



智能计算系统

第三章 编程框架使用

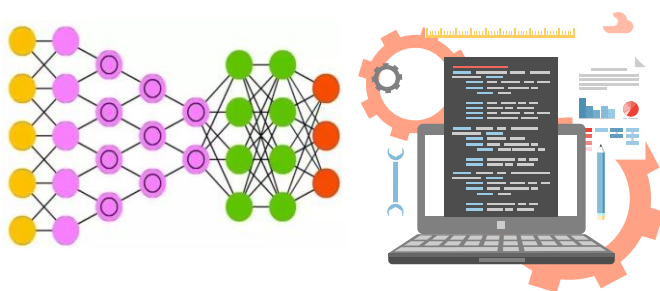
信息学院

王晶

jwang@ruc.edu.cn



Driving Example



编程框架



Bang

第三章将学习到实现深度学习算法所使用的编程框架的简单用法



为什么需要编程框架？



深度学习算法得到广泛关注，越来越多的公司、程序员需要使用深度学习算法



为什么需要编程框架？

计算误差对权重 w_{ij} 的偏导数是两次使用链式法则得到的：

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial \text{net}_j} \frac{\partial \text{net}_j}{\partial w_{ij}}$$

在右边的最后一项中，只有加权和 net_j 取决于 w_{ij} ，因此

$$\frac{\partial \text{net}_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \left(\sum_{k=1}^n w_{kj} o_k \right) = o_i.$$

神经元 j 的输出对其输入的导数就是激活函数的偏导数（这里假定使用逻辑函数）：

$$\frac{\partial o_j}{\partial \text{net}_j} = \frac{\partial}{\partial \text{net}_j} \varphi(\text{net}_j) = \varphi(\text{net}_j)(1 - \varphi(\text{net}_j))$$

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from mpl_toolkits.mplot3d import axes3d
4 from matplotlib import style
5
6 def SGD(samples, y, step_size=2, max_iter_count=1000):
7     m, var = samples.shape
8     theta = np.zeros(2)
9     y = y.flatten()
10    #进入循环内
11    loss = 1
12    iter_count = 0
13    iter_list=[]
14    loss_list=[]
15    theta1=[]
16    theta2=[]

```

有必要将算法中的常用操作封装成组件提供给程序员，以提高深度学习算法开发效率

但如果 j 是网络中任一内层，求 E 关于 o_j 的导数就不太简单了。

考虑 E 为接受来自神经元 j 的输入的所有神经元 $L = u, v, \dots, w$ 的输入的函数，

$$\frac{\partial E(o_j)}{\partial o_j} = \frac{\partial E(\text{net}_u, \text{net}_v, \dots, \text{net}_w)}{\partial o_j}$$

关于 o_j 取全微分，可以得到该导数的一个递归表达式：

$$\frac{\partial E}{\partial o_j} = \sum_{l \in L} \left(\frac{\partial E}{\partial \text{net}_l} \frac{\partial \text{net}_l}{\partial o_j} \right) = \sum_{l \in L} \left(\frac{\partial E}{\partial o_l} \frac{\partial o_l}{\partial \text{net}_l} \frac{\partial \text{net}_l}{\partial o_j} \right) = \sum_{l \in L} \left(\frac{\partial E}{\partial o_l} \frac{\partial o_l}{\partial \text{net}_l} w_{jl} \right)$$

