Deep Learning Tools

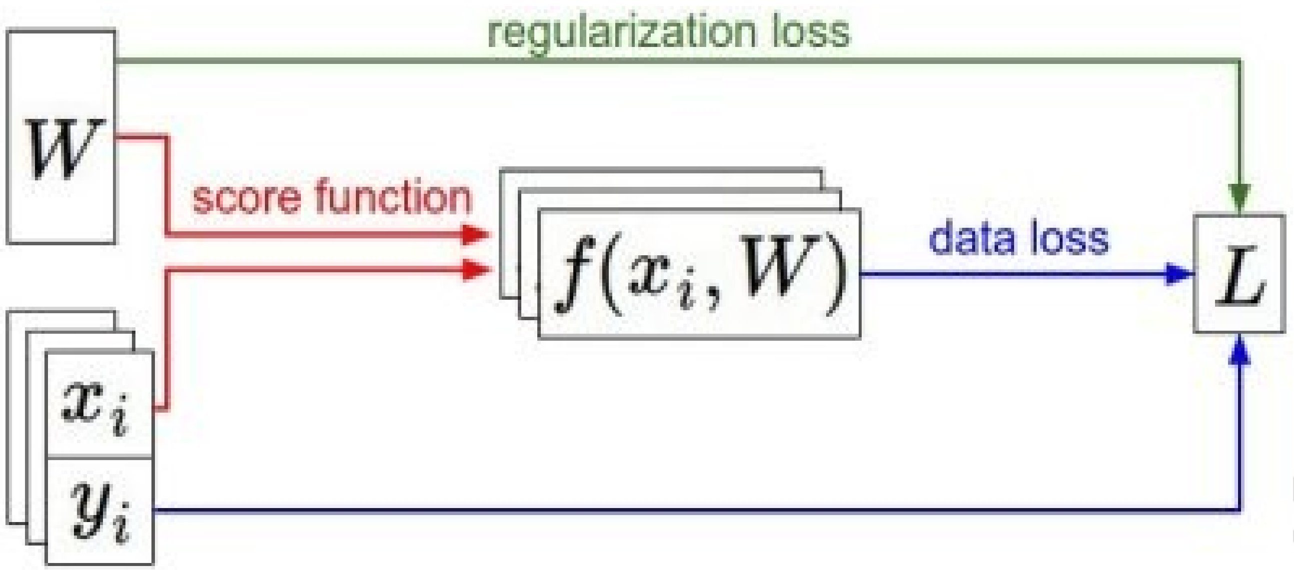
·AIPKU·

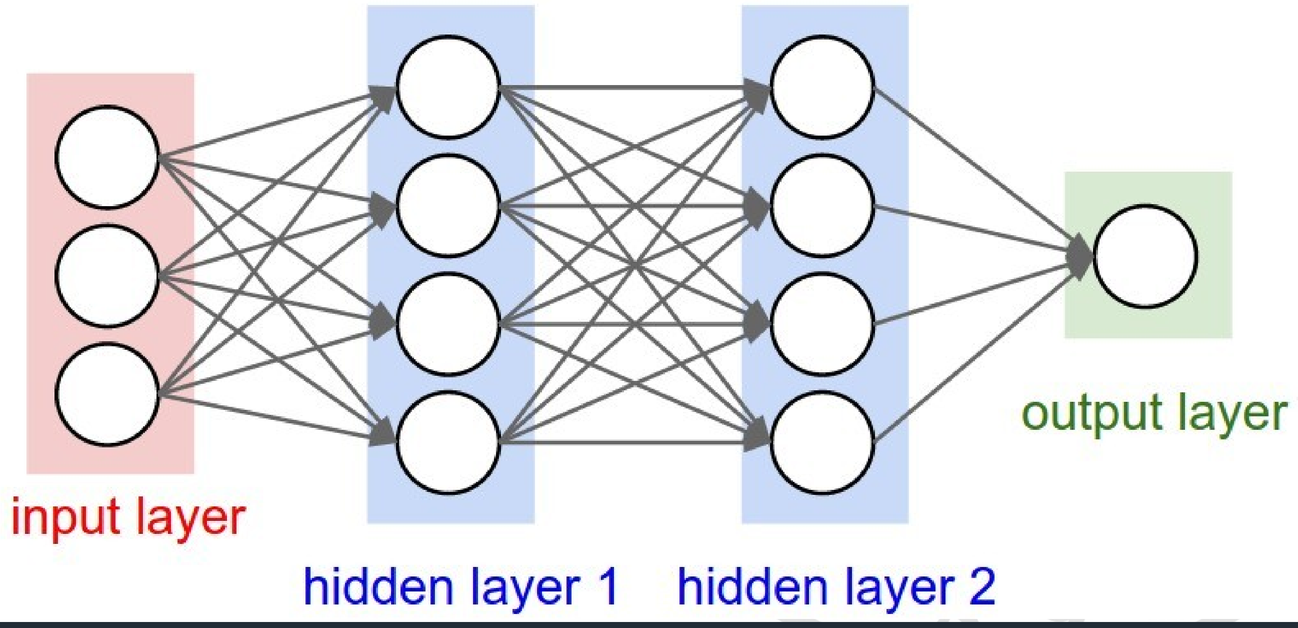
人工智能引论实践课 计算机视觉小班

主讲人：胡越予

Neural Networks

- Training a multi-layer network
  - Data Preparation
  - Feed-Forward
  - Loss Function
  - Back-Propagation
  - Gradient Descent





Deep Learning Framework

Caffe
(UC Berkeley)
→
Caffe2
(Facebook)

Torch
(NYU / Facebook)
→
PyTorch
(Facebook)

Theano
(U Montreal)
→
TensorFlow
(Google)

MXNet
(Amazon)
Developed by U Washington, CMU, MIT, Hong Kong U, etc but main framework of choice at AWS

CNTK
(Microsoft)

Deeplearning4j

And others...

