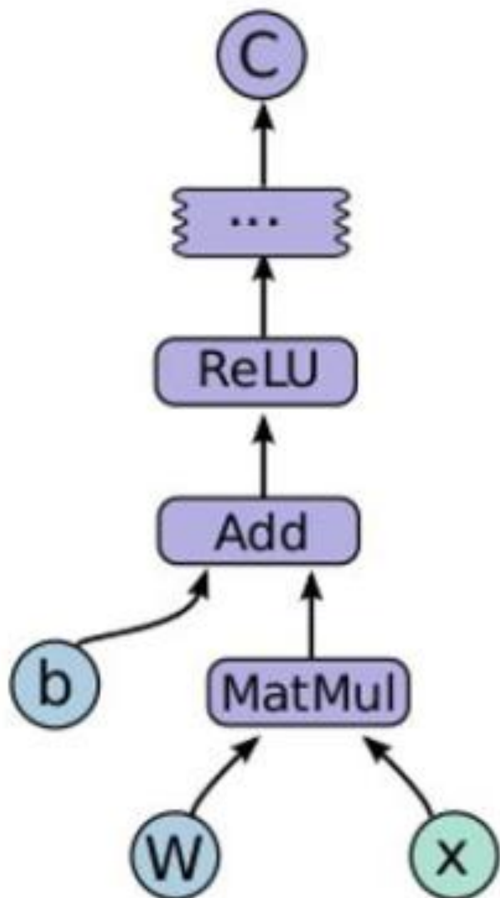
TensorFlow

Contents

1. 基本概念
2. 例程解析
3. 特征

01 基本概念

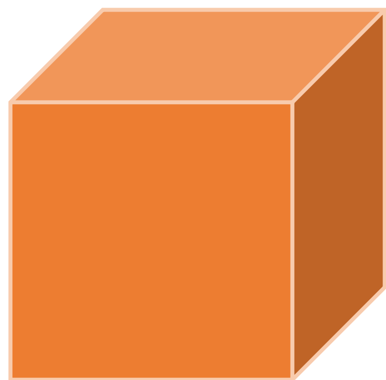
数据流图（tf.Graph）



- Tensorflow是基于图（Graph）的计算系统。
- 图的节点则是由操作（Operation）来构成的，而图的各个节点之间则是由张量（Tensor）作为边来连接在一起的。
- 数据流图表示计算指令之间的依赖关系和组合的方式,包含图结构的信息。
- 在构建数据流图时并不会真的触发计算。
- 数据流图的好处在于合理安排各个操作之间的并行或顺次执行,协同不同的底层设备,避免冗余操作。

01 基本概念

张量（tf.Tensor）

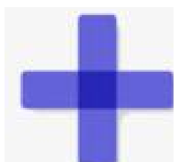


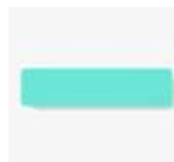
- **dtype:** tf.float32、tf.int8等, 数据类型
- **Shape:** (n1, n2, n3), 数据形状
- **Name:** tensor1, 数据名称
- **Op:** 该tensor的来源操作
- **Graph:** 该tensor属于的图
- …

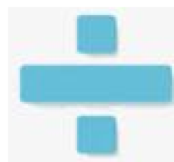
◆ 万物皆张量
◆ 张量包括常数\变量\占位符等
◆ 涉及到数据的都是张量

01 基本概念

操作（tf.Operation）







$1_{\times1}$	$1_{\times0}$	$1_{\times1}$	0	0
$0_{\times0}$	$1_{\times1}$	$1_{\times0}$	1	0
$0_{\times1}$	$0_{\times0}$	$1_{\times1}$	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