```

23 rand1 = np.random.randint(0,m,1)
24 h = np.dot(theta,samples[rand1].T)
25 #关键点，只需要一个样本点来更新权值
26 for i in range(len(theta)):
27     theta[i] = theta[i] - step_size*(1/m)*(h - y[rand1])*samples[rand1,i]
28 #计算总体的损失精度，等于各个样本损失精度之和
29 for i in range(m):
30     h = np.dot(theta.T, samples[i])
31     #每组样本点损失的精度
32     every_loss = (1/(var*m))*np.power((h - y[i]), 2)
33     loss = loss + every_loss
34
35 print("iter_count: ", iter_count, "the loss:", loss)
36

```

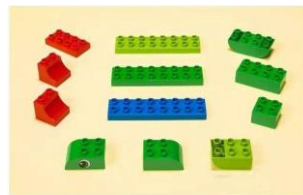
算法理论复杂

代码实现工作量大



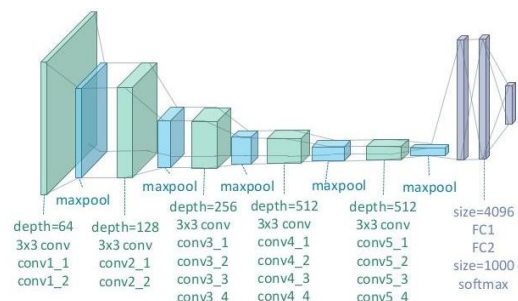
什么是编程框架

- 随着深度学习研究的深入，深度学习算法变得愈加复杂，研究人员需要花更多的时间和精力在算法的实现上
- **深度学习编程框架**：将深度学习算法中的基本操作封装成一系列组件，这一系列深度学习组件，即构成一套深度学习框架
- 编程框架能够帮助算法开发人员更简单的实现已有算法，或设计新的算法。也有助于硬件程序员更有针对性的对关键操作进行优化，使其能充分发挥硬件效率

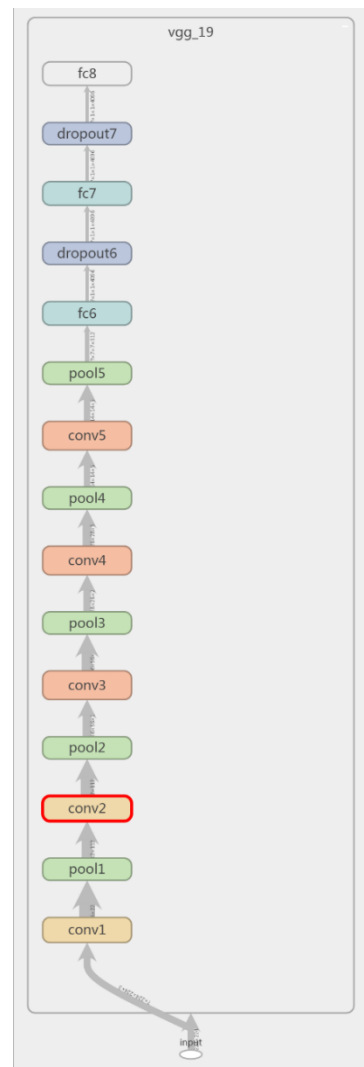




Driving example-VGGNET19



```
def build_vgg19(path):  
    net = {}  
    vgg_rawnet = scipy.io.loadmat(path)  
    vgg_layers = vgg_rawnet['layers'][0]  
    net['input'] = tf.Variable(np.zeros((1, IMAGE_HEIGHT, IMAGE_WIDTH, 3)).astype('float32'))  
    net['conv1_1'] = build_net('conv', net['input'], get_weight_bias(vgg_layers, 0))  
    net['conv1_2'] = build_net('conv', net['conv1_1'], get_weight_bias(vgg_layers, 2))  
    net['pool1'] = build_net('pool', net['conv1_2'])  
    net['conv2_1'] = build_net('conv', net['pool1'], get_weight_bias(vgg_layers, 5))  
    net['conv2_2'] = build_net('conv', net['conv2_1'], get_weight_bias(vgg_layers, 7))  
    net['pool2'] = build_net('pool', net['conv2_2'])  
    net['conv3_1'] = build_net('conv', net['pool2'], get_weight_bias(vgg_layers, 10))  
    net['conv3_2'] = build_net('conv', net['conv3_1'], get_weight_bias(vgg_layers, 12))  
    net['conv3_3'] = build_net('conv', net['conv3_2'], get_weight_bias(vgg_layers, 14))  
    net['conv3_4'] = build_net('conv', net['conv3_3'], get_weight_bias(vgg_layers, 16))  
    net['pool3'] = build_net('pool', net['conv3_4'])  
    net['conv4_1'] = build_net('conv', net['pool3'], get_weight_bias(vgg_layers, 19))  
    net['conv4_2'] = build_net('conv', net['conv4_1'], get_weight_bias(vgg_layers, 21))  
    net['conv4_3'] = build_net('conv', net['conv4_2'], get_weight_bias(vgg_layers, 23))  
    net['conv4_4'] = build_net('conv', net['conv4_3'], get_weight_bias(vgg_layers, 25))  
    net['pool4'] = build_net('pool', net['conv4_4'])  
    net['conv5_1'] = build_net('conv', net['pool4'], get_weight_bias(vgg_layers, 28))  
    net['conv5_2'] = build_net('conv', net['conv5_1'], get_weight_bias(vgg_layers, 30))  
    net['conv5_3'] = build_net('conv', net['conv5_2'], get_weight_bias(vgg_layers, 32))  
    net['conv5_4'] = build_net('conv', net['conv5_3'], get_weight_bias(vgg_layers, 34))  
    net['pool5'] = build_net('pool', net['conv5_4'])  
    return net
```



神经网络

TensorFlow实现