Software & Hardware

PyTorch

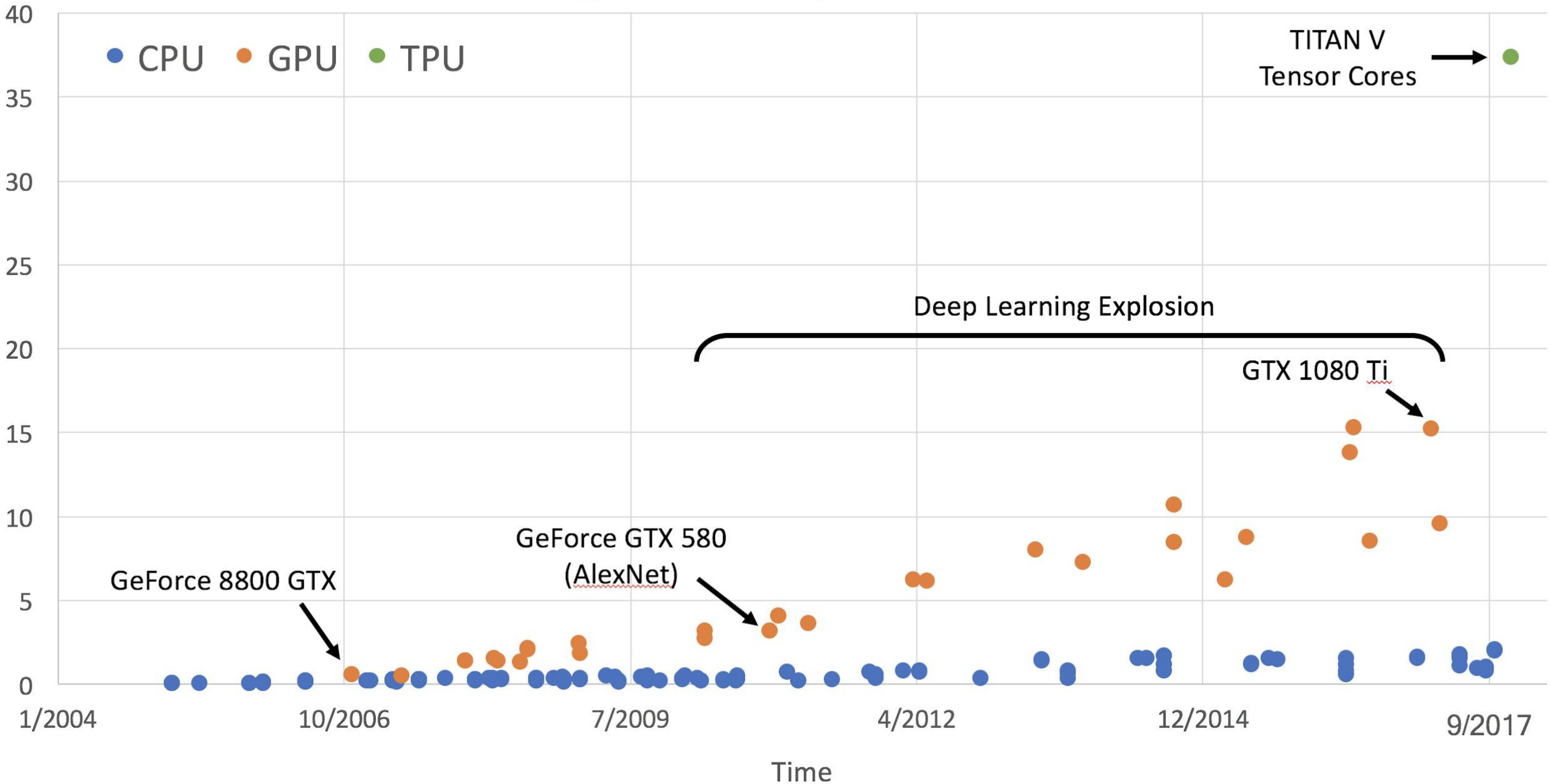
TensorFlow

NVIDIA CUDA

GPU and CUDA

- GFLOPs per Dollar

Source: http://cs231n.stanford.edu/2018/



TensorFlow

Deep Learning library by Google

导入TF: import tensorflow as tf

tf.keras: 一个简单的模块化的深度学习API

```
from tf.keras import layers


model = tf.keras.Sequential() # 序贯模型，不断地向模型中添加层
# Adds a densely-connected layer with 64 units to the model:
model.add(layers.Dense(64, activation='relu'))
# Add another:
model.add(layers.Dense(64, activation='relu'))
# Add a softmax layer with 10 output units:
model.add(layers.Dense(10, activation='softmax'))
```

