深度学习框架Tensorflow

Tensorflow

Tensorflow是google的第二代人工智能操作系统，是一个开源的基于python的机器学习框架，它是由Google开发，并在图形分类，音频处理，推荐系统和自然语言处理等场景有着丰富的应用，是目前最热门的机器学习框架之一。

Tensorflow

1.Tensorflow简介

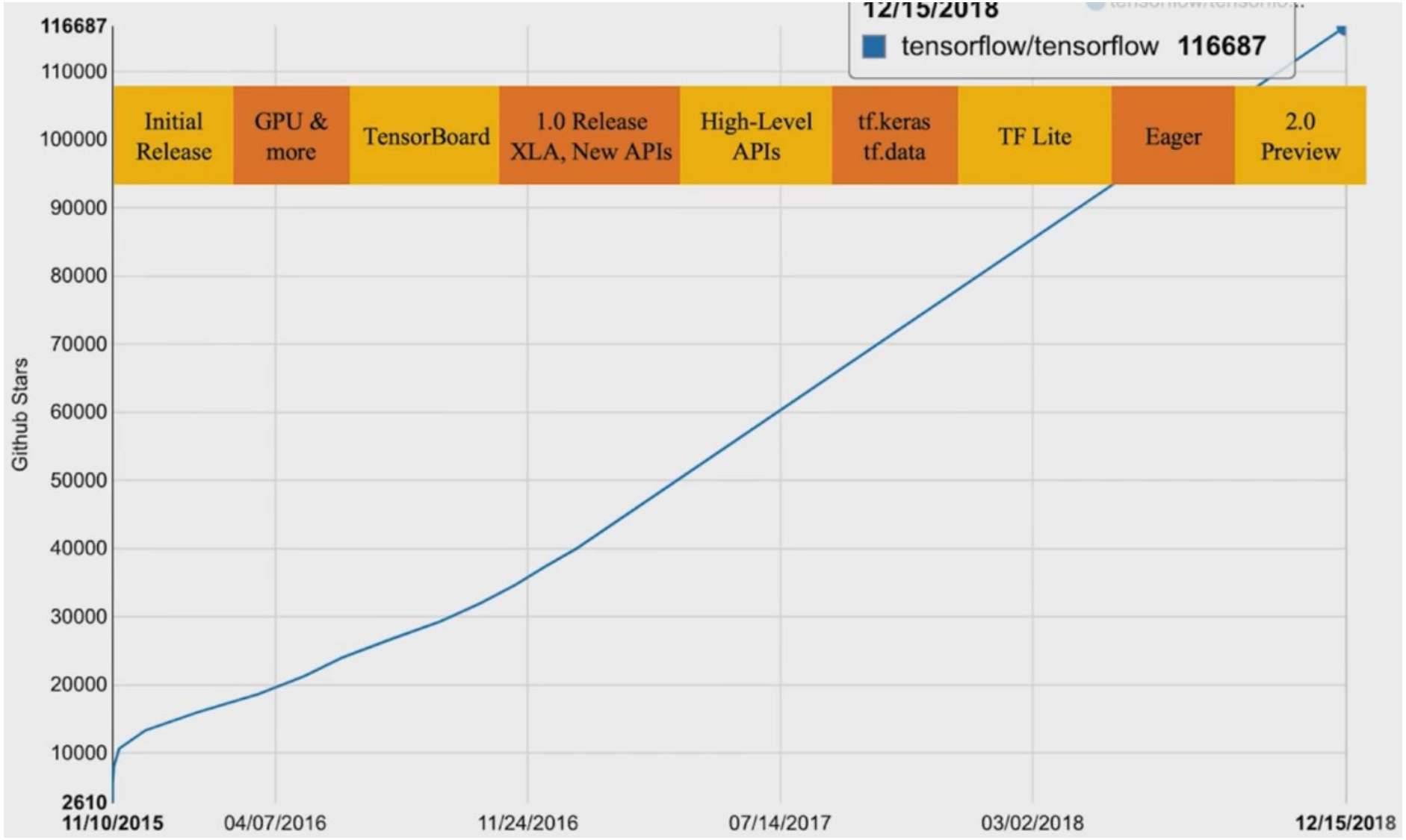
2.Tensorflow基本概念

3.Tensorflow例程解析—手写汉字识别

Tensorflow特点

- 支持所有流行语言，如 Python、C++、Java、R和Go。
- 可以在多种平台上工作，甚至是移动平台和分布式平台。
- Keras, slim——高级神经网络 API，已经与 TensorFlow 整合。
- 与 Torch/Theano 比较，TensorFlow 拥有更好的计算图表可视化。
- 允许模型部署到工业生产中，并且容易使用。
- TensorFlow 不仅仅是一个软件库，它是一套包括 TensorFlow，TensorBoard 和 TensorServing 的软件

Tensorflow发展历程



搭建Tensorflow

- Tensorflow支持的系统
  - Ubuntu 16.04 or later
  - Windows 7 or later
  - macOS 10.12.6 (Sierra) or later (no GPU support)

搭建Tensorflow

- 使用pip安装tensorflow

1. 安装python开发环境：https://www.python.org/

2. (可选)安装虚拟环境，有助于将软件包和系统隔离开来。

```
$ virtualenv --system-site-packages -p python2.7 ./venv
$ source ./venv/bin/activate  # activate virtual env
(venv) $ pip list  # show packages installed within the virtual environment
```

3. 安装tensorflow：

   CPU版本：pip install –upgrade tensorflow

   GPU版本：pip install –upgrade tensorflow-gpu

官网安装流程：https://tensorflow.google.cn/install

Hello world

•第一部分 import 模块包含代码将使用的所有库，在目前的代码中只使用 TensorFlow，其中语句 import tensorflow as tf 则允许 Python 访问 TensorFlow 所有的类、方法和符号。

```
import tensorflow as tf

message = tf.constant('Hello world!')

with tf.Session() as sess:
    print(sess.run(message))
```

•第二个模块包含图形定义部分...创建想要的计算图。在本例中计算图只有一个节点，tensor 常量消息由字符串“Hello world”构成。

•第三个模块是通过会话执行计算图，这部分使用 with 关键字创建了会话，最后在会话中执行以上计算图。

Hello world

```python
import tensorflow as tf

message = tf.constant('Hello world!')

with tf.Session() as sess:
    print(sess.run(message))
```

该程序打印计算图执行的结果，计算图的执行则使用 sess.run() 语句，sess.run 求取 message 中所定义的 tensor 值；计算图执行结果输入到 print 函数

```
totalMemory: 11.91GiB freeMemory: 11.76GiB
2020-10-13 20:52:15.007311: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1511] Adding visible gpu devices: 0
2020-10-13 20:52:15.276547: I tensorflow/core/common_runtime/gpu/gpu_device.cc:982] Device interconnect StreamExecutor with strength
1 edge matrix:
2020-10-13 20:52:15.276574: I tensorflow/core/common_runtime/gpu/gpu_device.cc:988]      0
2020-10-13 20:52:15.276580: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1001] 0:   N
2020-10-13 20:52:15.276668: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1115] Created TensorFlow device (/job:localhost/replic
a:0/task:0/device:GPU:0 with 11376 MB memory) -> physical GPU (device: 0, name: TITAN Xp, pci bus id: 0000:01:00.0, compute capabilit
y: 6.1)
Hello world!
```

Tensorflow

1.Tensorflow简介

2.Tensorflow基本概念

3.Tensorflow例程解析—手写汉字识别

基本概念

- 使用图(graphs)来表示计算任务
- 在称之为会话Session的上下文context上执行图
- 使用张量(tensor)表示数据
- 通过变量（Variable) 维护状态
- 使用feed和fetch可以为任意的操作赋值或者从中获得数据

Tensorflow 是一个编程系统，使用图(graphs)表示计算任务，图(graphs)中的节点称之为op(operation), 一个op获得0个或多个Tensor，执行计算，产生0个或多个Tensor。Tensor 看作是一个n维的数组或列表。 图必须在会话（Session）中被启动。

