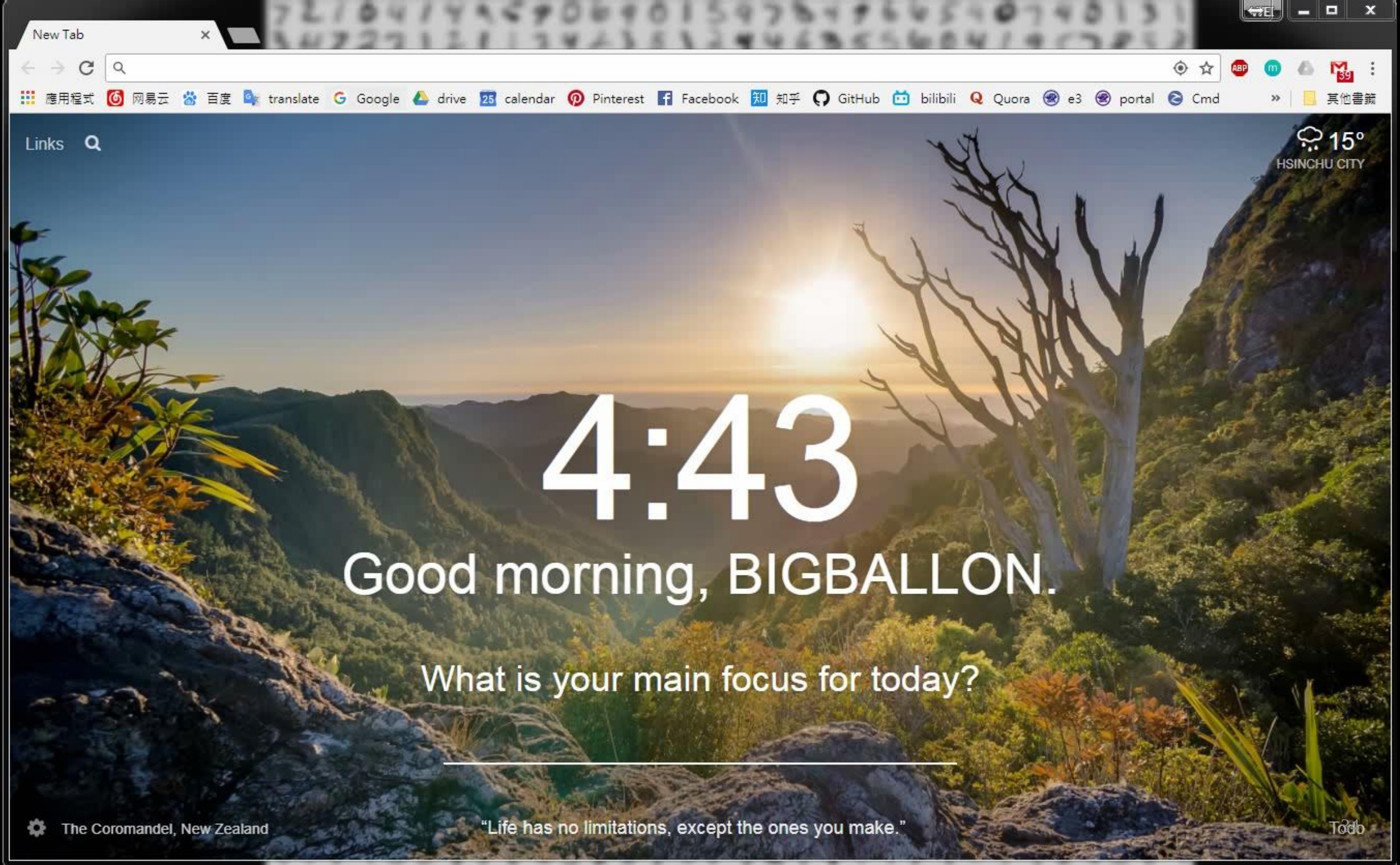
- Define a graph
- Define the loss
- Create the operations
- Create a session
- Train the model

```python
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf

def main(_):
  # Import data
  mnist = input_data.read_data_sets("/tmp/input_data", one_hot=True)

  # Create the model
  x = tf.placeholder(tf.float32, [None, 784])
  W = tf.Variable(tf.zeros([784, 10]))
  b = tf.Variable(tf.zeros([10]))
  y = tf.matmul(x, W) + b

  # Define loss
  y_ = tf.placeholder(tf.float32, [None, 10])
  cross_entropy = tf.reduce_mean(
                tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y))
  # Define train_step & evaluate_step
  train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
  correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
  accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

  # Create a session
  sess = tf.Session()
  sess.run(tf.global_variables_initializer())

  # Train
  for it in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

  # Test trained model
  print(sess.run(accuracy, feed_dict={x: mnist.test.images,y_: mnist.test.labels}))

if __name__ == '__main__':
  tf.app.run(main=main)
```

Magic: TensorBoard

- TensorBoard is a suite of web applications for inspecting and understanding your TensorFlow runs and graphs.
  - tf-dev-summit-tensorboard-tutorial
  - summaries_and_tensorboard
  - mnist_with_summaries.py

- Usage: `tensorboard --logdir path/to/logs`
- For remote: `ssh -L 6006:127.0.0.1:6006 dl2017@140.113.xxx.xxx -p xxxx`


New Tab
應用程式
网易云
百度
translate
Google
drive
calendar
Pinterest
Facebook
知乎
GitHub
bilibili
Quora
e3
portal
Cmd
其他書籤
Links
15°
HSINCHU CITY
4:43
Good morning, BIGBALLON.
What is your main focus for today?
The Coromandel, New Zealand
“Life has no limitations, except the ones you make.”
Todo

- Attach summaries
- Merge summary op
- Create a writer
- Run op in session
- Save summary

```python
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf

def main(_):
  mnist = input_data.read_data_sets("./data", one_hot=True)
  x = tf.placeholder(tf.float32, [None, 784])
  W = tf.Variable(tf.zeros([784, 10]))
  b = tf.Variable(tf.zeros([10]))
  y = tf.matmul(x, W) + b
  y_ = tf.placeholder(tf.float32, [None, 10])
  cross_entropy = tf.reduce_mean(
      tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y))
  train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
  correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
  accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

  # Attach summaries to loss & accuracy
  tf.summary.scalar('loss', cross_entropy)
  tf.summary.scalar('accuracy', accuracy)
  merged = tf.summary.merge_all()

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

  # create a writer to save logs
  writer = tf.summary.FileWriter('./tb_logs',sess.graph)
  for it in range(10000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    # run merged op in session
    _, summary = sess.run([train_step,merged], feed_dict={x: batch_xs, y_: batch_ys})
    writer.add_summary(summary, it)

  final_test = sess.run(accuracy, feed_dict={x: mnist.test.images,y_: mnist.test.labels})
  print("final test acc: %.2f"%final_test)

if __name__ == '__main__':
  tf.app.run(main=main)
```