常用配置

```
# Create a sigmoid layer:
layers.Dense(64, activation='sigmoid')
# Or:
layers.Dense(64, activation=tf.sigmoid)

# 对 Weights 使用L1正则化
layers.Dense(64, kernel_regularizer=tf.keras.regularizers.l1(0.01))

# 对 bias 使用L1正则化
layers.Dense(64, bias_regularizer=tf.keras.regularizers.l2(0.01))

# 使用正交初始化
layers.Dense(64, kernel_initializer='orthogonal')

# 使用常数初始化
layers.Dense(64, bias_initializer=tf.keras.initializers.constant(2.0))
```

Training

```python
model = tf.keras.Sequential([
# Adds a densely-connected layer with 64 units to the model:
layers.Dense(64, activation='relu'),
# Add another:
layers.Dense(64, activation='relu'),
# Add a softmax layer with 10 output units:
layers.Dense(10, activation='softmax')])

model.compile(optimizer=tf.keras.optimizers.SGD(0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Optimizer & Loss Function

- tf.keras.Model.compile 采用三个重要参数:
- optimizer：此对象会指定训练过程。从 tf.train 模块向其传递优化器实例，例如 tf.train.AdamOptimizer、tf.train.RMSPropOptimizer 或 tf.train.GradientDescentOptimizer。
- loss：要在优化期间最小化的函数。常见选择包括均方误差 (mse)、categorical_crossentropy 和 binary_crossentropy。损失函数由名称或通过从 tf.keras.losses 模块传递可调用对象来指定。
- metrics：用于监控训练。它们是 tf.keras.metrics 模块中的字符串名称或可调用对象。

```
# Configure a model for mean-squared error regression.
model.compile(optimizer=tf.train.AdamOptimizer(0.01),
              loss='mse',       # mean squared error
              metrics=['mae'])  # mean absolute error
```

自动求导

- TF 和 PyTorch 均支持自动求导，反向传播
- 讨论：如何实现自动求导?

Training

```
import numpy as np

data = np.random.random((1000, 32))
labels = np.random.random((1000, 10))

model.fit(data, labels, epochs=10, batch_size=32)
```

```
Epoch 1/10
1000/1000 [==============================] - 0s 253us/step - loss: 11.5766 - categorical_accuracy: 0.1110
Epoch 2/10
1000/1000 [==============================] - 0s 64us/step - loss: 11.5205 - categorical_accuracy: 0.1070
Epoch 3/10
1000/1000 [==============================] - 0s 70us/step - loss: 11.5146 - categorical_accuracy: 0.1100
Epoch 4/10
1000/1000 [==============================] - 0s 69us/step - loss: 11.5070 - categorical_accuracy: 0.0940
```

函数式API

- tf.keras.Sequential 模型是层的简单堆叠，无法表示任意模型。使用 Keras 函数式 API 可以构建复杂的模型拓扑
  - 多输入模型，
  - 多输出模型，
  - 具有共享层的模型（同一层被调用多次），
  - 具有非序列数据流的模型（例如，剩余连接）。
- 使用函数式 API 构建的模型具有以下特征：
  - 层实例可调用并返回张量。
  - 输入张量和输出张量用于定义 tf.keras.Model 实例。
  - 此模型的训练方式和 Sequential 模型一样。

函数式API

```python
inputs = tf.keras.Input(shape=(32,))  # Returns a placeholder tensor

# A layer instance is callable on a tensor, and returns a tensor.
x = layers.Dense(64, activation='relu')(inputs)
x = layers.Dense(64, activation='relu')(x)
predictions = layers.Dense(10, activation='softmax')(x)

# 在给定输入和输出的情况下实例化模型。
model = tf.keras.Model(inputs=inputs, outputs=predictions)

# The compile step specifies the training configuration.
model.compile(optimizer=tf.train.RMSPropOptimizer(0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Trains for 5 epochs
model.fit(data, labels, batch_size=32, epochs=5)
```

保存和恢复权重

```
model = tf.keras.Sequential([
layers.Dense(64, activation='relu'),
layers.Dense(10, activation='softmax')])

model.compile(optimizer=tf.train.AdamOptimizer(0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])


# Save weights to a TensorFlow Checkpoint file
model.save_weights('./weights/my_model')

# Restore the model's state,
# this requires a model with the same architecture.
model.load_weights('./weights/my_model')
```

Low-Level API

```
x = tf.placeholder(tf.float32, [None, 784])
y_ = tf.placeholder(tf.int64, [None])

y_conv = deepnn(x)

cross_entropy =
   tf.losses.sparse_softmax_cross_entropy(labels=y_, logits=y_conv)
cross_entropy = tf.reduce_mean(cross_entropy)
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
correct_prediction = tf.equal(tf.argmax(y_conv, 1), y_)
correct_prediction = tf.cast(correct_prediction, tf.float32)
accuracy = tf.reduce_mean(correct_prediction)
```

Low-Level API

```
def deepnn(x):
x_image = tf.reshape(x, [-1, 28, 28, 1])

# First convolutional layer - maps one grayscale image to 32 feature maps.
with tf.name_scope('conv1'):
    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)