Tensorflow

1.Tensorflow数据流图

2.张量（Tensor）

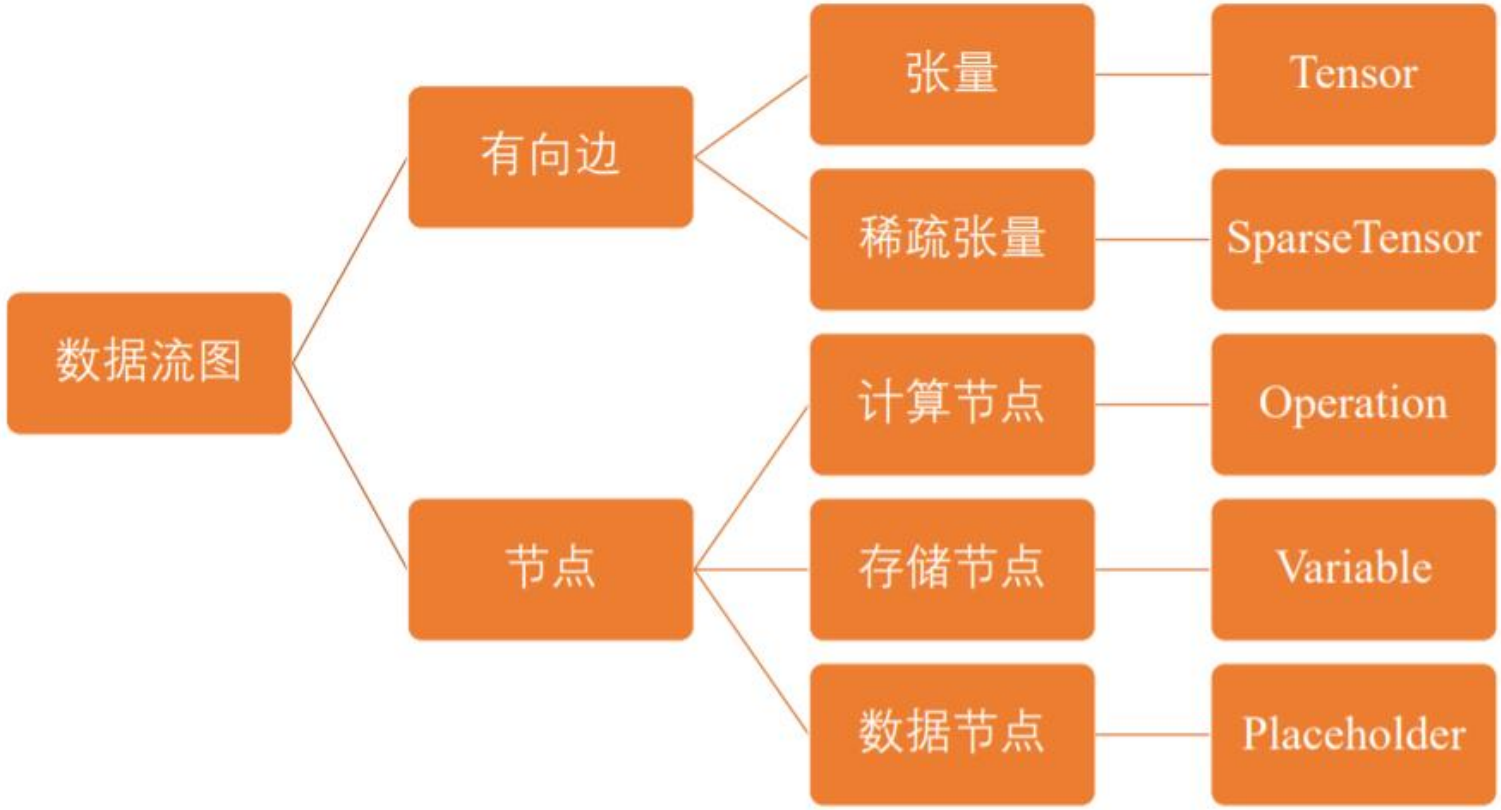
3.变量（Variable）

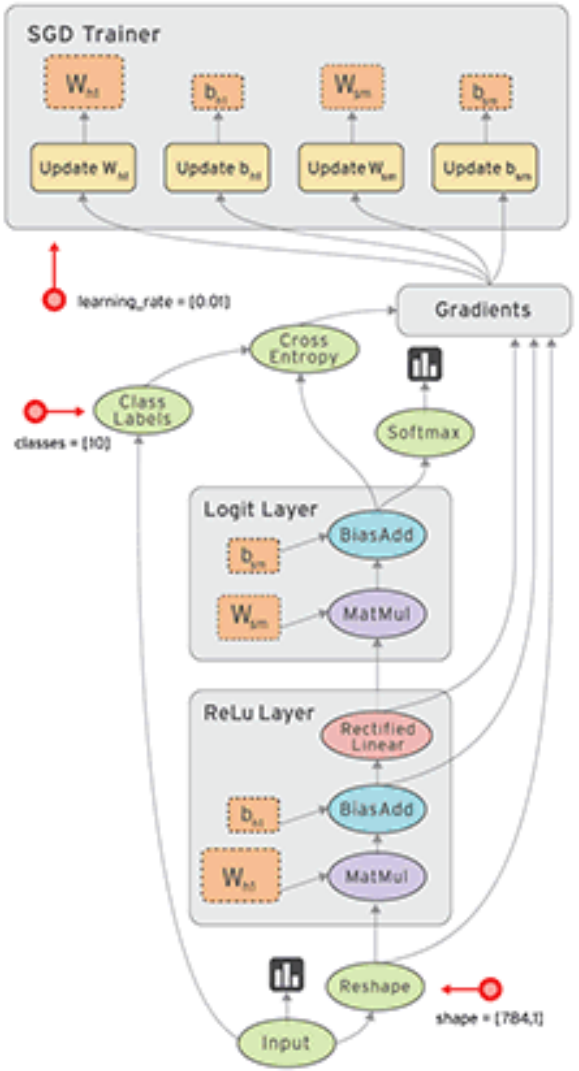
4.操作（Operation）

5.会话（Session）

6.优化器（Optimizer）

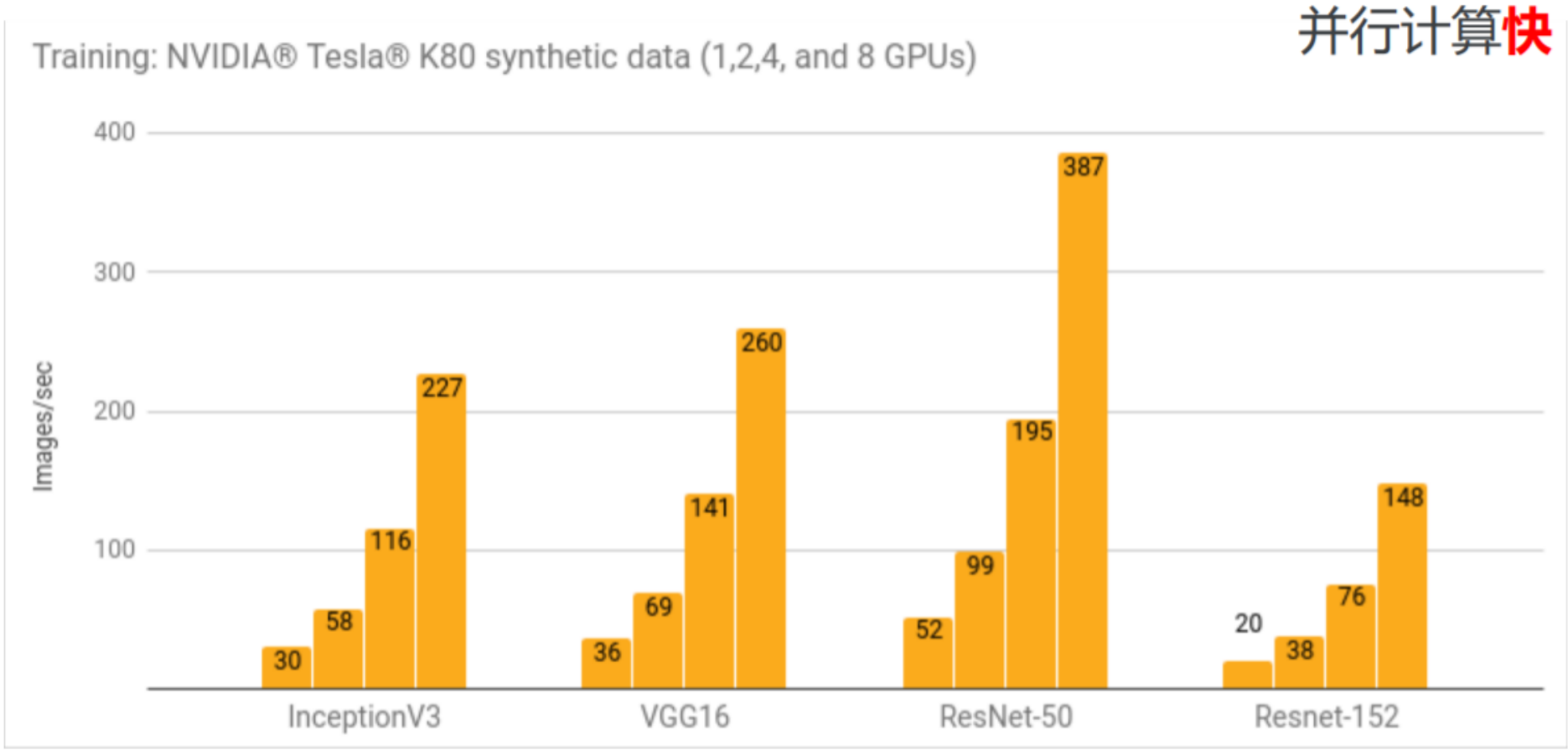
数据流图（graphs）


数据流图
有向边
节点
张量
稀疏张量
计算节点
存储节点
数据节点
Tensor
SparseTensor
Operation
Variable
Placeholder


SGD Trainer
Update W
Update b
Update W
Update b
learning_rate = [0.01]
Gradients
Cross Entropy
Class Labels
classes = [10]
Softmax
Logit Layer
BiasAdd
MatMul
ReLu Layer
Rectified Linear
BiasAdd
MatMul
Reshape
shape = [784,1]
Input