- Attach summaries
- Merge summary op
- Create a writer
- Run op in session
- Save summary

```python
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf

def main(_):
  mnist = input_data.read_data_sets("./data", one_hot=True)
  x = tf.placeholder(tf.float32, [None, 784])
  W = tf.Variable(tf.zeros([784, 10]))
  b = tf.Variable(tf.zeros([10]))
  y = tf.matmul(x, W) + b
  y_ = tf.placeholder(tf.float32, [None, 10])
  cross_entropy = tf.reduce_mean(
      tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y))
  train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
  correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
  accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

  # Attach summaries to loss & accuracy
  tf.summary.scalar('loss', cross_entropy)
  tf.summary.scalar('accuracy', accuracy)
  merged = tf.summary.merge_all()

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

  # create a writer to save logs
  writer = tf.summary.FileWriter('./tb_logs',sess.graph)
  for it in range(10000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    # run merged op in session
    _, summary = sess.run([train_step,merged], feed_dict={x: batch_xs, y_: batch_ys})
    writer.add_summary(summary, it)

  final_test = sess.run(accuracy, feed_dict={x: mnist.test.images,y_: mnist.test.labels})
  print("final test acc: %.2f"%final_test)

if __name__ == '__main__':
  tf.app.run(main=main)
```

- Attach summaries
- Merge summary op
- Create a writer
- Run op in session
- Save summary

- code is here

```python
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf

def main(_):
  mnist = input_data.read_data_sets("./data", one_hot=True)
  x = tf.placeholder(tf.float32, [None, 784])
  W = tf.Variable(tf.zeros([784, 10]))
  b = tf.Variable(tf.zeros([10]))
  y = tf.matmul(x, W) + b
  y_ = tf.placeholder(tf.float32, [None, 10])
  cross_entropy = tf.reduce_mean(
      tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y))
  train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
  correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
  accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

  # Attach summaries to loss & accuracy
  tf.summary.scalar('loss', cross_entropy)
  tf.summary.scalar('accuracy', accuracy)
  merged = tf.summary.merge_all()

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

  # create a writer to save logs
  writer = tf.summary.FileWriter('./tb_logs',sess.graph)
  for it in range(10000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    # run merged op in session
    _, summary = sess.run([train_step,merged], feed_dict={x: batch_xs, y_: batch_ys})
    writer.add_summary(summary, it)

  final_test = sess.run(accuracy, feed_dict={x: mnist.test.images,y_: mnist.test.labels})
  print("final test acc: %.2f"%final_test)

if __name__ == '__main__':
  tf.app.run(main=main)
```

Keras:The Python Deep Learning library

- Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano.
  - High-Level, User friendliness
  - Easy to use(very easy)
  - Support TensorFlow backend
  - Good documentation
  - Quick start : [Getting started: 30 seconds to Keras]
  - See more: [cifar-10-cnn]

Two ways to build model

- The Sequential model API

```
model = Sequential()
model.add(Dense(32, input_shape=(500,)))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

- Model class API

```
from keras.models import Model
from keras.layers import Input, Dense

a = Input(shape=(32,))
b = Dense(32)(a)
model = Model(inputs=a, outputs=b)
```

```python
def build_model():
    model = Sequential()
    model.add(Conv2D(6, (5, 5), padding='valid', activation = 'relu', kernel_initializer='he_normal', input_shape=(32,32,3)))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))
    model.add(Conv2D(16, (5, 5), padding='valid', activation = 'relu', kernel_initializer='he_normal'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))
    model.add(Flatten())
    model.add(Dense(120, activation = 'relu', kernel_initializer='he_normal'))
    model.add(Dense(84, activation = 'relu', kernel_initializer='he_normal'))
    model.add(Dense(10, activation = 'softmax', kernel_initializer='he_normal'))
    sgd = optimizers.SGD(lr=.1, momentum=0.9, nesterov=True)
    model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
    return model

def scheduler(epoch):
    if epoch <= 60:
        return 0.05
    if epoch <= 120:
        return 0.01
    return 0.001

if __name__ == '__main__':

    # load data
    (x_train, y_train), (x_test, y_test) = cifar10.load_data()
    y_train = keras.utils.to_categorical(y_train, 10)
    y_test = keras.utils.to_categorical(y_test, 10)
    x_train = x_train.astype('float32') / 255.0
    x_test = x_test.astype('float32') / 255.0

    # build network
    model = build_model()
    print(model.summary())

    # set callback
    cbks = [TensorBoard(log_dir='./lenet', histogram_freq=0),
            LearningRateScheduler(scheduler)]

    # start traing
    model.fit(x_train, y_train,batch_size=128,epochs=200,callbacks=cbks,
              validation_data=(x_test, y_test), shuffle=True)
    # save model
    model.save('lenet.h5')
```

```python
def build_model():
    model = Sequential()
    model.add(Conv2D(6, (5, 5), padding='valid', activation = 'relu', kernel_initializer='he_normal', input_shape=(32,32,3)))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))
    model.add(Conv2D(16, (5, 5), padding='valid', activation = 'relu', kernel_initializer='he_normal'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))
    model.add(Flatten())
    model.add(Dense(120, activation = 'relu', kernel_initializer='he_normal'))
    model.add(Dense(84, activation = 'relu', kernel_initializer='he_normal'))
    model.add(Dense(10, activation = 'softmax', kernel_initializer='he_normal'))
    sgd = optimizers.SGD(lr=.1, momentum=0.9, nesterov=True)
    model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
    return model

def scheduler(epoch):
    if epoch <= 60:
        return 0.05
    if epoch <= 120:
        return 0.01
    return 0.001

if __name__ == '__main__':

    # load data
    (x_train, y_train), (x_test, y_test) = cifar10.load_data()
    y_train = keras.utils.to_categorical(y_train, 10)
    y_test = keras.utils.to_categorical(y_test, 10)
    x_train = x_train.astype('float32') / 255.0
    x_test = x_test.astype('float32') / 255.0