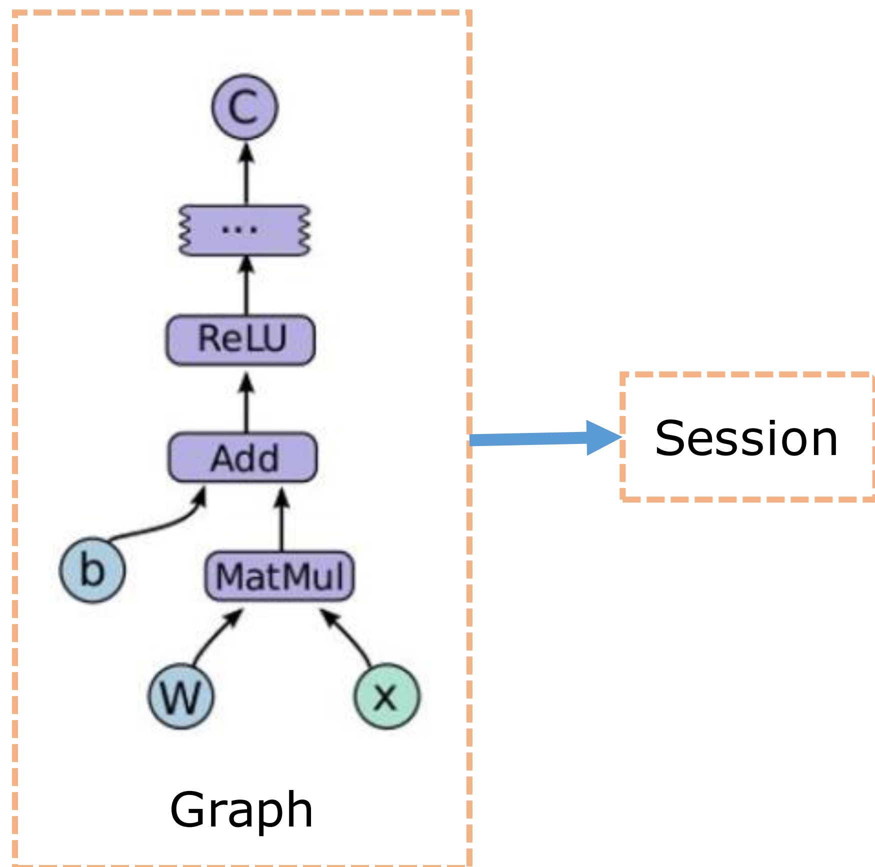
Convolved Feature

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。

01 基本概念

会话（tf.Session）

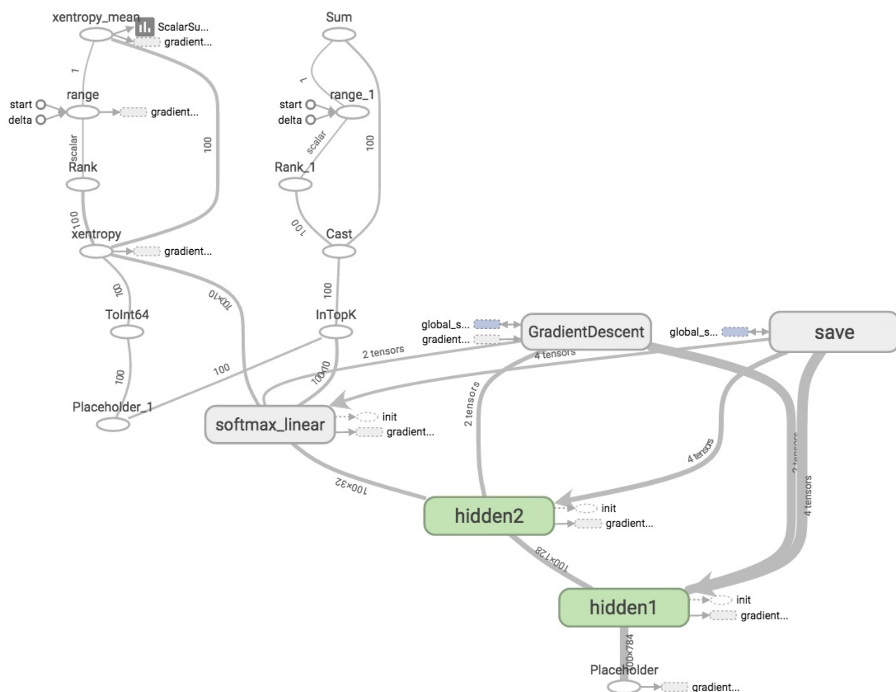