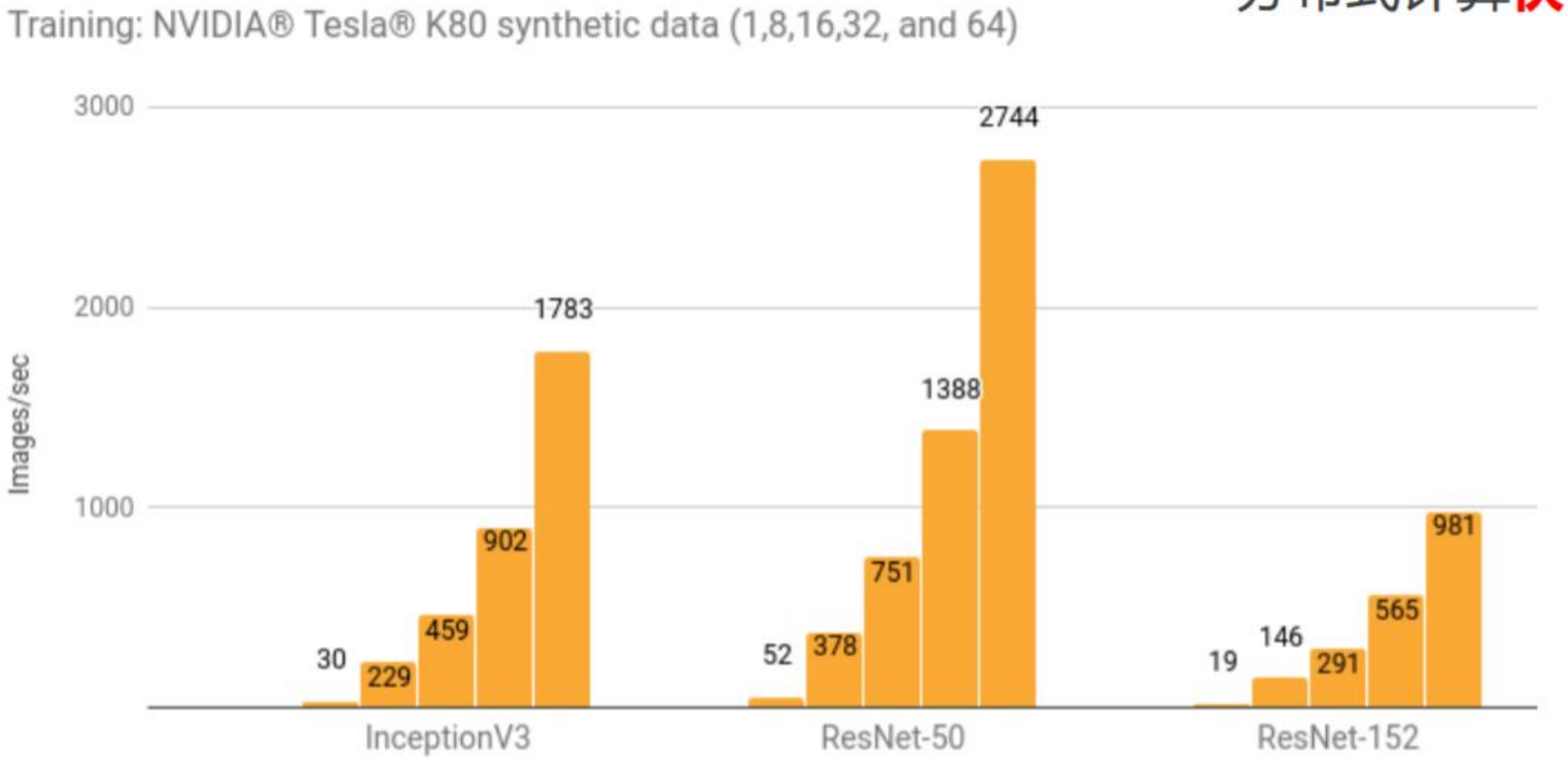
数据流图（graphs）

优势



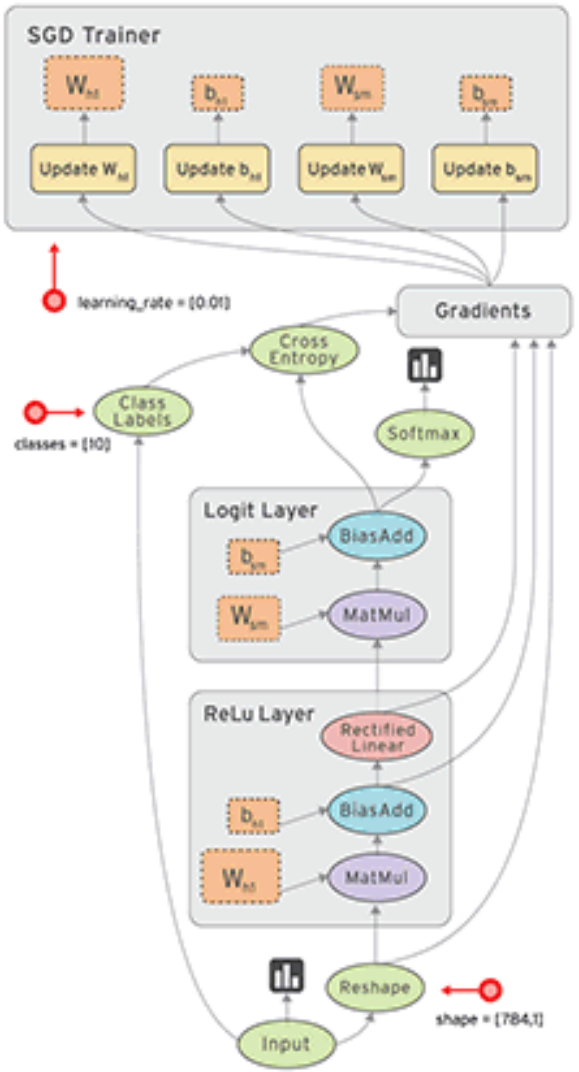
数据流图（graphs）

分布式计算**快**



数据流图（graphs）

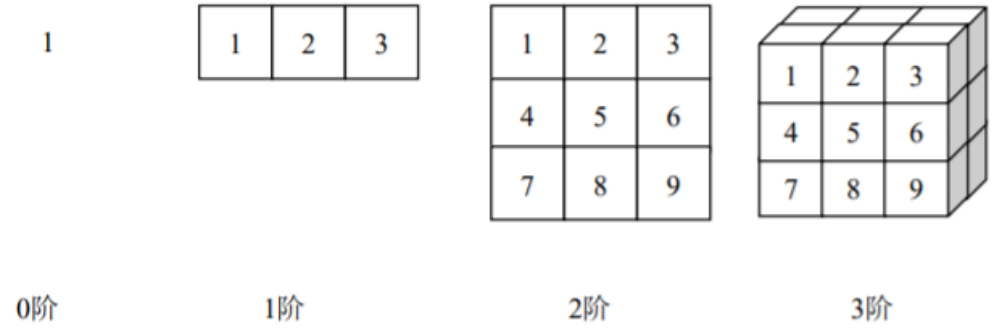
- 并行计算快
- 分布式计算快
- 预编译优化
- 可移植性好



张量（Tensor）

在数学里，张量是一种几何实体，广义上表示任意形式的数据，张量可以理解为0阶标量，1阶向量和2阶矩阵在高维空间的推广，张量的阶描述它表示数据的最大维度。

维数	阶	名字	例子
0-D	0	标量	S = 1 2 3
1-D	1	向量	V = [1,2,3]
2-D	2	矩阵	M = [[1,2],[2,3]]
N-D	n	张量	T = [[[…[[[……]]]…]]]



张量（Tensor）

- **dtype:** tf.float32、tf.int8等, 数据类型
- **Shape:** (n1, n2, n3), 数据形状
- **Name:** tensor1, 数据名称
- **Op:** 该tensor的来源操作
- **Graph:** 该tensor属于的图
- …

在**Tensorflow**中，张量（**Tensor**）表示某种相同数据类型的多维数据

因此张量有两个重要的特性：

- 数据类型
- 数组形状（各个维度的大小）

张量（Tensor）

Tensorflow 张量：

- 张量是用来表示多维数据的
- 张量是执行操作时的输入和输出数据
- 用户通过执行操作(Operation)来创建或计算张量
- 张量的形状不一定在编译时确定，可以在运行时通过形状推断计算得到

张量（Tensor）

- **常量**:

  标量常量： t_1 = tf.constant(4)

  常量向量： t_2 = tf.constant([2,3])

  其他特殊常量生成函数：tf.zeros([M,N],tf.dtype), tf.ones([M,N],tf,dtype), tf.random_uniform([M,N], minval, maxval, seed) ……

- **变量**：变量应该初始化的常量/随机值。下面的代码中创建了张量变量 t_a。两者将被初始化为形状为 [50，50] 的随机均匀分布，最小值=0，最大值=10：

  t_seed = tf.random_uniform([50,50], 0, 10, seed = 0)

  t_a = tf.Variable(t_seed)

- **占位符**：用于将数据提供给计算图。可以使用以下方法定义一个占位符:

  tf.placeholder(dtype,shape=None,name=None)