    # build network
    model = build_model()
    print(model.summary())

    # set callback
    cbks = [TensorBoard(log_dir='./lenet', histogram_freq=0),
            LearningRateScheduler(scheduler)]

    # start traing
    model.fit(x_train, y_train,batch_size=128,epochs=200,callbacks=cbks,
                validation_data=(x_test, y_test), shuffle=True)
    # save model
    model.save('lenet.h5')
```

```
Epoch 1/200
50000/50000 [==============================] - 2s 43us/step - loss: 2.1173 - acc: 0.2035 - val_loss: 1.7434 - val_acc: 0.3745
Epoch 2/200
50000/50000 [==============================] - 2s 34us/step - loss: 1.6598 - acc: 0.4047 - val_loss: 1.6489 - val_acc: 0.4188
Epoch 3/200
50000/50000 [==============================] - 2s 38us/step - loss: 1.5161 - acc: 0.4606 - val_loss: 1.5194 - val_acc: 0.4615
Epoch 4/200
50000/50000 [==============================] - 2s 34us/step - loss: 1.4285 - acc: 0.4929 - val_loss: 1.4456 - val_acc: 0.4864
```

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 6)         456
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 16)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_1 (Dense)              (None, 120)               48120
_________________________________________________________________
dense_2 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_3 (Dense)              (None, 10)                850
=================================================================
Total params: 62,006
Trainable params: 62,006
Non-trainable params: 0
```

ImageDataGenerator

```
keras.preprocessing.image.ImageDataGenerator(featurewise_center=False,
    samplewise_center=False,
    featurewise_std_normalization=False,
    samplewise_std_normalization=False,
    zca_whitening=False,
    zca_epsilon=1e-6,
    rotation_range=0.,
    width_shift_range=0.,
    height_shift_range=0.,
    shear_range=0.,
    zoom_range=0.,
    channel_shift_range=0.,
    fill_mode='nearest',
    cval=0.,
    horizontal_flip=False,
    vertical_flip=False,
    rescale=None,
    preprocessing_function=None,
    data_format=K.image_data_format())
```

```
datagen = ImageDataGenerator(horizontal_flip=True,
                             width_shift_range=0.125,height_shift_range=0.125,
                             fill_mode='constant',cval=0.)

datagen.fit(x_train)
model.fit_generator(datagen.flow(x_train, y_train,batch_size=batch_size),
                    steps_per_epoch=iterations,
                    epochs=epochs,
                    callbacks=cbks,
                    validation_data=(x_test, y_test))
```

Callbacks

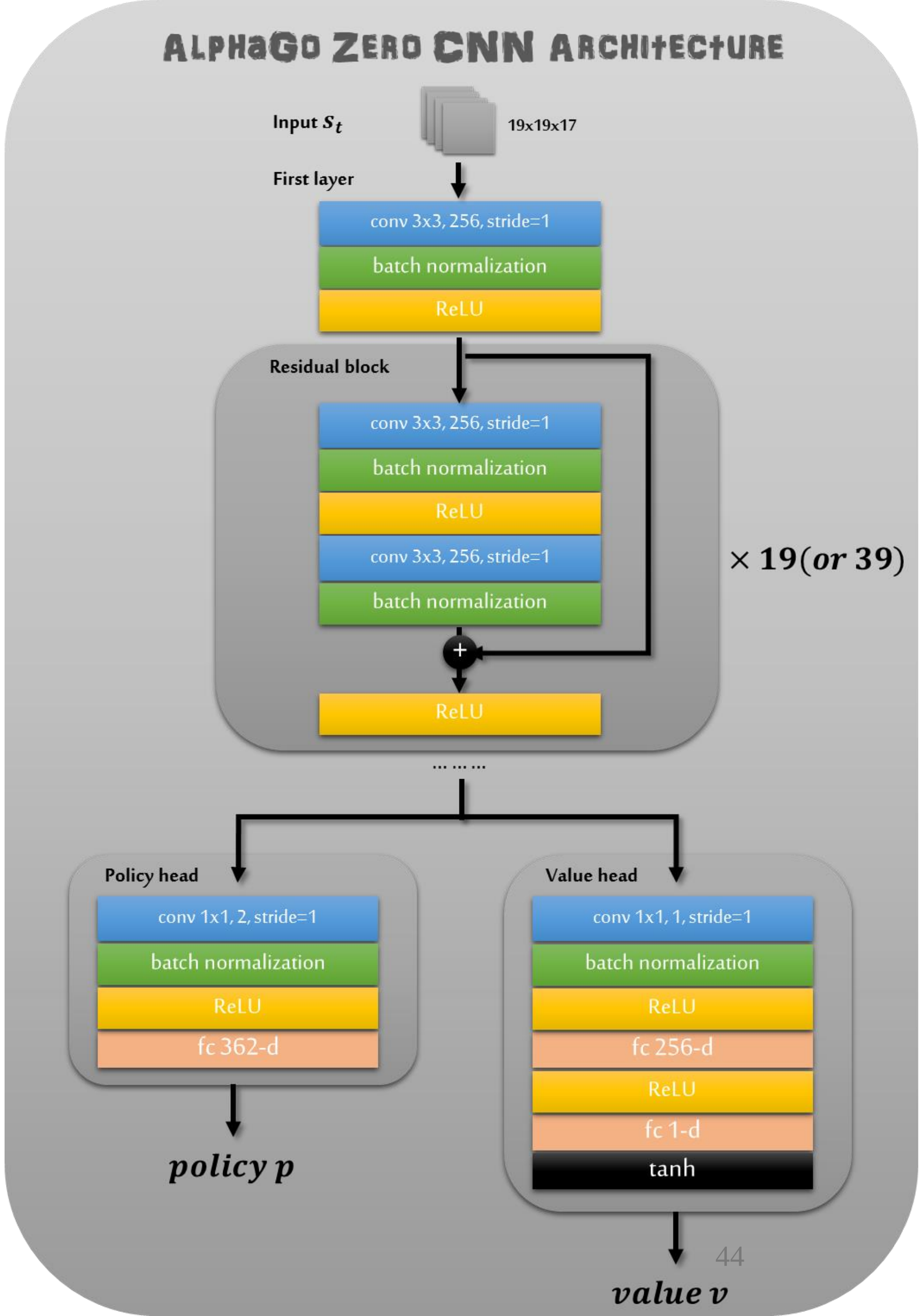
- TensorBoard()
- LearningRateScheduler()
- ModelCheckpoint()
- etc.

```
# set callback
cbks = [TensorBoard(log_dir='./resnet_32/', histogram_freq=0),
        LearningRateScheduler(scheduler),
        ModelCheckpoint('./checkpoint-{epoch}.h5', save_best_only=False, mode='auto', period=10)]

resnet.fit_generator(datagen.flow(x_train, y_train,batch_size=batch_size),
                     steps_per_epoch=iterations,
                     epochs=epochs,
                     callbacks=cbks,
                     validation_data=(x_test, y_test))
```