# Pooling layer - downsamples by 2X.
with tf.name_scope('pool1'):
    h_pool1 = max_pool_2x2(h_conv1)

# Second convolutional layer -- maps 32 feature maps to 64.
with tf.name_scope('conv2'):
    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)

.............

with tf.name_scope('fc2'):
    W_fc2 = weight_variable([1024, 10])
    b_fc2 = bias_variable([10])

y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2

return y_conv
```

Low-Level API

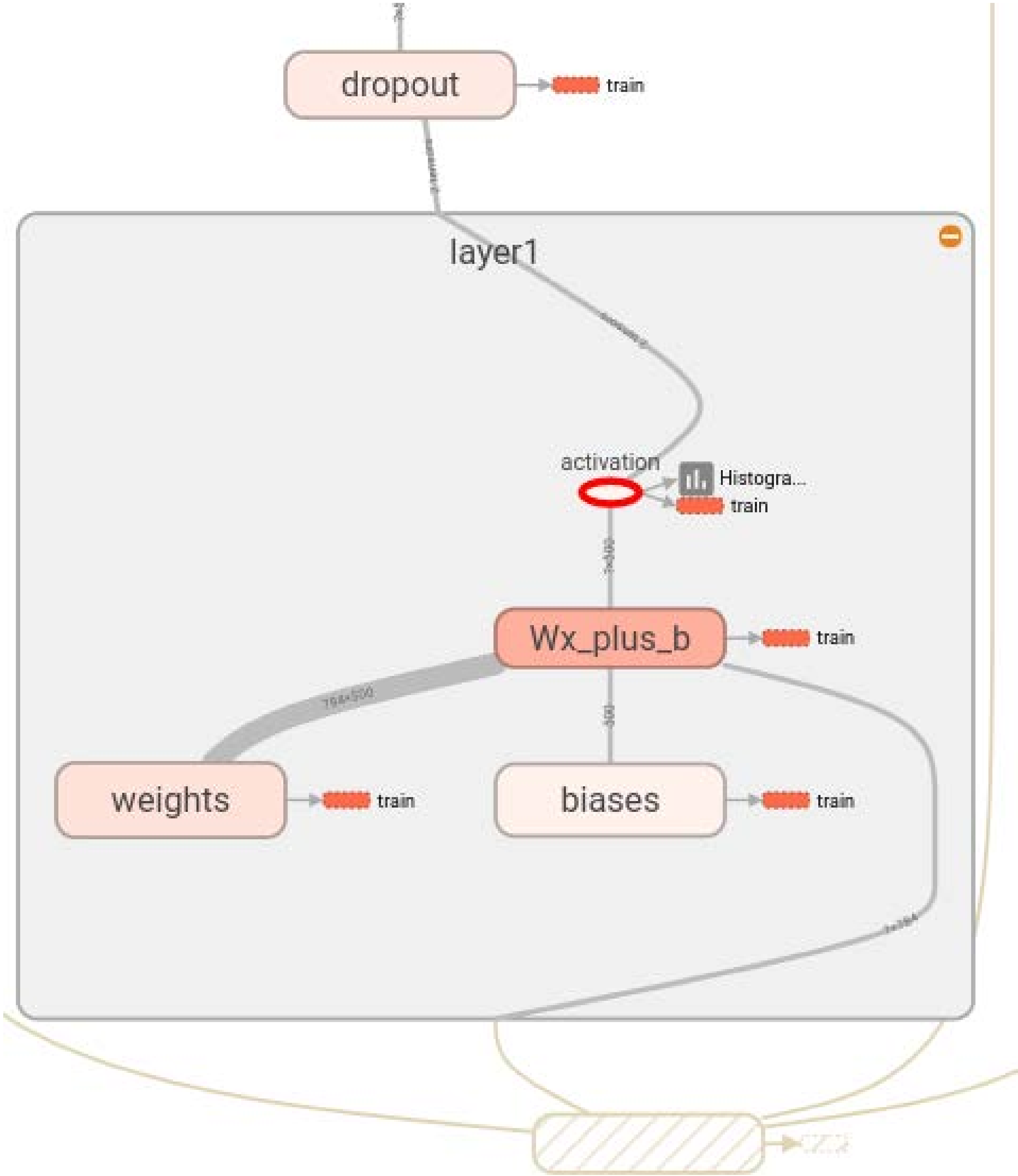
```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(20000):
        batch = mnist.train.next_batch(50)
        sess.run(train_step, feed_dict={x: batch[0], y_:batch[1]})
        if i % 100 == 0:
            train_accuracy =
                accuracy.eval(feed_dict={x: batch[0], y_: batch[1]})
            print('step %d, training accuracy %g' % (i, train_accuracy))
```

TensorFlow 静态图编程模型

- 所有的Tensor和op组成图
  - Tensor 和 op 各为节点
  - 各个运算表达式不实际执行运算，而是对图进行编辑
  - 需要用 Placeholder 描述输入节点
  - 真正的执行发生在 sess.run
  - 在 sess.run 的时候，运算图才进行“编译”
  - * 不支持对图的动态修改以及在 Python 上调试
    - 在 Python 代码中无法得知变量的值
    - 新版本融合了动态图特性（ Eager Execution ）

TensorFlow Graph

- Computation is modeled using a graph
- Most of the time the graph is static



使用 GPU

- TensorFlow 默认占用**所有 GPU 的所有显存**
  - 其实根本用不上
- 在命令行中使用环境变量 CUDA_VISIBLE_DEVICES 控制
  - CUDA_VISIBLE_DEVICES=0 python example.py
    程序运行时，TF只能使用系统的第 0 块卡
  - CUDA_VISIBLE_DEVICES=0,1 python example.py
    多卡分配

GPU 显存分配

在某些情况下，最理想的是进程只分配可用内存的一个子集，或者仅根据进程需要增加内存使用量。 TensorFlow 在 Session 上提供两个 Config 选项来进行控制。

第一个是 allow_growth 选项，它试图根据运行时的需要来分配 GPU 内存：它刚开始分配很少的内存，随着 Session 开始运行并需要更多 GPU 内存，我们会扩展 TensorFlow 进程所需的 GPU 内存区域。请注意，我们不会释放内存，因为这可能导致出现更严重的内存碎片情况。要开启此选项，请通过以下方式在 ConfigProto 中设置选项：