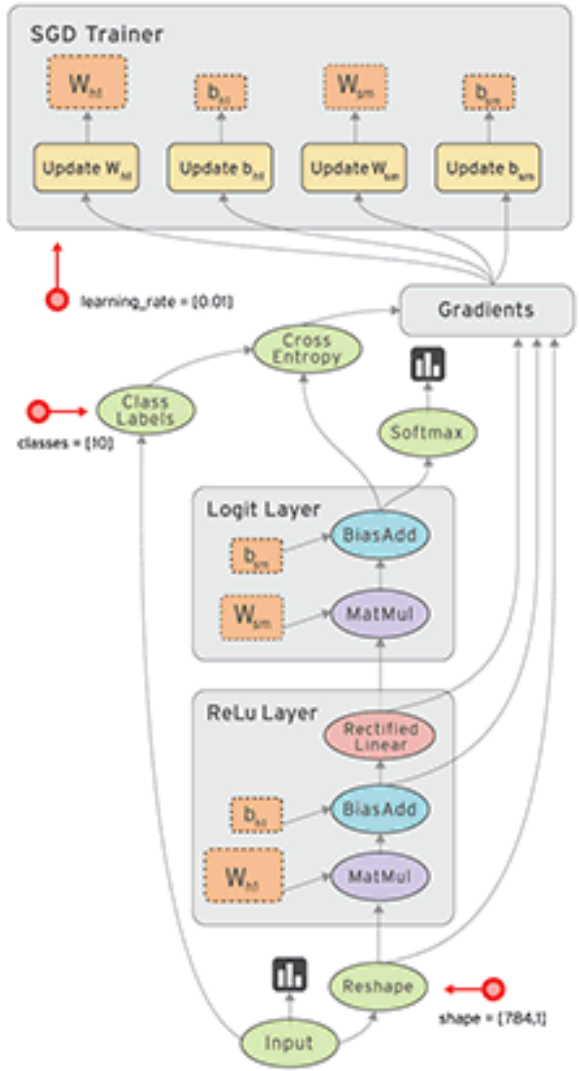
变量（Variable）

Tensortflow变量（Variable）的主要作用时维护特定节点的状态，如深度学习模型参数。

tf.Variable方法是操作，返回值是变量（特殊张量）

通过tf.Variable() 方法创建的变量，与张量一样，可以作为操作的输入和输出，不同之处在于：

- 张量的生命周期通常随依赖的计算完成而完成，内存也随即释放。
- 变量则常驻内存，在每一步训练时不断更新其值，以实现模型参数的更新

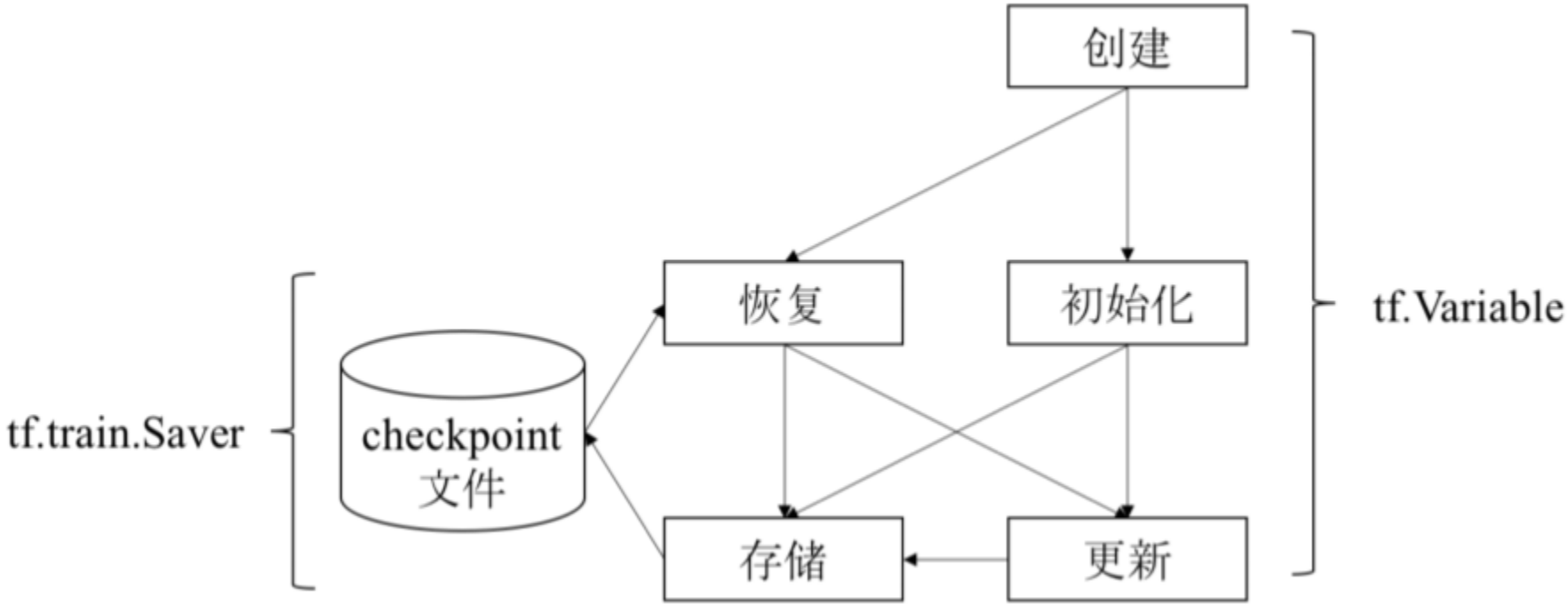


变量（Variable）

```
import tensorflow as tf

# 创建变量
w = tf.Variable(<initial-value>, name=<optional-name>)

# 将变量作为操作的输入
y = tf.matmul(w, ...another variable or tensor...)
z = tf.sigmoid(w + y)

# 使用 assign 或 assign_xxx 方法重新给变量赋值
w.assign(w + 1.0)
w.assign_add(1.0)
```

变量（Variable）

Tensorflow变量使用流程



操作（Operation）

Tensorflow 用数据流图表示算法模型。数据流图由节点和有向边组成，每个节点均对应一个具体的操作，因此，操作是模型功能的实际载体。

数据流图的节点按照功能分为3种：

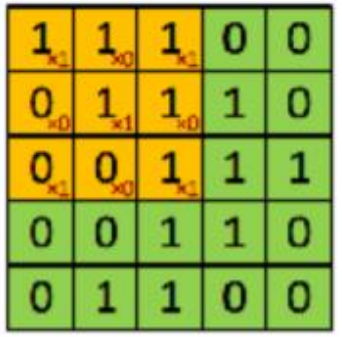
存储节点：有状态的变量操作，通常用来存储模型参数

计算节点：无状态的计算或控制操作，主要负责算法逻辑表达或者控制流程

数据节点：数据的占位符操作，用于描述图外输入数据的属性。

操作的输入和输出是张量或操作

操作（Operation）



tf.add **tf.subtract** **tf.multiply** **tf.divide** **tf.layer.conv2d**

接收零个或者多个Tensor对象作为输入，然后 产生零个或者多个Tensor对象作为输出。

操作（Operation）

Tensorflow 使用占位符操作表示图外的输入数据，如训练集和测试数据

Tensorflow数据流图描述算法的拓扑结构，其中各个节点表示抽象的函数映射或数学表达式

换句话说：数据流图本身是一个具有计算拓扑和内部结构的壳。在用户向数据流图填充数据前，图中并没有真正执行任何的计算。

```
#创建占位符
a = tf.placeholder(tf.float32)
b = tf.placeholder(tf.float32)

c = tf.add(a, b)

with tf.Session() as sess:
    result = sess.run(c, feed_dict={a:3, b:4})
    print(result)
```

会话（Session）

会话提供估算张量和执行操作的运行环境，所有的计算任务都由它链接的执行引擎完成，一个会话典型的使用流程分为3步：

```
#创建会话
sess = tf.Session(target=..., graph=..., config=...)

#执行会话
sess.run()

#关闭会话
sess.close()
```

参数	含义
target	会话的执行引擎
graph	会话加载的数据流图
config	会话启动时的配置项

会话执行（session）

```python
import tensorflow as tf

#创建一个常量
m1 = tf.constants([[3, 3]])
#创建一个常量
m2 = tf.constant([[2],[2]])

#创建一个矩阵乘法op
m3 = tf.matmul(m1, m2)

#创建一个会话，启动默认图
with tf.Session() as sess:
    result = sess.run(m3)
    print(result)
```

```python
import tensorflow as tf

#创建一个变量
m1 = tf.Variable([[3, 3]])
#创建一个常量
m2 = tf.constant([[2],[2]])

#创建一个矩阵乘法op
m3 = tf.matmul(m1, m2)

init = tf.global_variables_initializer()
#创建一个会话，启动默认图

with tf.Session() as sess:
    sess.run(init)
    result = sess.run(m3)
    print(result)
```

- 需要使用sess的run方法来启动静态图来执行矩阵乘法操作
- 变量的使用，在进行sess.run之前需要进行全局变量的初始化
  - 变量的定义将指定变量如何被初始化，但是必须显式初始化所有的声明变量。
  - 在计算图的定义中通过声明初始化操作对象来实现：
    init = tf.global_variables_initializer()

创建图和启动图

占位符
placeholder

feed

```
#创建占位符
a = tf.placeholder(tf.float32)
b = tf.placeholder(tf.float32)

c = tf.add(a, b)

with tf.Session() as sess:
    result = sess.run(c, feed_dict={a:3, b:4})
    print(result)
```

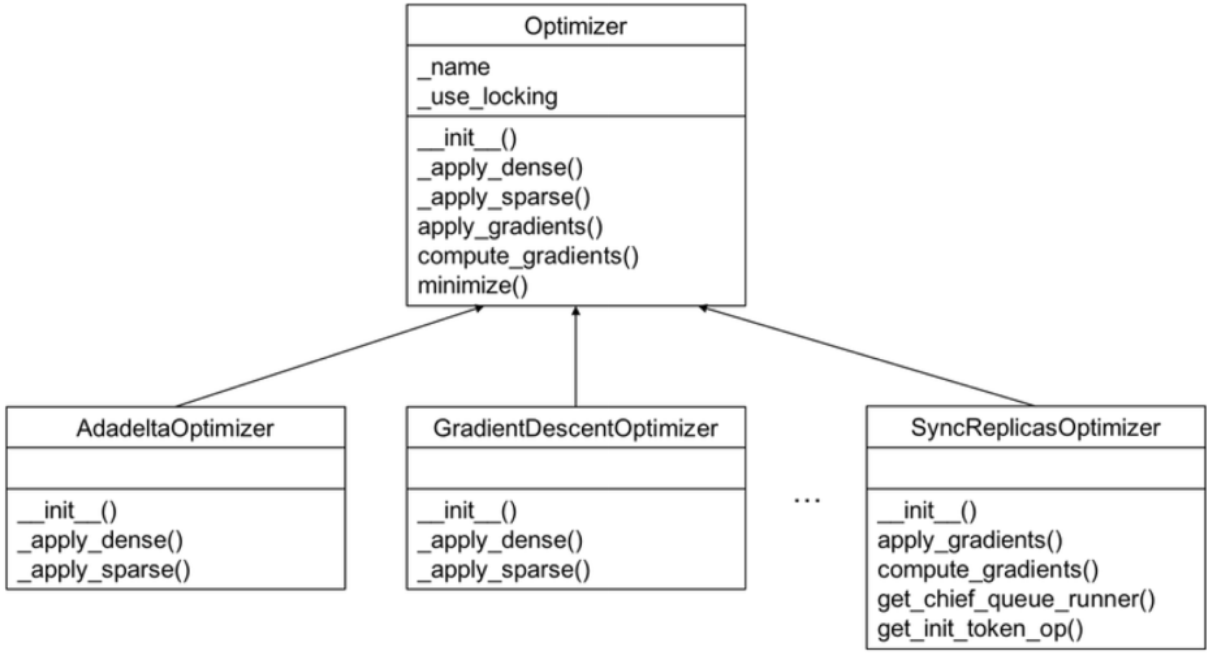
当我们需要执行得到c的运行结果时候我们就需要在会话运行时候，通过feed来插入a与b对应的值

优化器（Optimizer）

优化器是实现优化算法的载体：
一次优化应该分为3个步骤：
1. 计算梯度
2. 处理梯度：用户根据自己的需求处理梯度值，如梯度裁剪和梯度加权等；
3. 应用梯度：将处理后的梯度应用到模型参数

```
#计算梯度
optimizer = tf.train.AdamOptimizer(learning_rate=0.001)
grads_and_vars = optimizer.compute_gradients(loss, var_list, ...)

#处理梯度
clip_grads_and_vars = [(tf.clip_by_value(grad, -1.0,1.0), var)
                       for grad,var in grads_and_vars]

#应用梯度
tranin_op = optimizer.apply_gradients(clip_grads_and_vars)
```

```
#计算应用梯度
optimizer = tf.train.AdamOptimizer(learning_rate=0.001)
tranin_op = optimizer.minimize(loss)
```