Multi-output(AlphaGo Zero)

```python
class AGZeroModel:
  def build_model(self):
    N = self.N
    position = Input((N, N, 17))
    resnet = ResNet(n_stages=N)
    resnet.create(N, N, 17)
    x = resnet.model(position)

    dist = Conv2D(2, (1, 1))(x)
    dist = BatchNormalization()(dist)
    dist = Activation('relu')(dist)
    dist = Flatten()(dist)
    dist = Dense(N * N + 1, activation='softmax',
          name='distribution')(dist)

    res = Conv2D(1, (1, 1))(x)
    res = BatchNormalization()(res)
    res = Activation('relu')(res)
    res = Flatten()(res)
    res = Dense(256, activation='relu')(res)
    res = Dense(1, activation=' tanh ', name='result')(res)

    self.model = Model(position, [dist, res])
    self.model.compile('adam',
      ['categorical_crossentropy', 'binary_crossentropy'])
    self.model.summary()
```

Another model need to implement



Calling Python Program from C++

- See tensorflow_tricks/C_Python/
- This Demo will show how to call an pre-trained imagenet model to predict picture in C++.

```
├── prediction.cpp          % C++ file
├── vgg_model.py            % TensorFlow vgg model
├── makefile                % Compile file
├── little_demo             % An simple Demo
└── test_pic/               % Test pictures
        ├── cat.jpeg
        ├── puzzle.jpeg
        └── tiger.jpeg
```

```c
#include <Python.h>
#include <stdio.h>
#include <string.h>


int main(int argc, char *argv[]){

    Py_Initialize();
    if( !Py_IsInitialized() ){
        printf("Initialize failed\n");
        return -1;
    }
    PyRun_SimpleString("import sys");
    PyRun_SimpleString("sys.path.append('./')");

    PyObject *pName,*pModule,*pDict,*pFunc;

    // PyString_FromString for python2.x
    // PyUnicode_DecodeFSDefault for python3.x
    pName = PyUnicode_DecodeFSDefault("vgg_model");

    pModule = PyImport_Import(pName);
    if ( !pModule ){
        printf("Can't find Module\n");
        return -1;
    }
    pDict = PyModule_GetDict(pModule);
    if ( !pDict ){
        return -1;
    }
    pFunc = PyDict_GetItemString(pDict, "predict");
    if ( !pFunc || !PyCallable_Check(pFunc) ){
        printf("can't find function [predict]\n");
        return -1;
    }

    printf(" ===========> START CALL PYTHON SCRIPT <===========\n");

    printf(" ===========> 1st CALL <===========\n");
    PyObject_CallObject(pFunc,NULL);
    printf(" ===========> 2nd CALL <===========\n");
    PyObject_CallObject(pFunc,NULL);
    printf(" ===========> 3rd CALL <===========\n");
    PyObject_CallObject(pFunc,NULL);
    printf(" ===========> 4th CALL <===========\n");
    PyObject_CallObject(pFunc,NULL);

    printf(" ===========> CALLING FINISHED <===========\n");

    Py_DECREF(pName);
    Py_DECREF(pModule);

    // close Python
    Py_Finalize();
    return 0;
}
```

```
===========> START CALL PYTHON SCRIPT <===========
===========> 1st CALL <===========
Please input picture file to predict: huhu
file not exist!
===========> 2nd CALL <===========
Please input picture file to predict: test_pic/cat.jpeg
Predicted:  [('n02124075', 'Egyptian_cat', 0.93183666)]
===========> 3rd CALL <===========
Please input picture file to predict: test_pic/tiger.jpeg
Predicted:  [('n02129604', 'tiger', 0.82598984)]
===========> 4th CALL <===========
Please input picture file to predict: test_pic/puzzle.jpeg
Predicted:  [('n03598930', 'jigsaw_puzzle', 0.99813461)]
===========> CALLING FINISHED <===========
(deeplearning) bg@bg-cgi:~/Desktop/C_python$
```