```
_mlu_entry__ void Proposal(...) {  
    ...  
    __nram__ half scores[...];  
    __nramset__ half(scores, ...);  
    ...  
    __bang_maxpool(...);  
    ...  
}
```

Bang实现算子功能，嵌入编程框架中

在计算设备运行

生成计算图



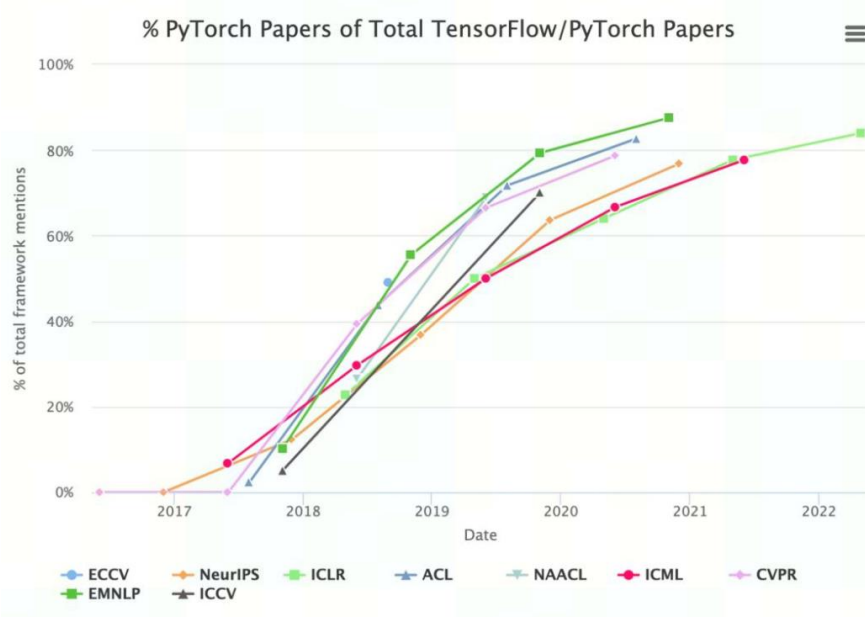
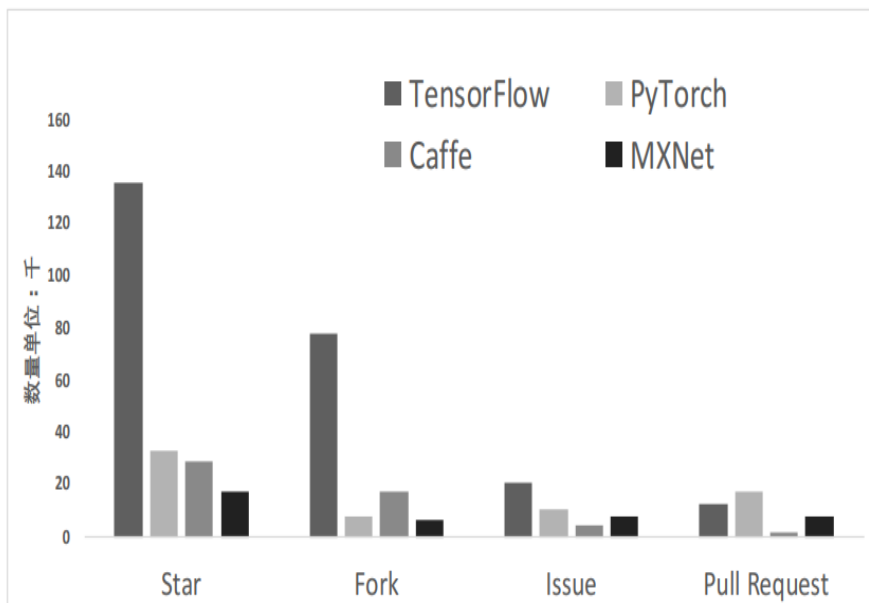
国内外主流编程框架

序号	框架名称	发布者	首次发布时间	logo
1	PyTorch	Facebook (Meta)	2017	
2	TensorFlow	Google	2015	
3	Keras	Google	2015	
4	Caffe	BVLC	2013	
5	PaddlePaddle	百度	2018	
6	OneFlow	一流	2020	
7	Mindspore	华为	2019	



主流深度学习编程框架

- 工业界TensorFlow使用人数多，github上面star多
- EMNLP、ACL、ICLR三家顶会的PyTorch的占比已经超过80%，这一占比数字在其他会议中也都保持在70%之上





TensorFlow概述

- TensorFlow---AlphaGo背后的力量