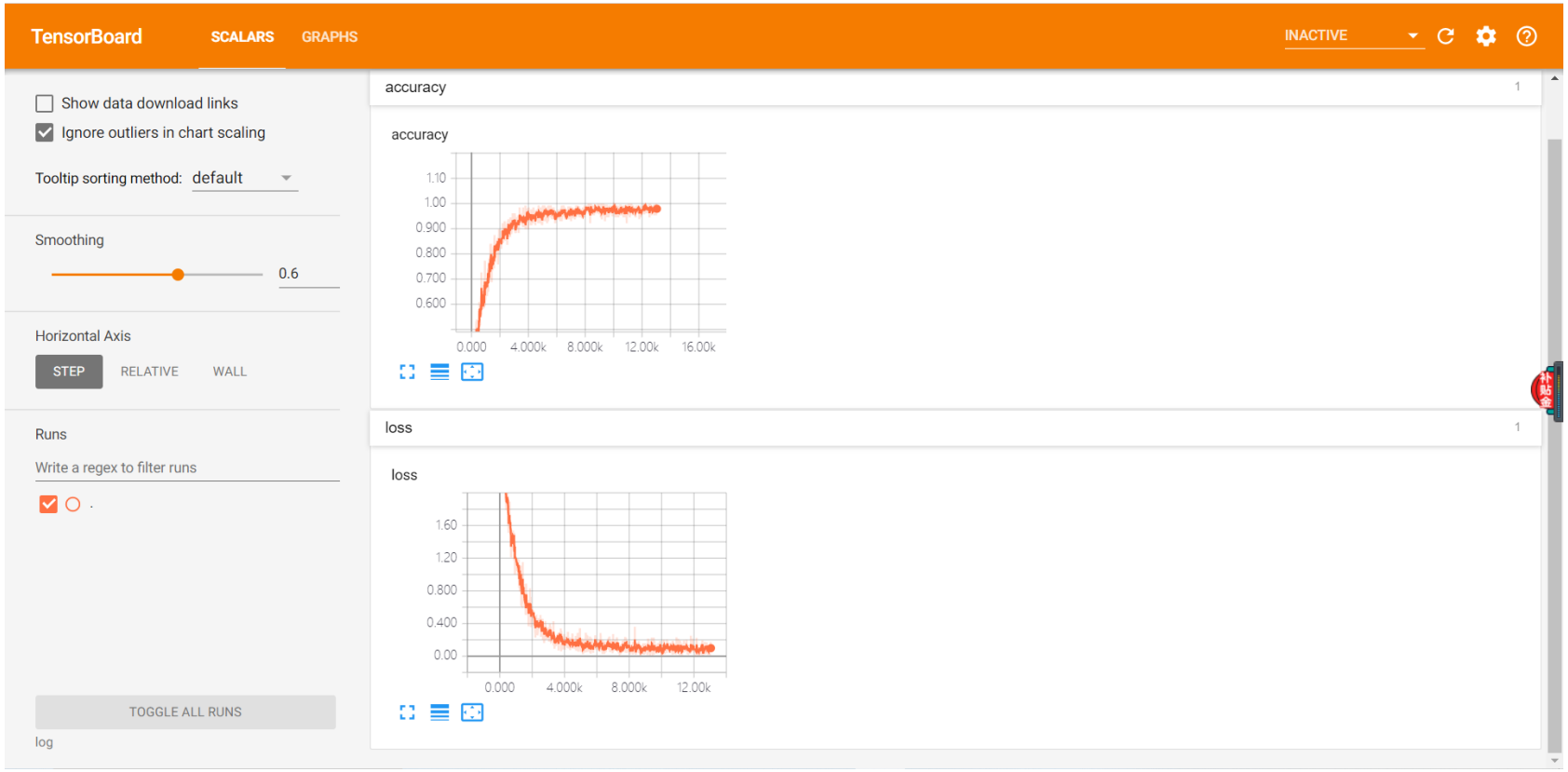


优化器（Operation）

优化器名称	文件路径
Adadelta	tensorflow/python/training/adadelta.py
Adagrad	tensorflow/python/training/adagrad.py
Adagrad Dual Averaging	tensorflow/python/training/adagrad_da.py
Adam	tensorflow/python/training/adam.py
Ftrl	tensorflow/python/training/ftrl.py
Gradient Descent	tensorflow/python/training/gradient_descent.py
Momentum	tensorflow/python/training/momentum.py
Proximal Adagrad	tensorflow/python/training/proximal_adagrad.py
Proximal Gradient Descent	tensorflow/python/training/proximal_gradient_descent.py
Rmsprop	tensorflow/python/training/rmsprop.py
Synchronize Replicas	tensorflow/python/training/sync_replicas_optimizer.py

可视化

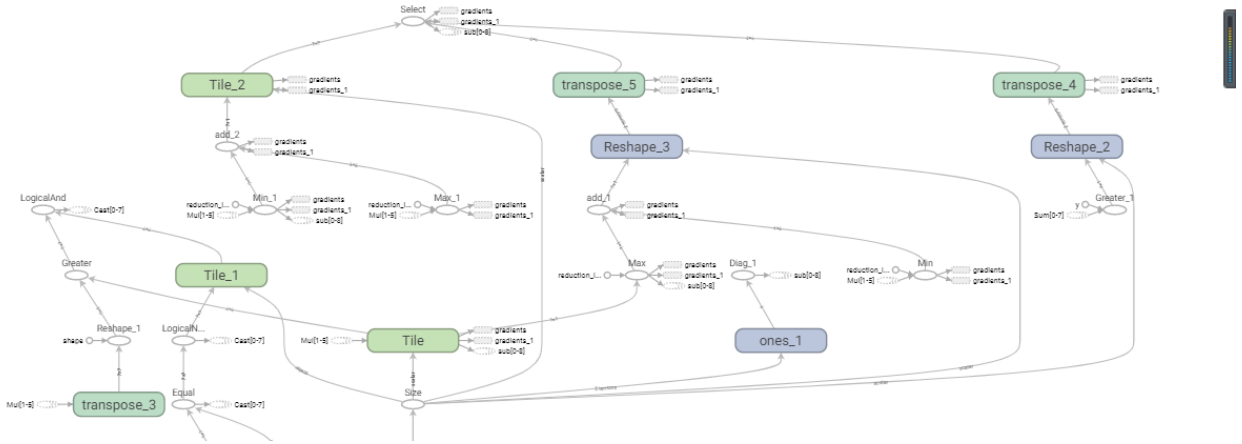
TensorFlow 使用 TensorBoard 来提供计算图形的可视化

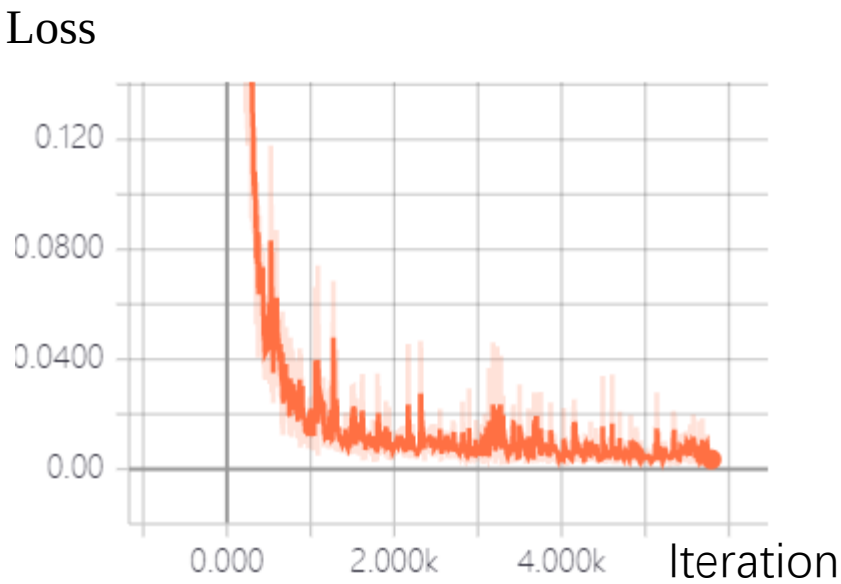


可视化

TensorFlow 使用 TensorBoard 来提供计算图形的可视化

- Accuracy和loss 等指标的变化
- 权重参数的分布
- 图的可视化





可视化

Tensorboard 使用流程

- 添加记录节点:tf.summary.scalar()/image()/histogram()等：
  - tf.summary.scalar(显示标量信息)，使用方法：

```
tf.summary.scalar(tags, values, collections=None, name=None)

例如：tf.summary.scalar('mean', mean)

一般在画loss,accuary时会用到这个函数。
```

可视化

Tensorboard 使用流程

- 添加记录节点:tf.summary.scalar()/image()/histogram()等：
  - tf.summary.histogram(显示直方图信息)，使用方法:

```
tf.summary.histogram(tags, values, collections=None, name=None)

例如:  tf.summary.histogram('histogram', var)

一般用来显示训练过程中变量的分布情况
```

可视化

Tensorboard 使用流程

- 添加记录节点:tf.summary.scalar()/image()/histogram()等：
  - tf.summary.image(显示图片信息)，使用方法：

```
tf.summary.image(tag, tensor, max_images=3, collections=None, name=None)
```

  - tf.summary.distribution():用来显示weight的分布

可视化

Tensorboard 使用流程

- 汇总记录节点：merged = tf.summary.merge_all()
  merge_all 可以将summary全部保存到磁盘中，以便tensorboard显示，使用方法：

```
tf.summaries.merge_all(key='summaries')
```

- 运行汇总节点，summary = sess.run(merged),得到汇总结果
- 日志书写器实例化：summary_writer = tf.summary.FileWriter(logdir, graph=sess.graph)，实例化的同时传入 graph 将当前计算图写入日志。使用方法：

```
tf.summary.FileWritter(path,sess.graph)
可以调用其add_summary（）方法将训练过程数据保存在filewriter指定的文件中
```