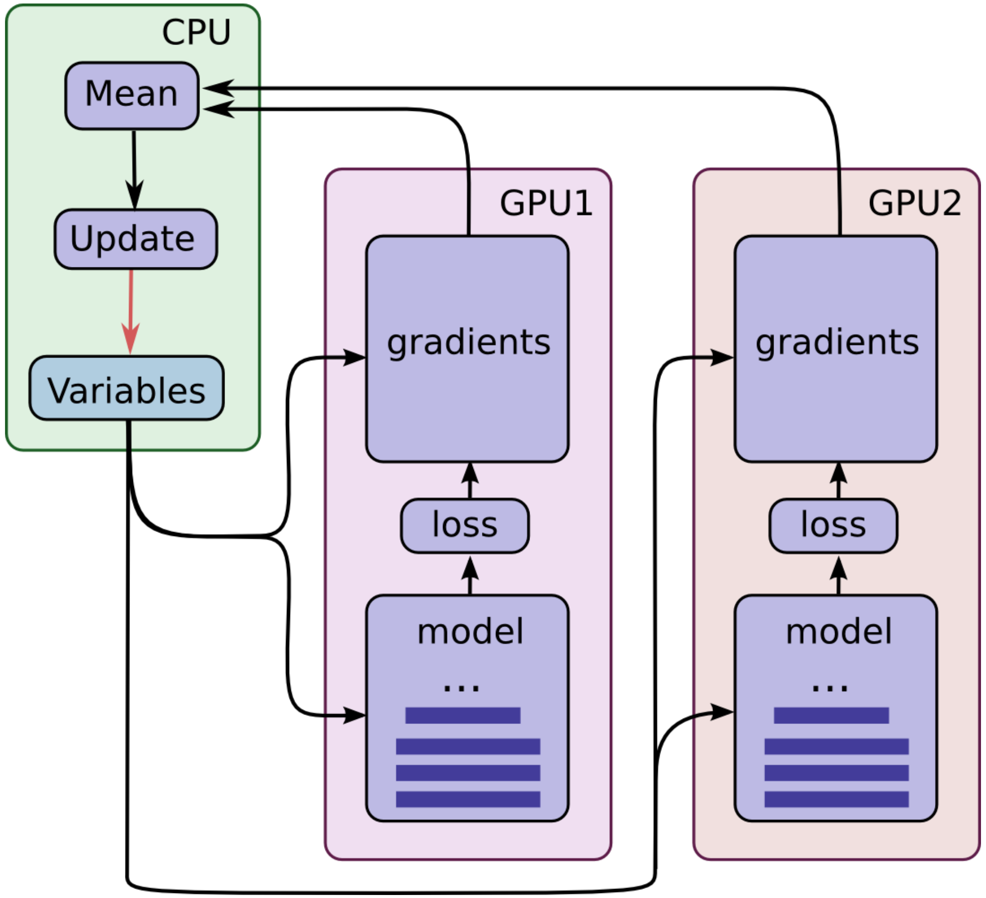
Distributed Deep Learning

Distributed Training

Distributed Deep Learning Frameworks

Distributed Training

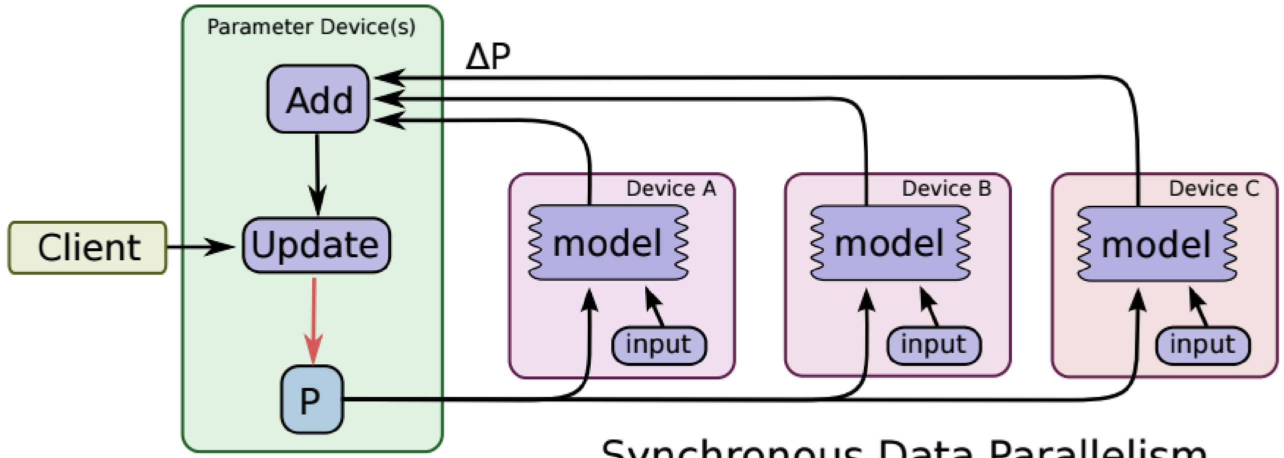
- One machine with one GPU
- One machine with many GPUs
  - TensorFlow
  - Caffe / Caffe 2
  - PyTorch
  - MXNet
  - CNTK
  - Keras



Distributed Training(cont.)

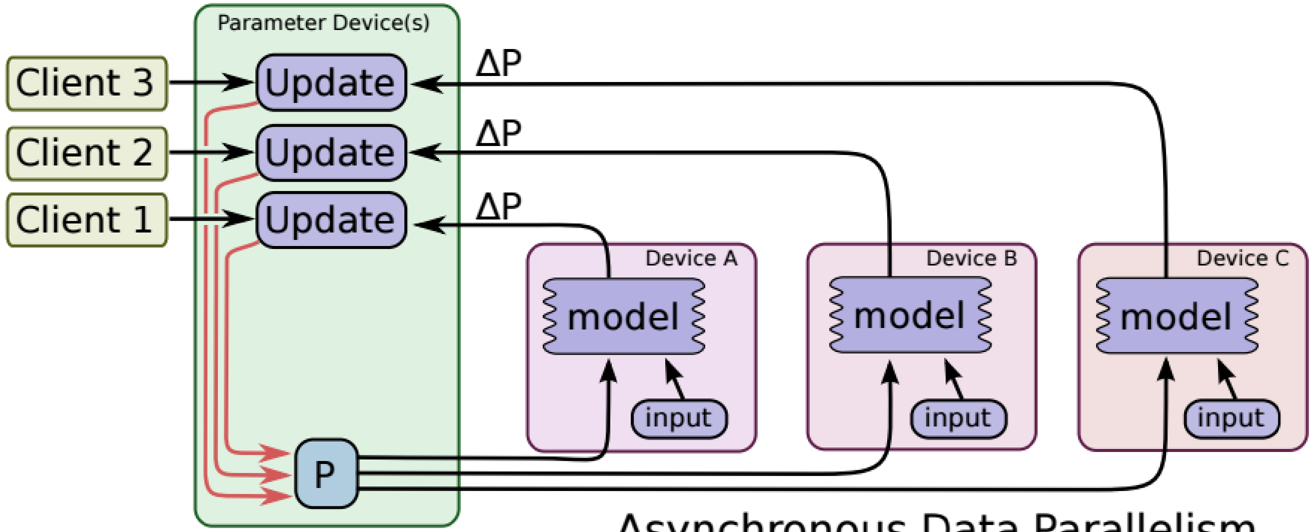
- One machine with one GPU
- One machine with many GPUs
- Multiple machines with multiple GPUs
  - Synchronous training:
    - all the workers will read the parameters at the same time, compute a training operation and wait for all the others to be done. Then the gradients will be averaged and a single update will be sent to the parameter server. So at any point in time, the workers will all be aware of the same values for the graph parameters
  - Asynchronous training:
    - the workers will read from the parameter server(s) asynchronously, compute their training operation, and send asynchronous updates. At any point in time, two different workers might be aware of different values for the graph parameters

Distributed Training(cont.)