```
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
session = tf.Session(config=config, ...)
```

第二个是 per_process_gpu_memory_fraction 选项，它可以决定每个可见 GPU 应分配到的内存占总内存量的比例。例如，您可以通过以下方式指定 TensorFlow 仅分配每个 GPU 总内存的 40%：

```
config = tf.ConfigProto()
config.gpu_options.per_process_gpu_memory_fraction = 0.4
session = tf.Session(config=config, ...)
```

Complicated API

Keras (https://keras.io/)

tf.keras (https://www.tensorflow.org/api_docs/python/tf/keras)

tf.layers (https://www.tensorflow.org/api_docs/python/tf/layers)

tf.estimator (https://www.tensorflow.org/api_docs/python/tf/estimator)

tf.contrib.estimator (https://www.tensorflow.org/api_docs/python/tf/contrib/estimator)

tf.contrib.layers (https://www.tensorflow.org/api_docs/python/tf/contrib/layers)

tf.contrib.slim (https://github.com/tensorflow/tensorflow/tree/master/tensorflow/contrib/slim)

Ships with TensorFlow

Sonnet (https://github.com/deepmind/sonnet)

By DeepMind

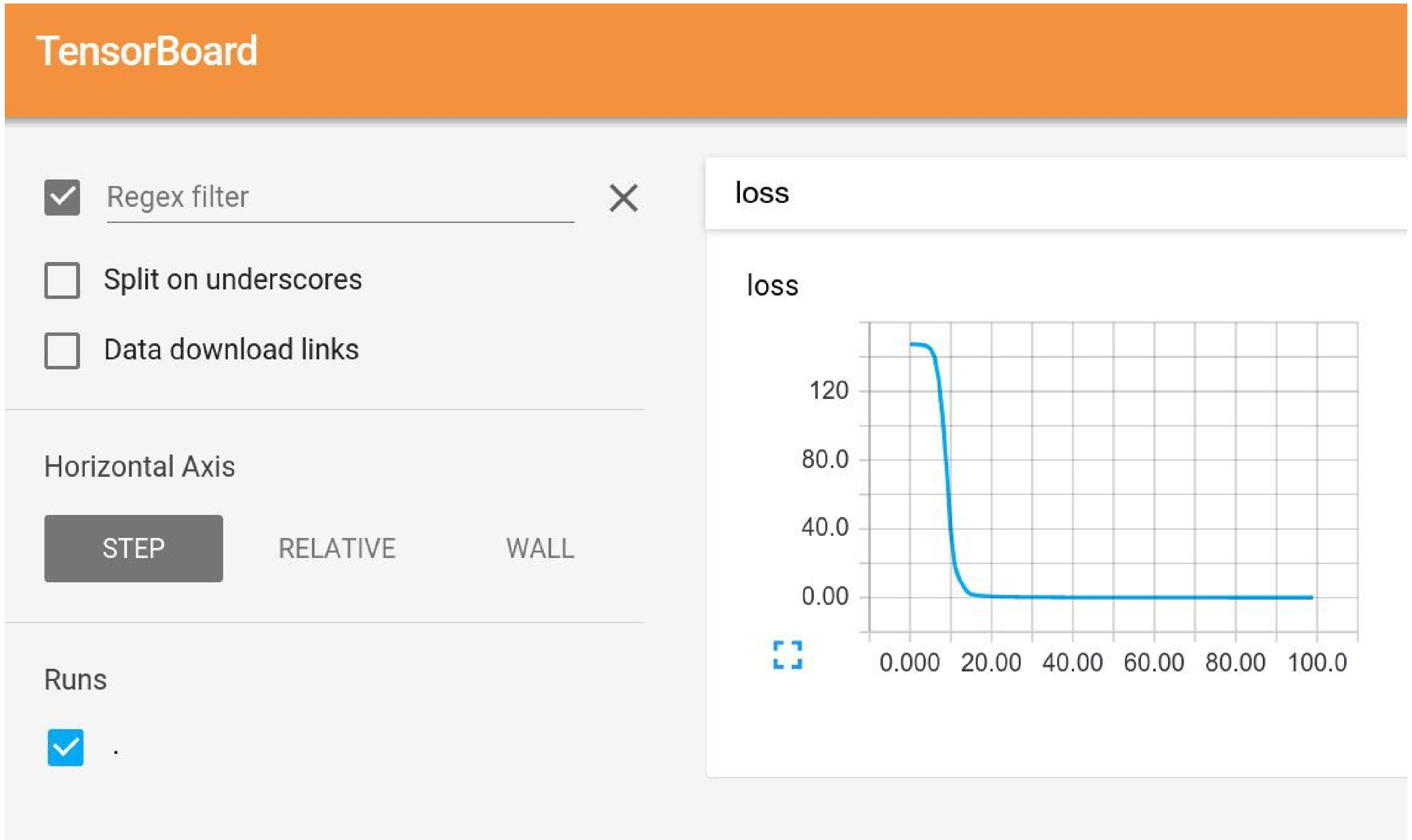
TFLearn (http://tflearn.org/)

TensorLayer (http://tensorlayer.readthedocs.io/en/latest/)

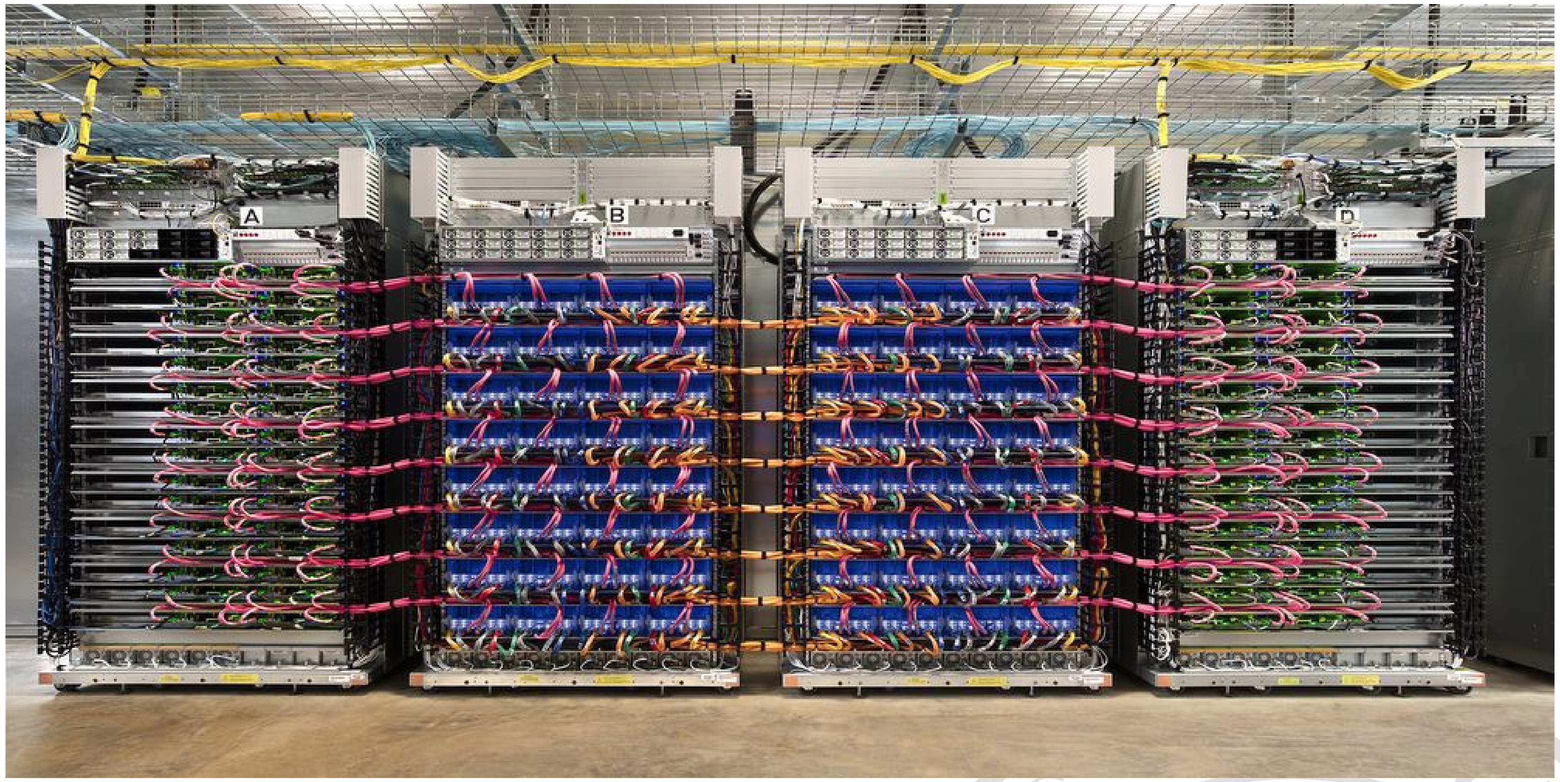
Third-Party

Tensorboard

- Add logging to code to record loss, stats, etc Run server and get pretty graphs!



TensorFlow TPU



Google Cloud TPU

= 180 TFLOPs of compute!

Google Cloud TPU Pod

= 64 Cloud TPUs

= 11.5 PFLOPs of compute!

TensorFlow 学习资料

- 官方 Tutorial
  - https://www.tensorflow.org/tutorials
  - 因为官方要推广新特性，所以有些会与历史代码不兼容
  - 推荐按 Keras API 来编写程序
- TensorFlow 安装教程：
- https://www.tensorflow.org/install?hl=zh_cn

PyTorch

- 动态图编程模型
  - 可以随时查看变量的结果
  - 可以动态灵活控制使用的设备（CPU，GPU）