- 调用日志书写器对象summary_writer的add_summary(summary,global_step = i) 方法将所有汇总写入文件。

可视化

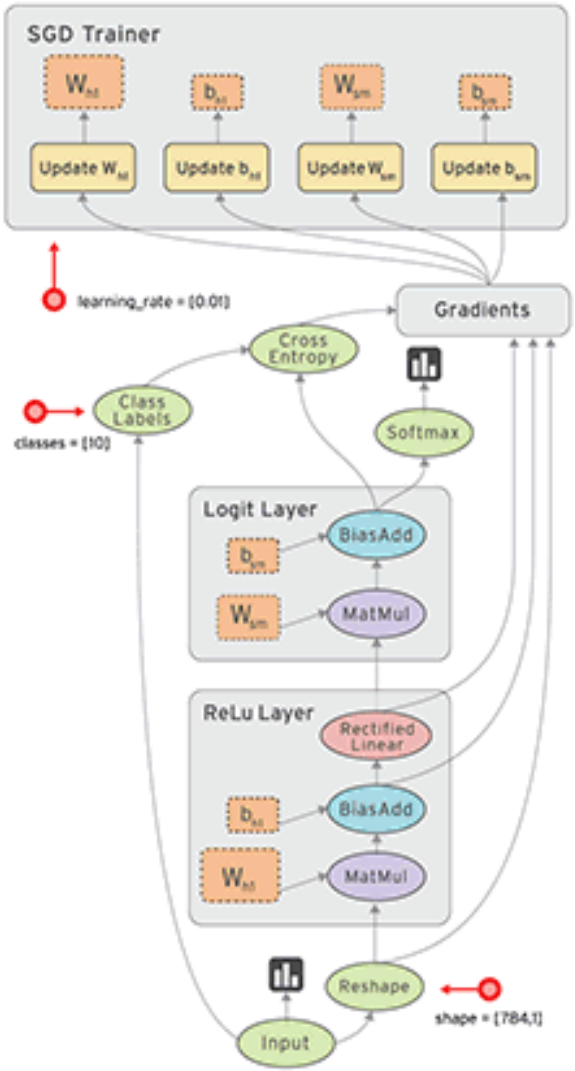
启动tensorboard：

在DOS运行：tensorboard –logdir=./log

会生成一个网址（http:\DESKTOP:6006），输入到浏览器中即可查看

整体流程

1.创建图(Graph)
2.创建占位符(Placeholder)
3.为图添加操作(Operation)
4.创建会话
5.实现计算



Tensorflow

1. Tensorflow简介

2. Tensorflow基本概念

3. Tensorflow例程解析—手写汉字识别

例程解析

手写汉字识别

数据集：**CASIA-HWDB： 中科院自动化系手写汉字样本数据集**

CASIA-HWDB 数据集是最常见的手写汉字识别数据集，它包含脱机、联机两部分，分单字、文本行两种类型：

- HWDB1.x：脱机单字，1.0~1.2 三个版本，数据格式为 `.gnt`
- OLHWDB1.x：联机单字，1.0~1.2 三个版本，
- HWDB2.x：脱机文本行，1.0~1.2 三个版本，数据格式为 `.dgrl`
- OLHWDB1.x：联机文本行，1.0~1.2 三个版本，

例程解析

手写汉字识别

数据集：**CASIA-HWDB**

Table 2. Format of offline isolated character data file (*.gnt)

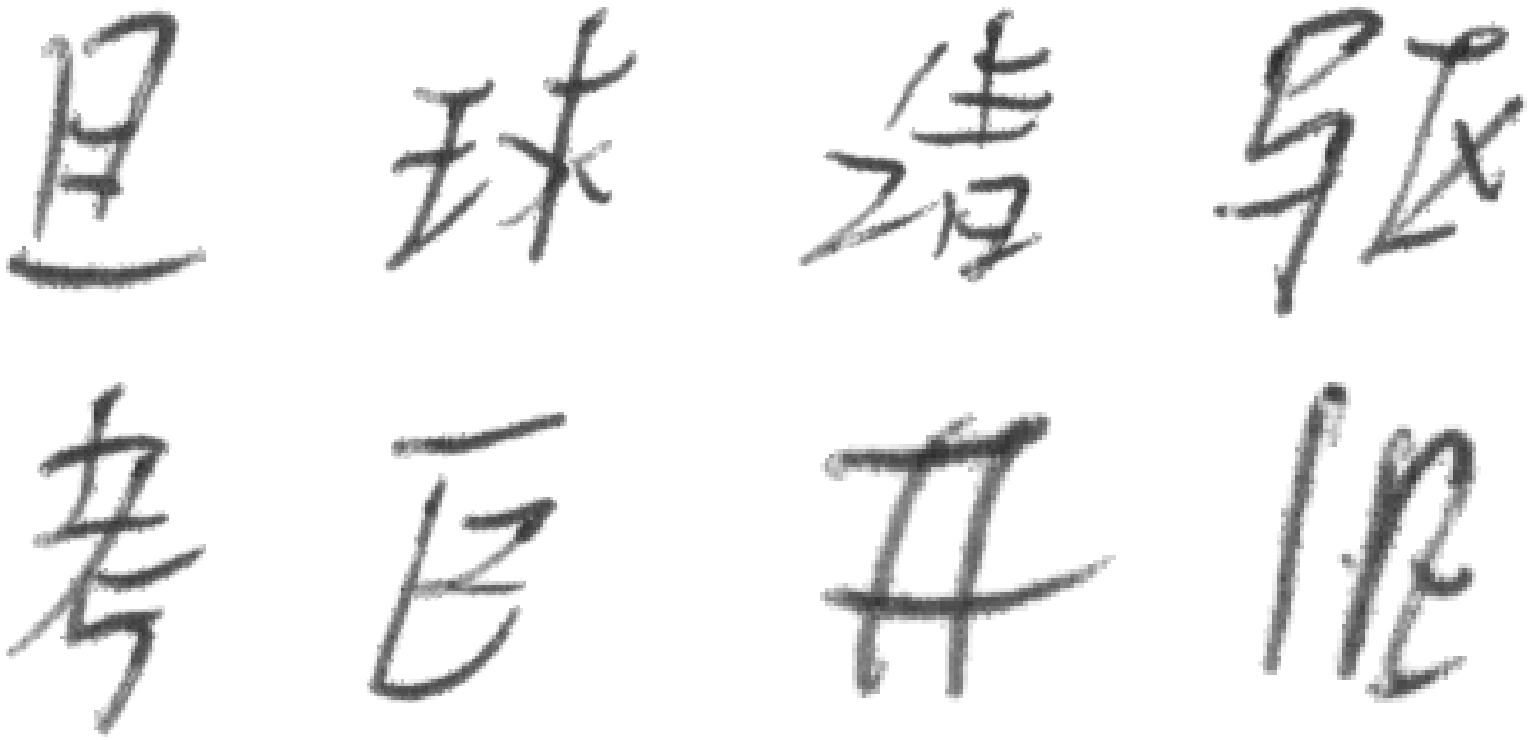
Item	Type	Length	Instance	Comment
Sample size	unsigned int	4B		Number of bytes for one sample (byte count to next sample)
Tag code (GB)	char	2B	"啊"=0xb0a1 Stored as 0xa1b0	
Width	unsigned short	2B		Number of pixels in a row
Height	unsigned short	2B		Number of rows
Bitmap	unsigned char	Width*Height bytes		Stored row by row

格式说明：sample_size表示样本的大小（即图像的大小），也就是说上一张的图像数据与下一张的数据大小是4B差，即一张图片的大小为4Bm,第二个Tag_code表示对应字符的GBK编码，也就是实际的汉字（标签）。接下来就是图像的长宽，最后Bitmap是压缩格式。

例程解析

手写汉字识别

数据集：**CASIA-HWDB**



例程解析

1.导入模块和所用的包

```
import os
import numpy as np
import struct
import PIL.Image
import scipy.misc
from sklearn.utils import shuffle
import tensorflow as tf
```

例程解析

2. 数据预处理

2.1载入图像和对应的汉字(训练集和测试集)

```
def read_from_gnt_dir(gnt_dir=train_data_dir):
    def one_file(f):
        header_size = 10
        while True:
            header = np.fromfile(f, dtype='uint8', count=header_size)
            if not header.size: break
            sample_size = header[0] + (header[1]<<8) + (header[2]<<16) + (header[3]<<24)
            tagcode = header[5] + (header[4]<<8)
            width = header[6] + (header[7]<<8)
            height = header[8] + (header[9]<<8)
            if header_size + width*height != sample_size:
                break
            image = np.fromfile(f, dtype='uint8', count=width*height).reshape((height, width))
            yield image, tagcode

    for file_name in os.listdir(gnt_dir):
        if file_name.endswith('.gnt'):
            file_path = os.path.join(gnt_dir, file_name)
            with open(file_path, 'rb') as f:
                for image, tagcode in one_file(f):
                    yield image, tagcode
```

例程解析

2. 数据预处理

2. 2准备训练集学习数据

```
train_data_dir = “HWDB1.1trn_gnt” #训练集存储路径
train_data_x = [] #训练集输入的图片(image)
train_data_y = [] #训练集图像的标注(label)
char_set = "的一是了我不人在他有这个上们来到时大地为子中你说
生国年着就那和要她出也得里后自以会家可下而过天去能对小多然
于心学么之都好看起发当没成只如事把还用第样道想作种开美总从
无情己面最女但现前些所同日手又行意动方期它头经长儿回位分爱
老因很给名法间斯知世什两次使身者被高已亲其进此话常与活正感"
for image, tagcode in read_from_gnt_dir(gnt_dir=train_data_dir):
    tagcode_unicode = struct.pack('>H', tagcode).decode('gb2312')
    if tagcode_unicode in char_set:
        train_data_x.append(resize_and_normalize_image(image))
        train_data_y.append(convert_to_one_hot(tagcode_unicode))

# shuffle样本
train_data_x, train_data_y = shuffle(train_data_x, train_data_y, random_state=0)
```

```python
def resize_and_normalize_image(img):
    # 补方
    pad_size = abs(img.shape[0]-img.shape[1]) // 2
    if img.shape[0] < img.shape[1]:
        pad_dims = ((pad_size, pad_size), (0, 0))
    else:
        pad_dims = ((0, 0), (pad_size, pad_size))
    img = np.lib.pad(img, pad_dims, mode='constant', constant_values=255)
    # 缩放
    img = scipy.misc.imresize(img, (64 - 4*2, 64 - 4*2))
    img = np.lib.pad(img, ((4, 4), (4, 4)), mode='constant', constant_values=255)
    assert img.shape == (64, 64)

    img = img.flatten()
    # 像素值范围-1到1
    img = (img - 128) / 128
    return img
```

```python
# one hot
def convert_to_one_hot(char):
    vector = np.zeros(len(char_set))
    vector[char_set.index(char)] = 1
    return vector
```