Synchronous Data Parallelism

Recommend if possible



Asynchronous Data Parallelism

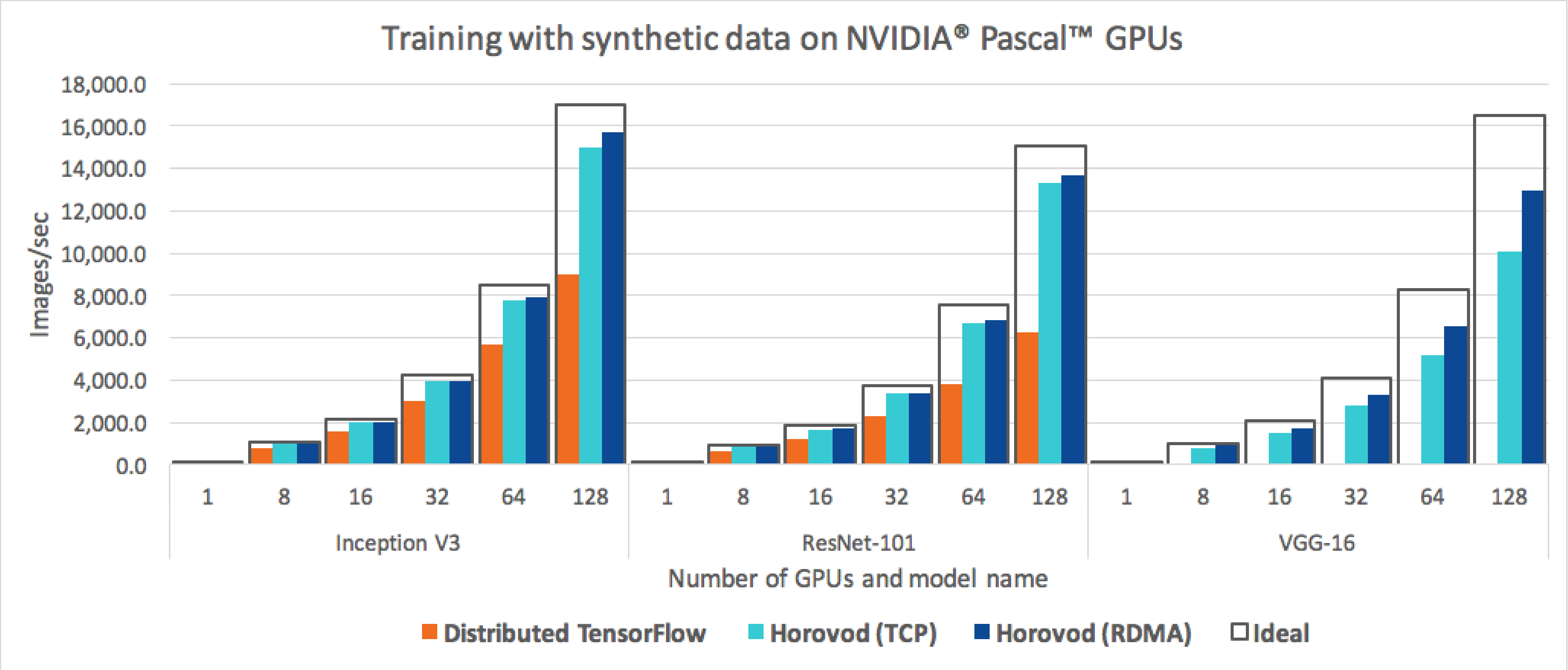
Distributed Deep Learning Frameworks

- Caffe-MPI: A parallel Framework on the GPU Clusters
  - Inspur(浪潮)
  - Only support 16 GPUs

Network	Framework	Speed (# of Samples per second)				
		1 GPU	2 GPUs	4 GPUs	8 GPUs	16 GPUs
AlexNet	Caffe-MPI	1800	3602	6948	14283	26371
	CNTK	1423	1988	3332	6517	12574
	MXNet	1386	2711	3238	5759	7939
	TensorFlow	1543	2941	3689	7102	12511
GoogleNet	Caffe-MPI	413	820	1539	3151	5886
	CNTK	453	792	1457	2469	4894
	MXNet	425	822	1588	2824	4470
	TensorFlow	397	732	1384	2639	4814
ResNet-50	Caffe-MPI	142	276	557	1098	2127
	CNTK	134	251	457	868	1666
	MXNet	133	265	513	720	1118
	TensorFlow	134	260	490	575	905

Distributed Deep Learning Frameworks

- Horovod: Distributed training framework for TensorFlow
  - Fast and easy to use



Distributed TensorFlow

Multi-GPUs Training

Distributed Training

Multi-GPUs

- TF maps nearly all of the GPU memory of all GPUs by default
  - Two option methods

```
# method 1
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
session = tf.Session(config=config, ...)
# method 2
config = tf.ConfigProto()
config.gpu_options.per_process_gpu_memory_fraction = 0.4
session = tf.Session(config=config, ...)
```

  - If you have more than one GPUs, run this cmd:

```
-CUDA_VISIBLE_DEVICES=1 python my_script.py
```

Multi-GPUs(cont.)

- Manual device placement

```
# Creates a graph.
a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3], name='a')
b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2], name='b')
c = tf.matmul(a, b)
# Creates a session with log_device_placement set to True.
sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
# Runs the op.
print(sess.run(c))
```

You should see the following output:

```
Device mapping:
/job:localhost/replica:0/task:0/device:GPU:0 -> device: 0, name: Tesla K40c, pci bus
id: 0000:05:00.0
b: /job:localhost/replica:0/task:0/device:GPU:0
a: /job:localhost/replica:0/task:0/device:GPU:0
MatMul: /job:localhost/replica:0/task:0/device:GPU:0
[[ 22.  28.]
 [ 49.  64.]]
```

Multi-GPUs(cont.)