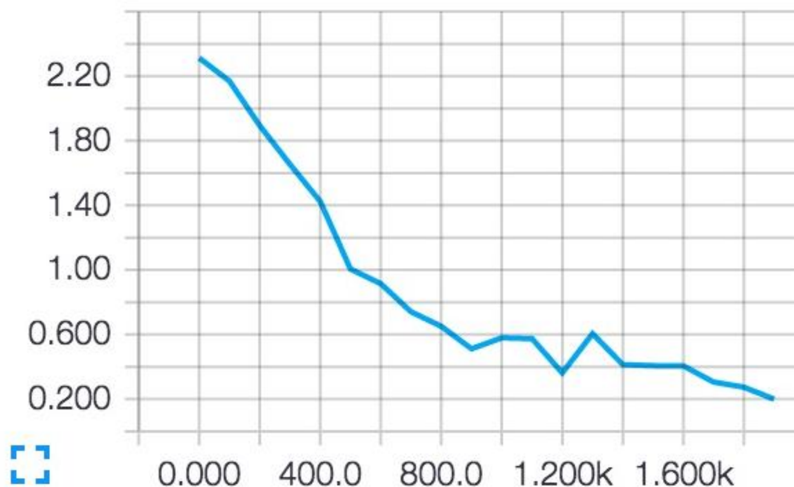
Session

- 图构建完成后要通过会话实现计算.
- 拥有具体的物理资源(如cpu、gpu)
- 提供了Operation执行和Tensor求值的环境
- 在计算Graph时,并不一定是Graph中的所有节点都被计算了,而是指定的计算节点或者该节点的输出结果被需要时