- 谷歌团队于2015年开发并开源，Github上活跃度最高的深度学习编程框架
- TensorFlow及其变种可以工作于各种类型的异构系统，包括手机、平板电脑等移动设备，数百台机器和数千种计算设备（如GPU）的大规模分布式系统



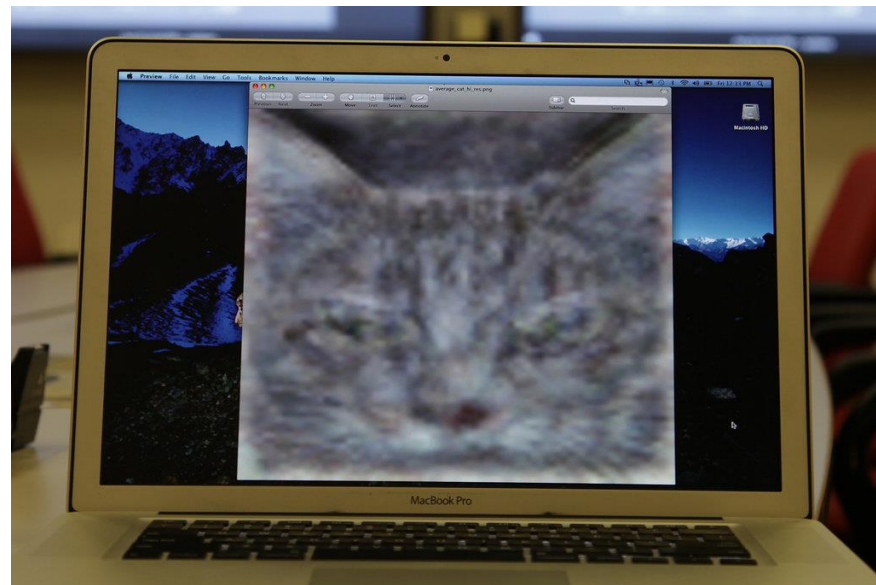
谷歌第一代分布式深度学习平台DistBelief

- 谷歌大脑项目的早期成果，2011年开发完成，是第一代可扩展的分布式深度学习平台，可用于深度神经网络的训练和预测。
- 谷歌团队使用DistBelief进行了各种各样的研究，包括无监督学习、图像分类、视频分类、语音识别、序列预测、行人检测、强化学习等。
- 该平台将Google语音识别率提高了25%，在Google Photos 中建立了图片搜索，并驱动了 Google 的图片字幕匹配实验



谷歌第一代分布式深度学习平台DistBelief

- 谷歌基于DistBelief建立了大规模的神经网络，开展了著名的“**猫脸识别**”实验
- 在YouTube视频中提取出的1000万张图像中，在没有外界干涉的条件下，通过自我学习，识别出了猫脸
- 仅可用于神经网络，且和Google内部基础产品联系紧密，在实际配置和应用中存在局限性





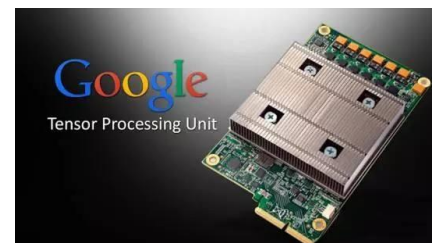
谷歌第二代大规模机器学习系统TensorFlow