例程解析

2. 数据预处理

2. 2准备训练集学习数据

```
train_data_dir = “HWDB1.1trn_gnt” #训练集存储路径
train_data_x = [] #训练集输入的图片(image)
train_data_y = [] #训练集图像的标注(label)
char_set = "的一是了我不人在他有这个上们来到时大地为子中你说
生国年着就那和要她出也得里后自以会家可下而过天去能对小多然
于心学么之都好看起发当没成只如事把还用第样道想作种开美总从
无情己面最女但现前些所同日手又行意动方期它头经长儿回位分爱
老因很给名法间斯知世什两次使身者被高已亲其进此话常与活正感"
for image, tagcode in read_from_gnt_dir(gnt_dir=train_data_dir):
    tagcode_unicode = struct.pack('>H', tagcode).decode('gb2312')
    if tagcode_unicode in char_set:
        train_data_x.append(resize_and_normalize_image(image))
        train_data_y.append(convert_to_one_hot(tagcode_unicode))

# shuffle样本
train_data_x, train_data_y = shuffle(train_data_x, train_data_y, random_state=0)
```

```python
def resize_and_normalize_image(img):
    # 补方
    pad_size = abs(img.shape[0]-img.shape[1]) // 2
    if img.shape[0] < img.shape[1]:
        pad_dims = ((pad_size, pad_size), (0, 0))
    else:
        pad_dims = ((0, 0), (pad_size, pad_size))
    img = np.lib.pad(img, pad_dims, mode='constant', constant_values=255)
    # 缩放
    img = scipy.misc.imresize(img, (64 - 4*2, 64 - 4*2))
    img = np.lib.pad(img, ((4, 4), (4, 4)), mode='constant', constant_values=255)
    assert img.shape == (64, 64)

    img = img.flatten()
    # 像素值范围-1到1
    img = (img - 128) / 128
    return img
```

```python
# one hot
def convert_to_one_hot(char):
    vector = np.zeros(len(char_set))
    vector[char_set.index(char)] = 1
    return vector
```

例程解析

2. 数据预处理

2. 2准备测试集学习数据

```
test_data_dir = “HWDB1.1tst_gnt” #测试集图片存储路径
test_data_x = [] #测试集输入的图片(image)
test_data_y = [] #测试集图像的标注(label)

for image, tagcode in read_from_gnt_dir(gnt_dir=test_data_dir):
    tagcode_unicode = struct.pack('>H', tagcode).decode('gb2312')
    if tagcode_unicode in char_set:
        test_data_x.append(resize_and_normalize_image(image))
        test_data_y.append(convert_to_one_hot(tagcode_unicode))

# shuffle样本
test_data_x, test_data_y = shuffle(test_data_x, test_data_y, random_state=0)
```

例程解析

3. 搭建模型

3.1定义学习超参数

```
batch_size = 128  # 每一批次所用的训练数据
num_batch = len(train_data_x) // 一共有多少个批次
learning_rate=0.001 #学习率
epoch = 100 # 纪元数(使用所有训练数据一次为一个纪元)
```

例程解析

3. 搭建模型

3.1定义学习超参数

```
batch_size = 128  # 每一批次所用的训练数据
num_batch = len(train_data_x) // 一共有多少个批次
learning_rate=0.001 #学习率
epoch = 100 # 纪元数(使用所有训练数据一次为一个纪元)
```

3.2 创建占位符

```
X = tf.placeholder(tf.float32, [None, 64*64]) #输入图片
Y = tf.placeholder(tf.float32, [None, 140])  #输入label
keep_prob = tf.placeholder(tf.float32) #dropout层保留率
```

3. 搭建模型

3. 3构建数据流图

例程解析

```
def chinese_hand_write_cnn():
    x = tf.reshape(X, shape=[-1, 64, 64, 1])  # [128, 64, 64, 1]
    # 2 conv layers
    w_c1 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
    b_c1 = tf.Variable(tf.zeros([32]))
    conv1 = tf.nn.relu(tf.nn.bias_add(tf.nn.conv2d(x, w_c1, strides=[1, 1, 1, 1], padding='SAME'), b_c1)) #[128, 64, 64, 32]
    conv1 = tf.nn.max_pool(conv1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')  #[128, 32, 32, 32]

    w_c2 = tf.Variable(tf.random_normal([3, 3, 32, 64], stddev=0.01))
    b_c2 = tf.Variable(tf.zeros([64]))
    conv2 = tf.nn.relu(tf.nn.bias_add(tf.nn.conv2d(conv1, w_c2, strides=[1, 1, 1, 1], padding='SAME'), b_c2))  #[128, 32, 32, 64]
    conv2 = tf.nn.max_pool(conv2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')   #[128, 16, 16, 64]

    # fully connect layer
    w_d = tf.Variable(tf.random_normal([8*32*64, 1024], stddev=0.01))
    b_d = tf.Variable(tf.zeros([1024]))
    dense = tf.reshape(conv2, [-1, w_d.get_shape().as_list()[0]])  #[128, 16*16*64]
    dense = tf.nn.relu(tf.add(tf.matmul(dense, w_d), b_d))  #[128, 1024]
    dense = tf.nn.dropout(dense, keep_prob)

    w_out = tf.Variable(tf.random_normal([1024, 140], stddev=0.01))
    b_out = tf.Variable(tf.zeros([140]))
    out = tf.add(tf.matmul(dense, w_out), b_out)  # [128, 140]
    return out
```

3. 搭建模型

3. 3构建数据流图

```
output = chinese_hand_write_cnn()  #得到网络的输出

loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(output, Y))  #计算代价函数
optimizer = tf.train.AdamOptimizer(learning_rate=0.001).minimize(loss)  #使用梯度下降算法最小化代价函数

accuracy = tf.reduce_mean(tf.cast(tf.equal(tf.argmax(output, 1), tf.argmax(Y, 1)), tf.float32)) #计算精度
init = tf.global_variables_initializer()#使用全局变量初始化选项
```

3.4 tensorboard 可视化

```
tf.summary.scalars("loss", loss)  #记录损失
tf.summary.scalars("accuracy", accuracy)  #记录精度
merged_summary_op = tf.merge_all_summaries()
```