01 基本概念

可视化（tensorboard）



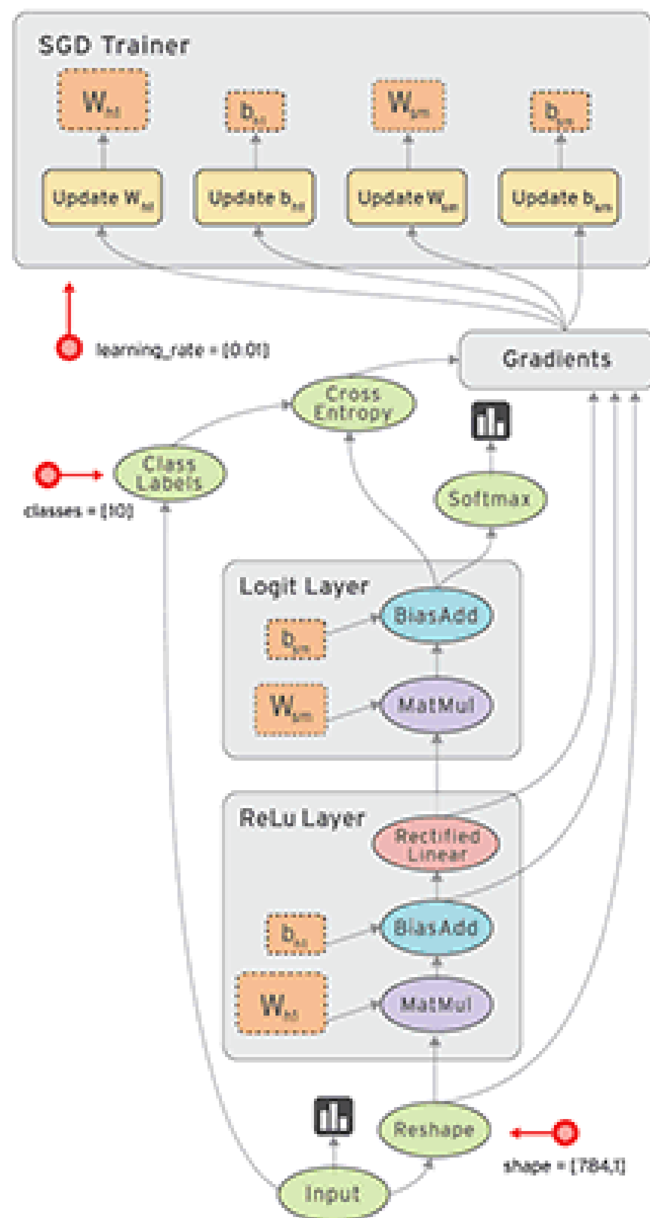


数据流图、绘制分析图、显示附加数据等。

01 基本概念

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



02 例程解析

```
# 导入模块
import numpy as np
import tensorflow as tf
```

02 例程解析

```
# 导入模块
import numpy as np
import tensorflow as tf
# 准备学习数据
train_X = np.random.normal(1, 5, 200) # 输入特征
train_Y = 0.5*train_X+2+np.random.normal(0, 1, 200) # 学习目标
L = len(train_X) # 样本量
# 定义学习超参数
epoch = 200 # 纪元数（使用所有学习数据一次为1纪元）
learn_rate = 0.005 # 学习速度
```

02 例程解析

```
#准备学习数据
train_X = np.random.normal(1, 5, 200) # 输入特征
train_Y = 0.5*train_X+2+np.random.normal(0, 1, 200) # 学习目标
L = len(train_X) # 样本量
#定义学习超参数
epoch = 200 # 纪元数（使用所有学习数据一次为1纪元）
learn_rate = 0.005 # 学习速度
#定义数据流图
temp_graph = tf.Graph()
with temp_graph.as_default():
  X = tf.placeholder(tf.float32) # 定义张量占位符
  Y = tf.placeholder(tf.float32)
  k = tf.Variable(np.random.randn(), dtype=tf.float32)
  b = tf.Variable(0, dtype=tf.float32) # 定义变量
  linear_model = k*X+b # 线性模型
  cost = tf.reduce_mean(tf.square(linear_model - Y)) # 代价函数
  optimizer = tf.train.GradientDescentOptimizer(learning_rate=learn_rate) # 梯度下降算法
  train_step = optimizer.minimize(cost) # 最小化代价函数
  init = tf.global_variables_initializer() # 使用变量全局初始化选项
```

02 例程解析

```
# 定义数据流图
temp_graph = tf.Graph()
with temp_graph.as_default():
  X = tf.placeholder(tf.float32) # 定义张量占位符
  Y = tf.placeholder(tf.float32)
  k = tf.Variable(np.random.randn(), dtype=tf.float32)
  b = tf.Variable(0, dtype=tf.float32) # 定义变量
  linear_model = k*X+b # 线性模型
  cost = tf.reduce_mean(tf.square(linear_model - Y)) # 代价函数
  optimizer = tf.train.GradientDescentOptimizer(learning_rate=learn_rate) # 梯度下降算法
  train_step = optimizer.minimize(cost) # 最小化代价函数
  init = tf.global_variables_initializer() # 使用变量全局初始化选项
# 创建会话
with tf.Session(graph=temp_graph) as sess:
  sess.run(init) # 变量初始化
  for i in range(epoch):
      sess.run(train_step, feed_dict={X: train_X, Y: train_Y}) # 运行“最小化代价函数”
      temp_cost = sess.run(cost, feed_dict={X: train_X, Y: train_Y}) # 代价函数
```

03 用法特点

1. 高度灵活(相对于Caffe): Python \ 完备的基本操作 \
2. 自动求微分: 定义模型 -> 结合损失函数
3. 多语言支持: Python \ C++ ...
4. Keras: 第三方API
5. 支持分布式
6. 可移植性: PC \ 服务器 \ 手机...