- ▶ 作为DistBelief的继任者，由谷歌团队开发并于2015年11月开源的深度学习框架，用于实施和部署大规模机器学习模型
- ▶ 相比DistBelief，具有以下优点：
 - ▶ 支持多种高级语言作为输入：Python、C、C++、Java、Go
 - ▶ 更灵活的编程模型
 - ▶ 更高的性能
 - ▶ 支持在更广泛的异构硬件平台上进行训练和使用更大规模的神经网络模型



TensorFlow的应用

- Google多个产品（Gmail, Google Play Recommendation, Search, Translate, Map等）
- AlphaGo
- TPU2.0/3.0
- IBM PowerAI
- Intel Movidius Myriad
- 寒武纪MLU100/270





安装与技术资料获取

- www.tensorflow.org
TensorFlow官网
- <https://github.com/tensorflow/tensorflow>
官方Github仓库
- <http://web.stanford.edu/class/cs20si/>
斯坦福大学TensorFlow系列课程
- <https://developers.google.cn/machine-learning/crash-course/>
谷歌官方视频教程



PyTorch起源-Torch

- Torch是瑞士亚普研究所（IDIAP）在2002年发布的一款机器学习框架，采用LuaJIT脚本编程语言为接口，内核采用C/C++来实现
- ▶ 核心是易于使用的神经网络和优化库，同时在实现复杂的神经网络拓扑结构方面具有最大的灵活性
- ▶ 在GPU上具有较高的性能
- ▶ 作者：Ronan Collobert ,
Samy Bengio , Johnny
Mariéthoz





PyTorch简介

- PyTorch=Py+Torch
- 2017年Facebook AI Research开源
- PyTorch是一个基于Torch的Python开源机器学习库，用于自然语言处理等应用程序，具有强大的GPU加速的张量计算（如Numpy）能力
- 产生背景：
 - 编程语言提供了面向深度学习的高效编程库（NumPy、Eigen、Torch等）
 - Python开源生态蓬勃发展



设计原则

- 基于Python
- 把研究人员放在首位
- 易用
- 高性能
- 内部实现简单，节省学习时间

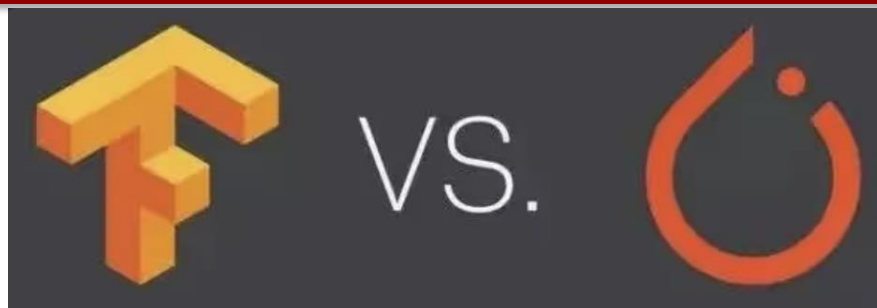
Framework	Throughput (higher is better)					
	AlexNet	VGG-19	ResNet-50	MobileNet	GNMTv2	NCF
Chainer	778 \pm 15	N/A	219 \pm 1	N/A	N/A	N/A
CNTK	845 \pm 8	84 \pm 3	210 \pm 1	N/A	N/A	N/A