- Manual device placement

```
# Creates a graph.
with tf.device('/cpu:0'):
  a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3], name='a')
  b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2], name='b')
c = tf.matmul(a, b)
# Creates a session with log_device_placement set to True.
sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
# Runs the op.
print(sess.run(c))
```

```
Device mapping:
/job:localhost/replica:0/task:0/device:GPU:0 -> device: 0, name: Tesla K40c, pci bus
id: 0000:05:00.0
b: /job:localhost/replica:0/task:0/cpu:0
a: /job:localhost/replica:0/task:0/cpu:0
MatMul: /job:localhost/replica:0/task:0/device:GPU:0
[[ 22.  28.]
 [ 49.  64.]]
```

- Using Multi-GPUs

```
# Creates a graph.
c = []
for d in ['/device:GPU:2', '/device:GPU:3']:
  with tf.device(d):
    a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3])
    b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2])
    c.append(tf.matmul(a, b))
with tf.device('/cpu:0'):
  sum = tf.add_n(c)
# Creates a session with log_device_placement set to True.
sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
# Runs the op.
print(sess.run(sum))
```

You will see the following output.

```
Device mapping:
/job:localhost/replica:0/task:0/device:GPU:0 -> device: 0, name: Tesla K20m, pci bus
id: 0000:02:00.0
/job:localhost/replica:0/task:0/device:GPU:1 -> device: 1, name: Tesla K20m, pci bus
id: 0000:03:00.0
/job:localhost/replica:0/task:0/device:GPU:2 -> device: 2, name: Tesla K20m, pci bus
id: 0000:83:00.0
/job:localhost/replica:0/task:0/device:GPU:3 -> device: 3, name: Tesla K20m, pci bus
id: 0000:84:00.0
Const_3: /job:localhost/replica:0/task:0/device:GPU:3
Const_2: /job:localhost/replica:0/task:0/device:GPU:3
MatMul_1: /job:localhost/replica:0/task:0/device:GPU:3
Const_1: /job:localhost/replica:0/task:0/device:GPU:2
Const: /job:localhost/replica:0/task:0/device:GPU:2
MatMul: /job:localhost/replica:0/task:0/device:GPU:2
AddN: /job:localhost/replica:0/task:0/cpu:0
[[  44.   56.]
 [  98.  128.]]
```

Training Model on Multiple GPU cards

- Place an individual model replica on each GPU.
- Update model parameters synchronously by waiting for all GPUs to finish processing a batch of data.
- See [cifar10_multi_gpu_train.py]()

- One question: iteration or batch size?
  - Epoch 100
  - Iteration 400
  - Batch size 256
  - Assume we have 8 GPUs, Batch size / 8 or Iteration / 8??

- Training on Multi-GPUs(TensorFlow)

```
with tf.device('/cpu:0'):
  tower_grads = []
  reuse_vars = False

  with tf.name_scope('input'):
    x  = tf.placeholder(tf.float32,[None, image_size, image_size, 3], name='input_x')
    y_ = tf.placeholder(tf.float32, [None, class_num], name='input_y')
  learning_rate = tf.placeholder(tf.float32)

  # cal gradient on each GPU
  for i in range(FLAGS.gpu_number):
    with tf.device('/gpu:%d' % i):
      x_split = x[i * FLAGS.batch_size: (i+1) * FLAGS.batch_size]
      y_split = y_[i * FLAGS.batch_size: (i+1) * FLAGS.batch_size]

      logits_train = inference(x_split,reuse=reuse_vars)
      loss, l2 = cal_loss(logits_train,y_split)
      optimizer = tf.train.MomentumOptimizer(learning_rate,
            FLAGS.momentum,use_nesterov=True)
      grads = optimizer.compute_gradients(loss + l2 * FLAGS.weight_decay)

      reuse_vars = True
      tower_grads.append(grads)

  # average gradients
  tower_grads = average_gradients(tower_grads)
  train_op = optimizer.apply_gradients(tower_grads)
```

• Training on Multi-GPUs(Keras)

```python
def slice_batch(x, n_gpus, part):
    sh = K.shape(x)
    L =  sh[0] // n_gpus
    if part == n_gpus - 1:
        return x[part*L:]
    return x[part*L:(part+1)*L]


def to_multi_gpu(model, n_gpus=2):
    if n_gpus ==1:
        return model

    with tf.device('/cpu:0'):
        x = Input(model.input_shape[1:])
    towers = []
    for g in range(n_gpus):
        with tf.device('/gpu:' + str(g)):
            slice_g = Lambda(slice_batch, lambda shape: shape,
                             arguments= {'n_gpus':n_gpus, 'part':g})(x)
            towers.append(model(slice_g))

    with tf.device('/cpu:0'):
        merged = Concatenate(axis=0)(towers)
    return Model(inputs=[x], outputs=merged)
```

```python
model      = Model(img_input, output)

# -------------- Multi-GPU------------------#
model      = to_multi_gpu(model,n_gpus=gpu_number)
# -------------- Multi-GPU------------------#
```

- Training on Multi-GPUs(Keras)
  - DenseNet-160x24 See densenet_multi_gpu.py
  - Use 2 GTX 1080
  - Batch Size 64(32 each GPU)
  - Training Time: **50 h 20 min**
  - Accuracy: **95.90%**

```python
def slice_batch(x, n_gpus, part):
    sh = K.shape(x)
    L =  sh[0] // n_gpus
    if part == n_gpus - 1:
        return x[part*L:]
    return x[part*L:(part+1)*L]


def to_multi_gpu(model, n_gpus=2):
    if n_gpus ==1:
        return model

    with tf.device('/cpu:0'):
        x = Input(model.input_shape[1:])
    towers = []
    for g in range(n_gpus):
        with tf.device('/gpu:' + str(g)):
            slice_g = Lambda(slice_batch, lambda shape: shape,
                             arguments= {'n_gpus':n_gpus, 'part':g})(x)
            towers.append(model(slice_g))

    with tf.device('/cpu:0'):
        merged = Concatenate(axis=0)(towers)
    return Model(inputs=[x], outputs=merged)
```

```python
model     = Model(img_input, output)

# ------------- Multi-GPU-----------------#
model     = to_multi_gpu(model,n_gpus=gpu_number)
# ------------- Multi-GPU-----------------#
```

- Really??
  - Keras has a built-in utility, which can produce a data-parallel version of any model, and achieves quasi-linear speedup on up to 8 GPUs. (wow!)

```python
from keras.utils import multi_gpu_model

with tf.device('/cpu:0'):
    model = Xception(weights=None,
                     input_shape=(height, width, 3),
                     classes=num_classes)

parallel_model = multi_gpu_model(model, gpus=8)
```

Distributed Training(TensorFlow)

- Distributed TensorFlow
- CIFAR10-distribute-latest
- Running Distributed TensorFlow Example via Docker
- DISTRIBUTED TENSORFLOW EXAMPLE

Distributed Training(TensorFlow)

- Cluster: A TensorFlow cluster comprises a one or more "jobs"
- Job: A job comprises a list of "tasks"
  - Parameter Server(ps)
    - a job named ps typically hosts nodes that store and update variables
    - **need to kill the process after training**
  - Worker(worker)
    - a job named worker typically hosts stateless nodes that perform compute-intensive tasks.