```python
import torch
x = torch.ones(5, 3)
y = torch.eye((5, 5))
print(y.matmul(x)) # all 1. tensor with shape (5,5)
```

```python
if torch.cuda.is_available():
   device = torch.device("cuda") # a CUDA device object
   y = torch.ones_like(x, device=device) # directly create a tensor on GPU
   x = x.to(device) # or just use strings ``.to("cuda")``
   z = x + y
   print(z)
   print(z.to("cpu", torch.double))
```

Network

```python
class Net(nn.Module):

def __init__(self):
     super(Net, self).__init__()
     # 1 input image channel, 6 output channels, 5x5 square convolution
     # kernel
     self.conv1 = nn.Conv2d(1, 6, 5)
     self.conv2 = nn.Conv2d(6, 16, 5)
     # an affine operation: y = Wx + b
     self.fc1 = nn.Linear(16 * 5 * 5, 120)
     self.fc2 = nn.Linear(120, 84)
     self.fc3 = nn.Linear(84, 10)

def forward(self, x):
     # Max pooling over a (2, 2) window
     x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
     # If the size is a square you can only specify a single number
     x = F.max_pool2d(F.relu(self.conv2(x)), 2)
     x = x.view(-1, self.num_flat_features(x))
     x = F.relu(self.fc1(x))
     x = F.relu(self.fc2(x))
     x = self.fc3(x)
     return x
```

Training Loop

```
import torch.optim as optim
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

for epoch in range(2): # loop over the dataset multiple times
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs
        inputs, labels = data
        # zero the parameter gradients
        optimizer.zero_grad()
        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        # print statistics
        running_loss += loss.item()

        if i % 2000 == 1999: # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
            (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0
```

Testing

```
correct = 0
total = 0
with torch.no_grad():
   for data in testloader:
      images, labels = data
      outputs = net(images)
      _, predicted = torch.max(outputs.data, 1)
      total += labels.size(0)
      correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 10000 test images: %d %%' % (
100 * correct / total))
```

Using GPU

- `net.to(device)`
- 对于所有的 Training Iteration

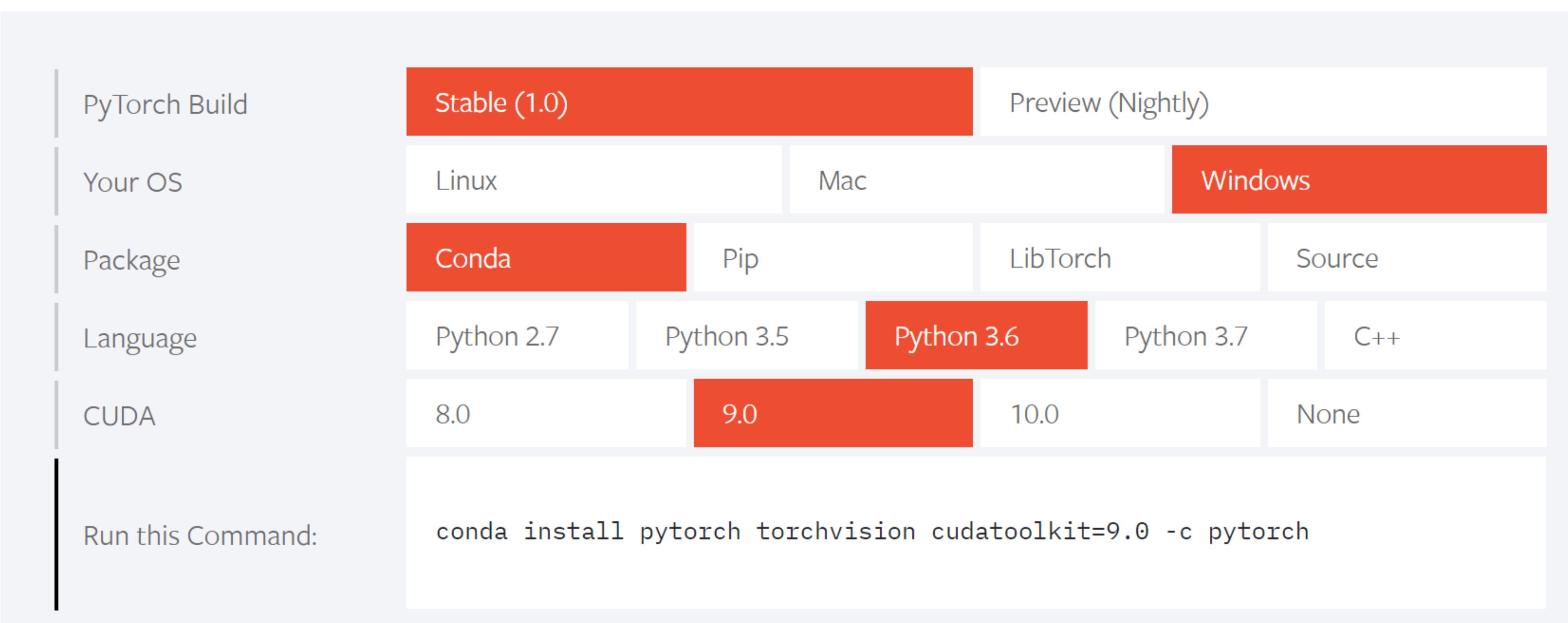
```
inputs, labels = inputs.to(device), labels.to(device)
```

PyTorch 特性

- 所有的运算都是立即进行的
- 需要显式的设备分配，设备间传输
- 官方 Tutorial

https://pytorch.org/tutorials/

- 安装教程：https://pytorch.org/



Static vs Dynamic: Conditional

$$y = \begin{cases} w1 * x & \text{if } z > 0 \\ w2 * x & \text{otherwise} \end{cases}$$

PyTorch: Normal Python