MXNet	1554 \pm 22	113 \pm 1	218 \pm 2	444 \pm 2	N/A	N/A
PaddlePaddle	933 \pm 123	112 \pm 2	192 \pm 4	557 \pm 24	N/A	N/A
TensorFlow	1422 \pm 27	66 \pm 2	200 \pm 1	216 \pm 15	9631 \pm 1.3%	4.8e6 \pm 2.9%
PvTorch	1547 \pm 316	119 \pm 1	212 \pm 2	463 \pm 17	15512 \pm 4.8%	5.4e6 \pm 3.4%

Table 1: Training speed for 6 models using 32bit floats. Throughput is measured in images per second for the AlexNet, VGG-19, ResNet-50, and MobileNet models, in tokens per second for the GNMTv2 model, and in samples per second for the NCF model. The fastest speed for each model is shown in bold.



学习资料

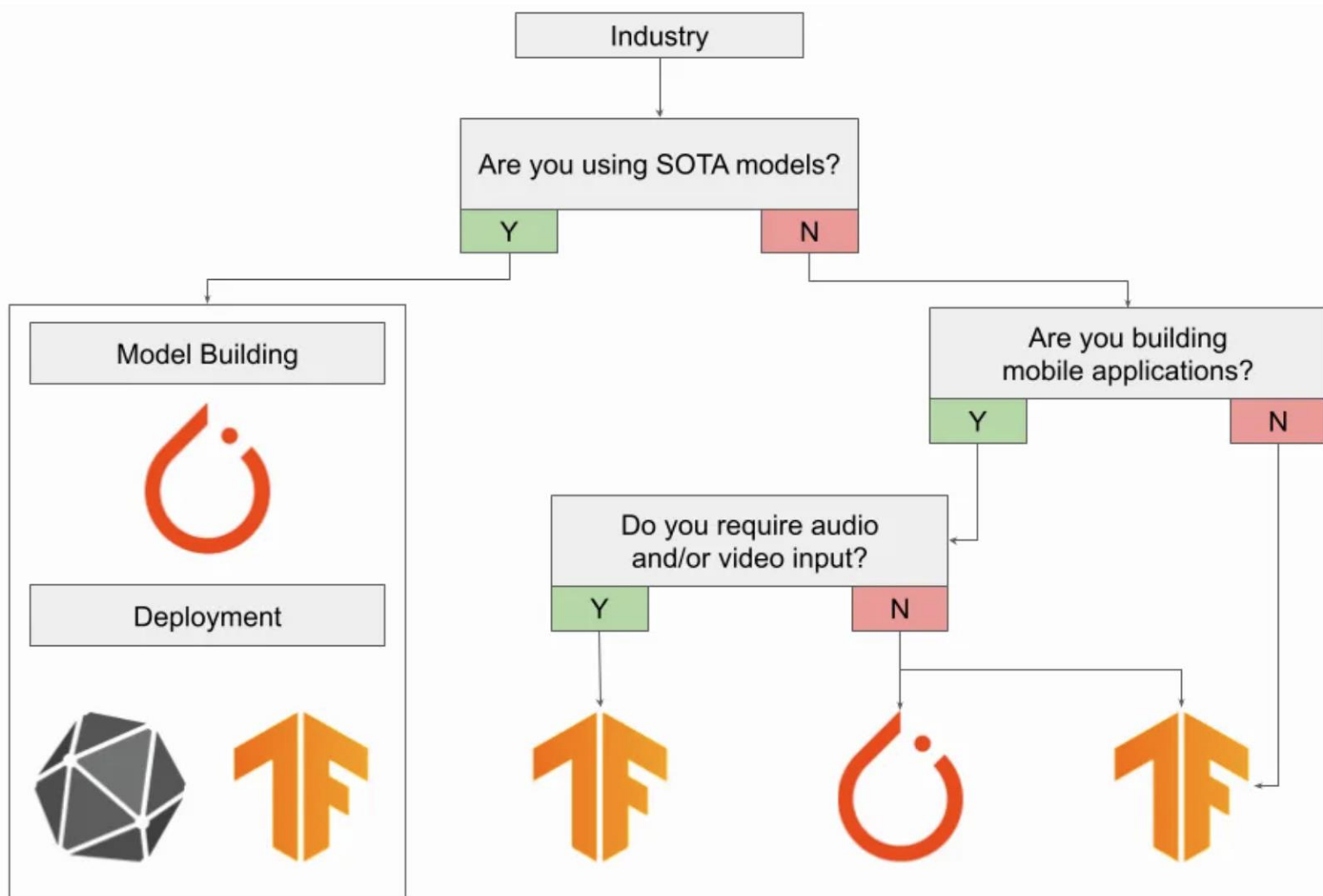
- 一些有趣的应用