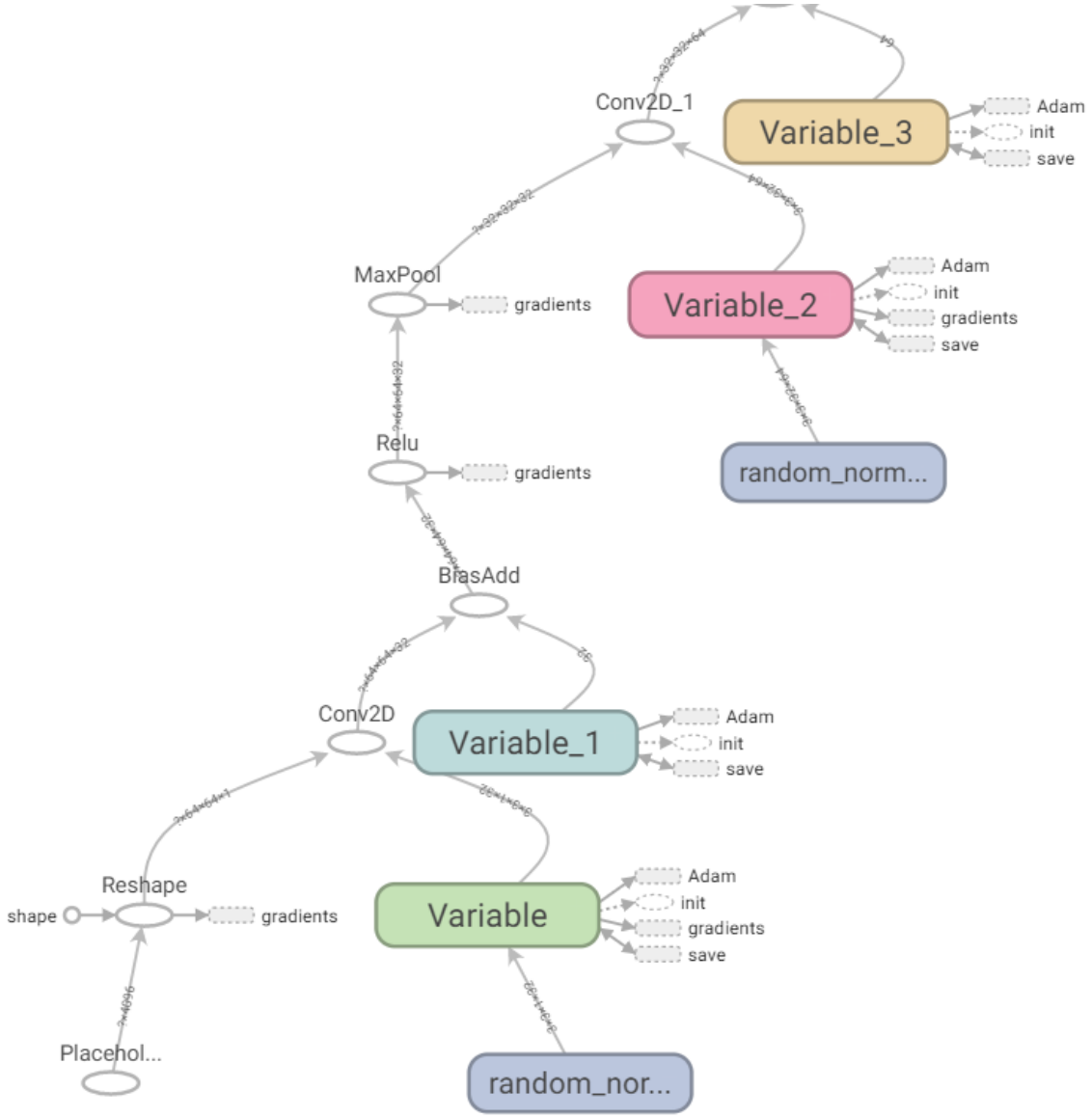
3. 搭建模型

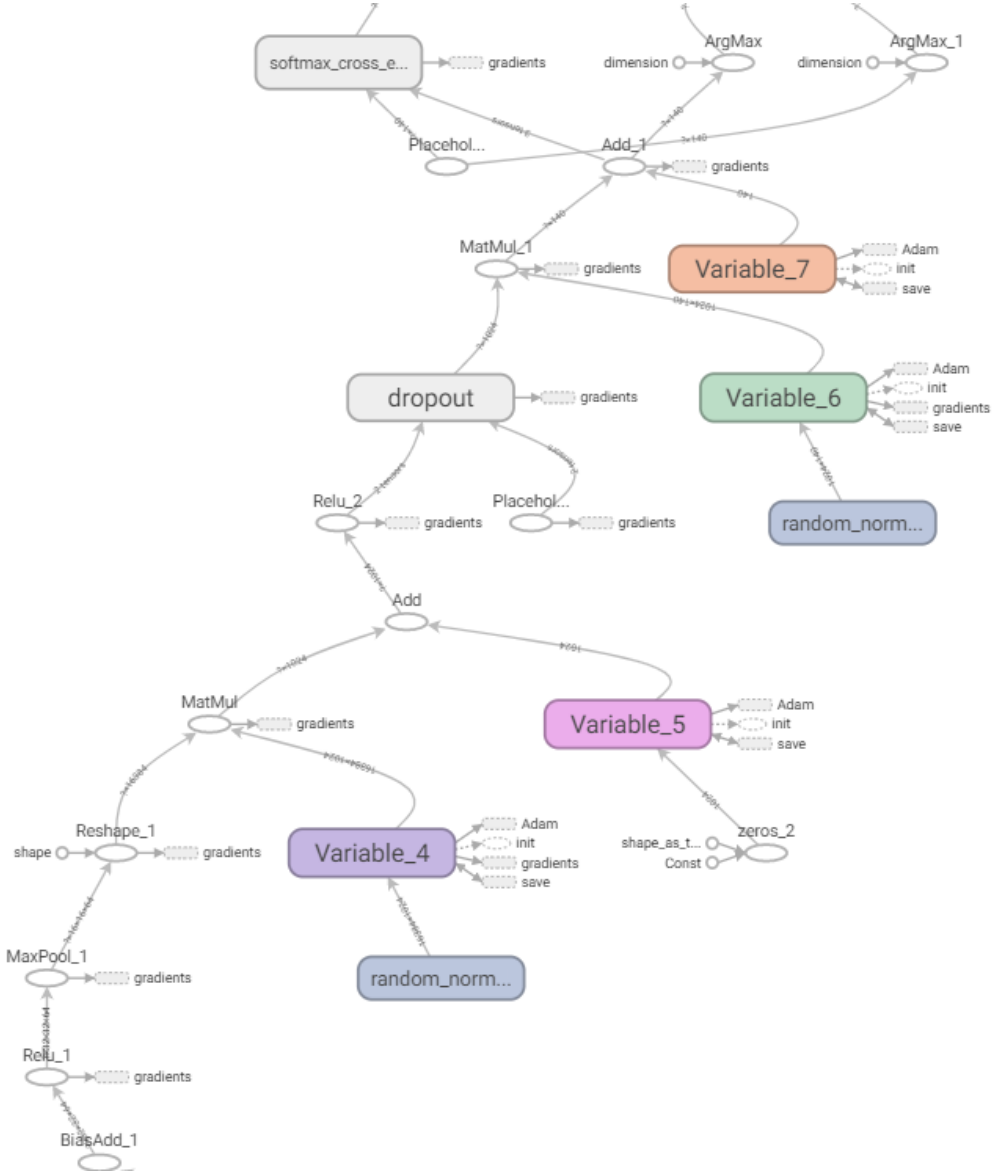
3. 4创建会话

例程解析

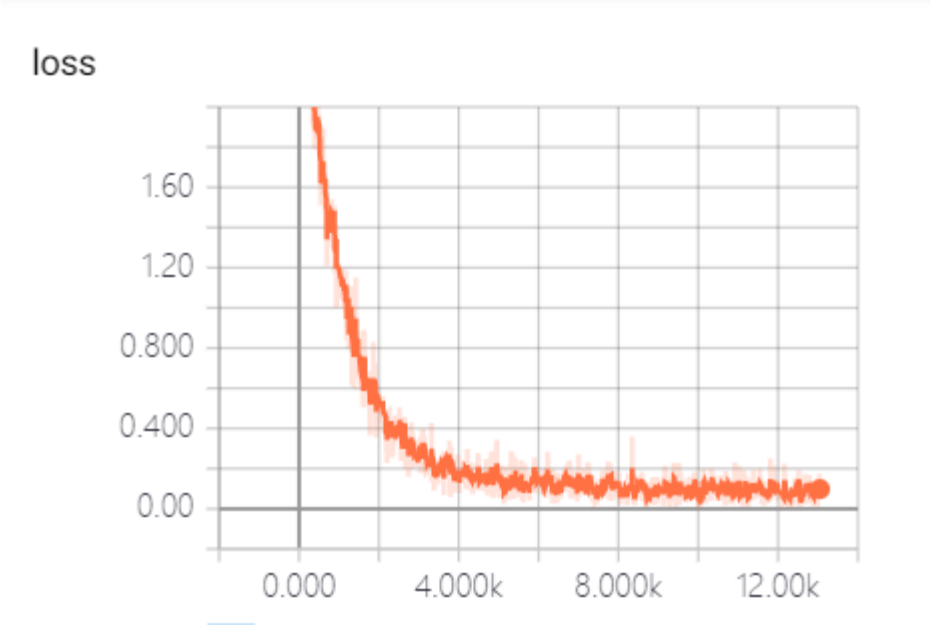
```
with tf.Session() as sess:
    sess.run(init)

    # 命令行执行 tensorboard --logdir=./log  打开浏览器访问http://0.0.0.0:6006
    summary_writer = tf.train.SummaryWriter('./log', graph=tf.get_default_graph())

    for e in range(epoch):
        for i in range(num_batch):
            batch_x = train_data_x[i*batch_size : (i+1)*batch_size]
            batch_y = train_data_y[i*batch_size : (i+1)*batch_size]
            _, loss_, summary = sess.run([optimizer, loss, merged_summary_op], feed_dict={X: batch_x, Y: batch_y, keep_prob: 0.5})
            # 每次迭代都保存日志
            summary_writer.add_summary(summary, e*num_batch+i)
            print(e*num_batch+i, loss_)

            if  (e*num_batch+i) % 100 == 0:
                # 计算准确率
                acc = accuracy.eval({X: text_data_x[:500], Y: text_data_y[:500], keep_prob: 1.})
                #acc = sess.run(accuracy, feed_dict={X: text_data_x[:500], Y: text_data_y[:500], keep_prob: 1.})
                print(e*num_batch+i, acc)
```

数据流图

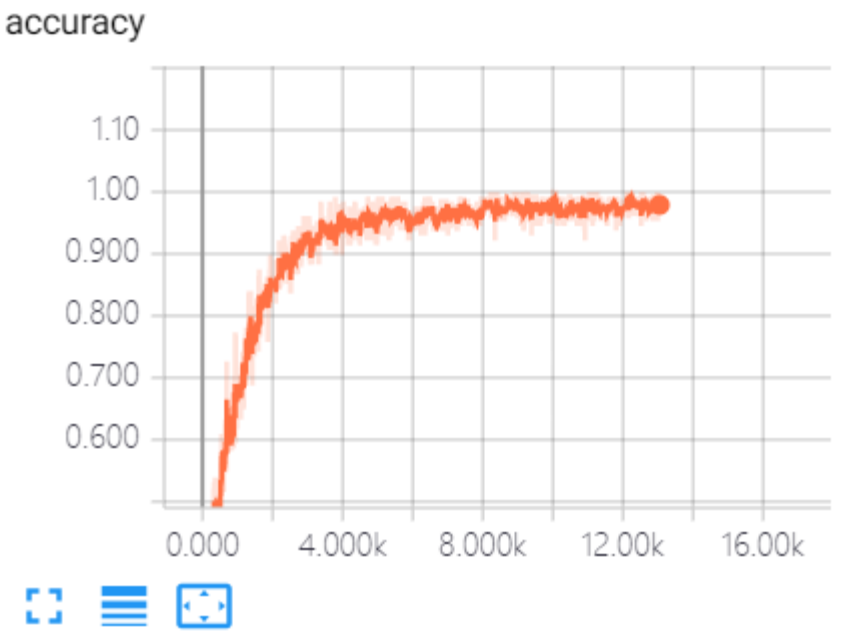

Conv2D_1
Variable_3
Adam
init
save
MaxPool
gradients
Variable_2
Adam
init
gradients
save
Relu
gradients
random_norm...
BiasAdd
Conv2D
Variable_1
Adam
init
save
Reshape
shape
gradients
Variable
Adam
init
gradients
save
Placehol...
random_nor...


softmax_cross_e...
gradients
ArgMax
dimension
ArgMax_1
dimension
Placehol...
Add_1
gradients
MatMul_1
gradients
Variable_7
Adam
init
save
dropout
gradients
Variable_6
Adam
init
gradients
save
Relu_2
gradients
Placehol...
gradients
random_norm...
Add
MatMul
gradients
Variable_5
Adam
init
save
Reshape_1
shape
gradients
Variable_4
Adam
init
gradients
save
shape_as_t...
Const
zeros_2
MaxPool_1
gradients
random_norm...
Relu_1
gradients
BiasAdd_1

损失和acc的指标监控：



损失loss值



测试集accuracy