```
N, D, H = 3, 4, 5

x = torch.randn(N, D, requires_grad=True)
w1 = torch.randn(D, H)
w2 = torch.randn(D, H)

z = 10
if z > 0:
    y = x.mm(w1)
else:
    y = x.mm(w2)
```

TensorFlow: Special TF control flow operator!

```
N, D, H = 3, 4, 5
x = tf.placeholder(tf.float32, shape=(N, D))
z = tf.placeholder(tf.float32, shape=None)
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(D, H))

def f1(): return tf.matmul(x, w1)
def f2(): return tf.matmul(x, w2)
y = tf.cond(tf.less(z, 0), f1, f2)

with tf.Session() as sess:
    values = {
        x: np.random.randn(N, D),
        z: 10,
        w1: np.random.randn(D, H),
        w2: np.random.randn(D, H),
    }
    y_val = sess.run(y, feed_dict=values)
```

Resource: http://cs231n.stanford.edu/2018/

Static vs Dynamic: Loops

$$y_t = (y_{t-1} + x_t) * w$$

PyTorch: Normal Python

```
T, D = 3, 4
y0 = torch.randn(D, requires_grad=True)
x = torch.randn(T, D)
w = torch.randn(D)

y = [y0]
for t in range(T):
    prev_y = y[-1]
    next_y = (prev_y + x[t]) * w
    y.append(next_y)
```

TensorFlow: Special TF control flow

```
T, N, D = 3, 4, 5
x = tf.placeholder(tf.float32, shape=(T, D))
y0 = tf.placeholder(tf.float32, shape=(D,))
w = tf.placeholder(tf.float32, shape=(D,))

def f(prev_y, cur_x):
    return (prev_y + cur_x) * w

y = tf.foldl(f, x, y0)

with tf.Session() as sess:
    values = {
        x: np.random.randn(T, D),
        y0: np.random.randn(D),
        w: np.random.randn(D),
    }
    y_val = sess.run(y, feed_dict=values)
```

Resource: http://cs231n.stanford.edu/2018/

补充：GPU 和 CUDA

- GPU Graphic Processing Unit
- CUDA Compute Unified Device Architecture
- GPU 对于计算机来说，更像是一种专有硬件而不是 Processing Unit
  - CPU 发送“程序”到 GPU
  - CPU 发送数据到 GPU
  - GPU 执行计算
  - CPU 从 GPU 取回结果
  - CPU 处理结果（显示，整合）
- 因此 GPU 的调用不是自然而然的

补充：GPU 和 CUDA

- “世界上有10000种安装 CUDA 的方法，只有几种是对的”
- 操作系统：
  - Windows
  - macOS
  - Linux
- Ubuntu 推荐使用 deb 方式，其它发行版用 runfile 方式
- 在驱动合适的情况下 anaconda 可以安装 CUDA （推荐！）

CUDA 安装

- 下载： https://developer.nvidia.com/cuda-downloads （建议不要用最新版，9.0和9.2版一般比较合适）

Select Target Platform

Click on the green buttons that describe your target platform. Only supported platforms will be shown.

Operating System	Windows / **Linux** / Mac OSX
Architecture	**x86_64** / ppc64le
Distribution	Fedora / OpenSUSE / RHEL / CentOS / SLES / **Ubuntu**
Version	18.10 / **18.04** / 16.04 / 14.04
Installer Type	runfile (local) / **deb (local)** / deb (network) / cluster (local)

Download Installer for Linux Ubuntu 18.04 x86_64

The base installer is available for download below.

Base Installer Download (1.6 GB)

Installation Instructions:

```
1. `sudo dpkg -i cuda-repo-ubuntu1804-10-1-local-10.1.105-418.39_1.0-1_amd64.deb`
2. `sudo apt-key add /var/cuda-repo-<version>/7fa2af80.pub`
3. `sudo apt-get update`
4. `sudo apt-get install cuda`
```

Other installation options are available in the form of meta-packages. For example, to install all the library packages, replace "cuda" with the "cuda-libraries-10-1" meta package. For more information on all the available meta packages click here.

CUDNN

- Accelerate Deep Learning Task for CUDA
- https://developer.nvidia.com/cudnn