Distributed Training(TensorFlow)

```
# On 192.168.2.241:
$ python trainer.py \
     --ps_hosts=192.168.2.241:2222,192.168.2.242:2222 \
     --worker_hosts=192.168.2.243:2222,192.168.2.244:2222,192.168.2.245:2222, \
     --job_name=ps --task_index=0
# On 192.168.2.242:
$ python trainer.py \
     --ps_hosts=192.168.2.241:2222,192.168.2.242:2222 \
     --worker_hosts=192.168.2.243:2222,192.168.2.244:2222,192.168.2.245:2222, \
     --job_name=ps --task_index=1
# On 192.168.2.243:
$ python trainer.py \
     --ps_hosts=192.168.2.241:2222,192.168.2.242:2222 \
     --worker_hosts=192.168.2.243:2222,192.168.2.244:2222,192.168.2.245:2222, \
     --job_name=worker --task_index=0
# On 192.168.2.244:
$ python trainer.py \
     --ps_hosts=192.168.2.241:2222,192.168.2.242:2222 \
     --worker_hosts=192.168.2.243:2222,192.168.2.244:2222,192.168.2.245:2222, \
     --job_name=worker --task_index=1
# On 192.168.2.245:
$ python trainer.py \
     --ps_hosts=192.168.2.241:2222,192.168.2.242:2222 \
     --worker_hosts=192.168.2.243:2222,192.168.2.244:2222,192.168.2.245:2222, \
     --job_name=worker --task_index=2
```

Distributed Training(Horovod)

- Horovod: Distributed training framework for TensorFlow
  - Fast and easy to use
  - Support TensorFlow and Keras
- Installation:
  - Install Open MPI
  - Install NCCL 2(opt: RDMA and GPUDirect)
  - Install Horovod

• TensorFlow Implement

```python
def main(_):
    # Initialize Horovod.
    hvd.init()

    mnist = learn.datasets.mnist.read_data_sets('MNIST-data-%d' % hvd.rank())

    # Build model...
    with tf.name_scope('input'):
        image = tf.placeholder(tf.float32, [None, 784], name='image')
        label = tf.placeholder(tf.float32, [None], name='label')
    predict, loss = conv_model(image, label, tf.contrib.learn.ModeKeys.TRAIN)

    opt = tf.train.RMSPropOptimizer(0.01)

    # Add Horovod Distributed Optimizer.
    opt = hvd.DistributedOptimizer(opt)

    global_step = tf.contrib.framework.get_or_create_global_step()
    train_op = opt.minimize(loss, global_step=global_step)

    # Pin GPU to be used to process local rank (one GPU per process)
    config = tf.ConfigProto()
    config.gpu_options.allow_growth = True
    config.gpu_options.visible_device_list = str(hvd.local_rank())

    checkpoint_dir = './checkpoints' if hvd.rank() == 0 else None

    with tf.train.MonitoredTrainingSession(checkpoint_dir=checkpoint_dir,
                                           hooks=hooks,
                                           config=config) as mon_sess:
        while not mon_sess.should_stop():
            # Run a training step synchronously.
            image_, label_ = mnist.train.next_batch(100)
            mon_sess.run(train_op, feed_dict={image: image_, label: label_})
```

- Keras Implement

```python
# Initialize Horovod.
hvd.init()

# Pin GPU to be used to process local rank (one GPU per process)
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
config.gpu_options.visible_device_list = str(hvd.local_rank())
K.set_session(tf.Session(config=config))
```

```python
# set iteration or batch size depends on GPUs
iterations          = 50000 // batch_size // hvd.size()
# or batch_size = batch_size // hvd.size()
```

```python
# set optimizer
sgd = optimizers.SGD(lr=.1, momentum=0.9, nesterov=True)
sgd = hvd.DistributedOptimizer(sgd)
```

Distributed Training(Horovod)

- Usage:
  - Put the file in the same directory
  - Then run the following cmd(only need to run it on one machine)

```
mpirun -np 4 \
    -H 192.168.2.241:1,192.168.2.242:1,192.168.2.243:1,192.168.2.244:1  \
    -bind-to none -map-by slot \
    -x NCCL_DEBUG=INFO -x LD_LIBRARY_PATH \
    python train.py
```

- My test:
  - Residual Network(110 layers) for CIFAR-10:
    - Single GPU: **270 min(4 h 30 min)**
    - Distributed by Horovod(4 machines): **81min(1 h 21min)**

Thanks for your attention

Appendix1: Params for VGG16

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000

Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0

Appendix2: FLOPs for CIFAR10

Feature map Size		FLOPs	
1x1x10	fc,1000	1*1*4096*10*1*1	= 40960
1x1x4096	fc,4096	1*1*4096*4096*1*1	= 16777216
1x1x4096	fc,4096	1*1*512*4096*1*1	= 2097152
1x1x512	pooling		
2x2x512	3x3 conv,512	2*2*512*512*3*3	= 9437184
2x2x512	3x3 conv,512	2*2*512*512*3*3	= 9437184
2x2x512	3x3 conv,512	2*2*512*512*3*3	= 9437184
2x2x512	pooling		
4x4x512	3x3 conv,512	4*4*512*512*3*3	= 37748736
4x4x512	3x3 conv,512	4*4*512*512*3*3	= 37748736
4x4x512	3x3 conv,512	4*4*256*512*3*3	= 18874368
4x4x256	pooling		
8x8x256	3x3 conv,256	8*8*256*256*3*3	= 37748736
8x8x256	3x3 conv,256	8*8*256*256*3*3	= 37748736
8x8x256	3x3 conv,256	8*8*128*256*3*3	= 18874368
8x8x128	pooling		
16x16x128	3x3 conv,128	16*16*128*128*3*3	= 37748736
16x16x128	3x3 conv,128	16*16*64*128*3*3	= 18874368
16x16x64	pooling		
32x32x64	3x3 conv,64	32*32*64*64*3*3	= 37748736
32x32x64	3x3 conv,64	32*32*3*64*3*3	= 1769472
32x32x3	input		

- Number of FLOPs :
  - 332111872
  - About **0.332 GFLOPS**