 - <https://github.com/ritchieng/the-incredible-pytorch>
- 斯坦福课程关于Pytorch的介绍
 - <https://cs230.stanford.edu/blog/pytorch/>
- 官方github
 - <https://github.com/pytorch>



声明式编程 使用者要求高	命令式编程 调试方便，初学者友好
有强大的移植性 倾向于工业应用领域	语法简单，开发周期短 倾向于科研领域
静态图 工业部署方便 TensorBoard等生态好	动态图 运行效率比较低

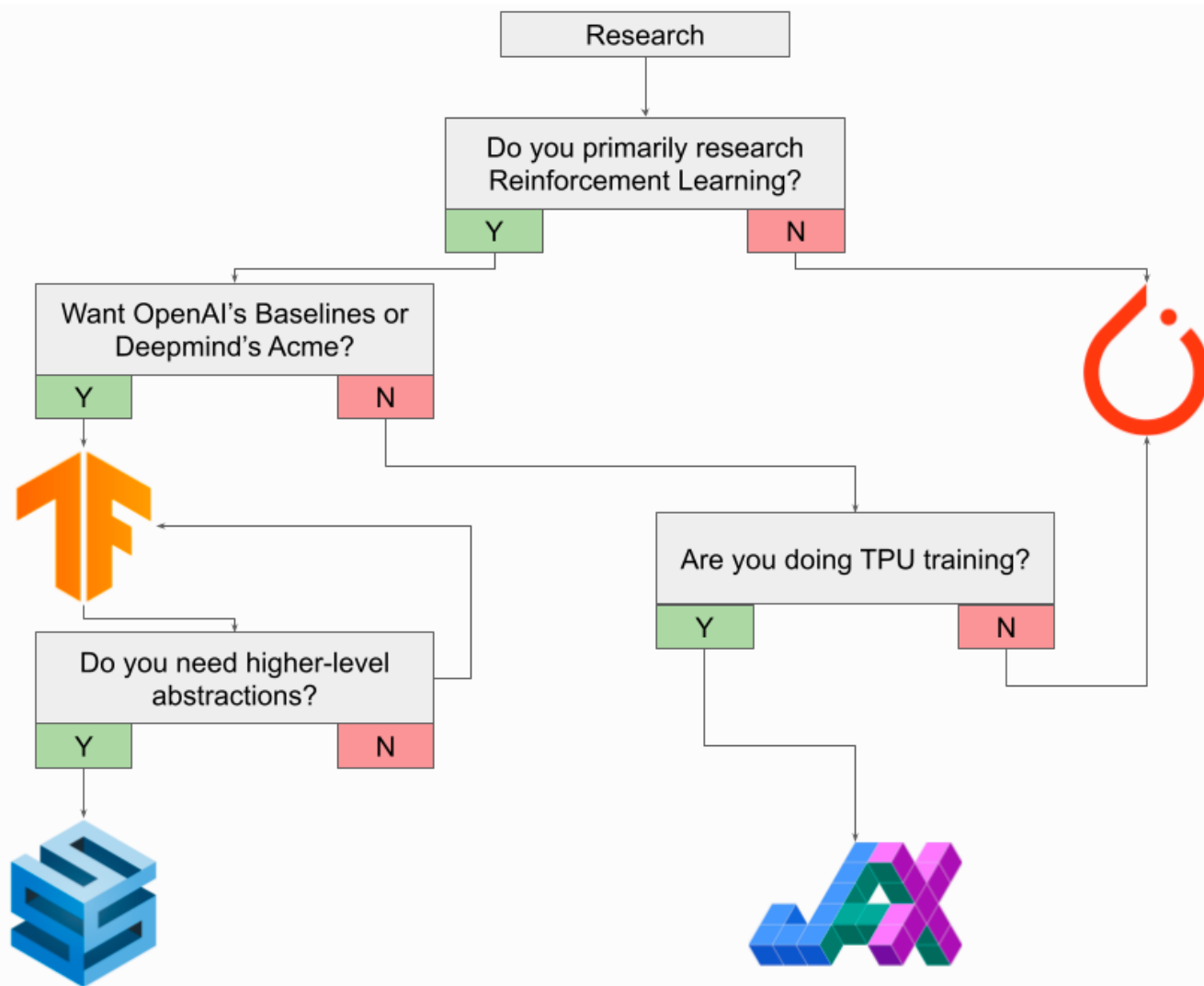


怎么选？-如果我在工业界



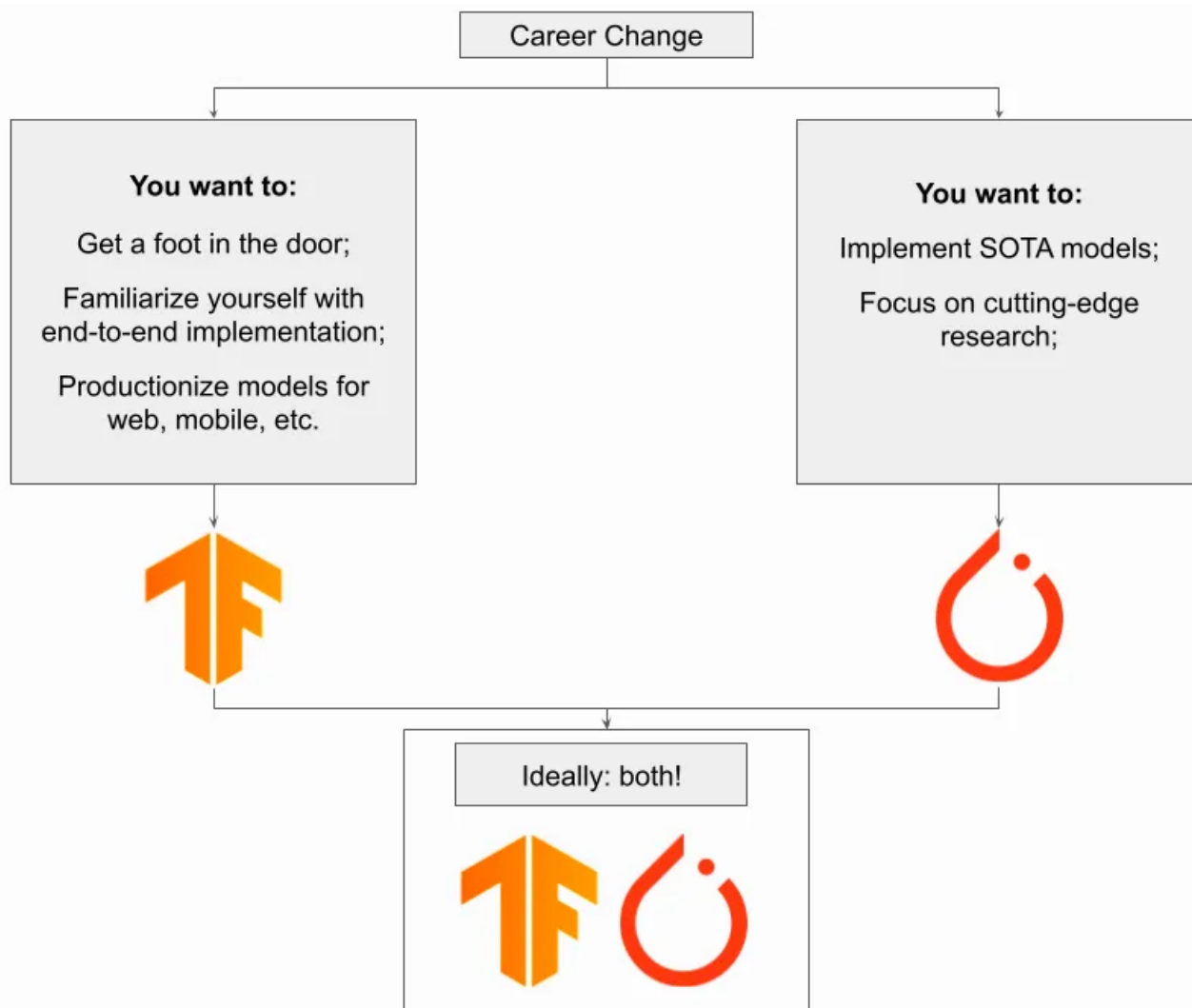


怎么选？-如果我在学术界





怎么选？-如果我求职



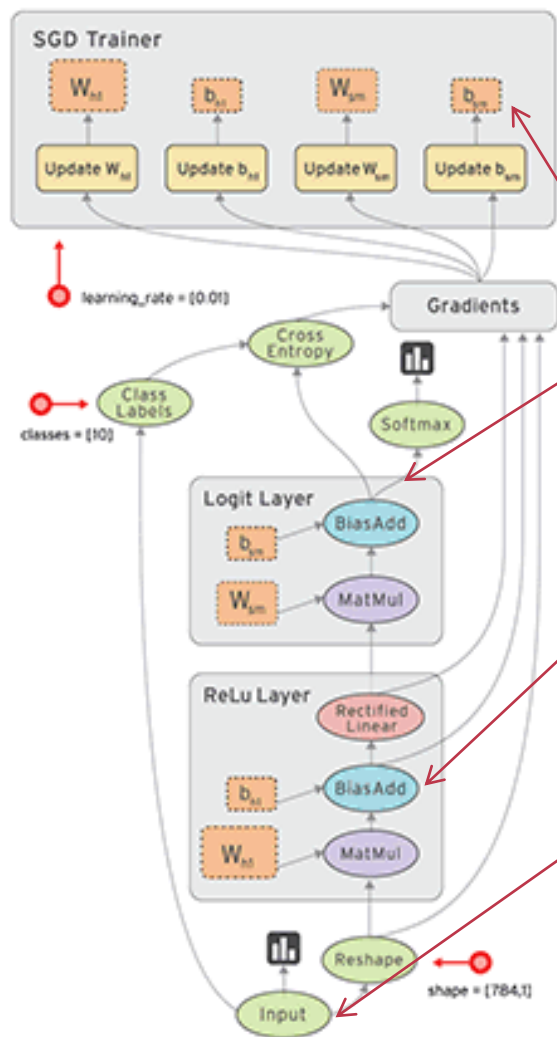


TensorFlow编程模型及基本概念

- 命令式编程与声明式编程
 - **命令式编程**：关注程序执行的具体步骤，计算机按照代码中的顺序一步一步执行具体的运算。整体优化困难。
 - 示例：交互式UI程序、操作系统
 - **声明式编程**：告诉计算机想要达到的目标，不指定具体的实现步骤，而是通过函数、推论规则等描述数据之间的关系，优化较容易。
 - 示例：人工智能、深度学习



TensorFlow中的几个基本概念

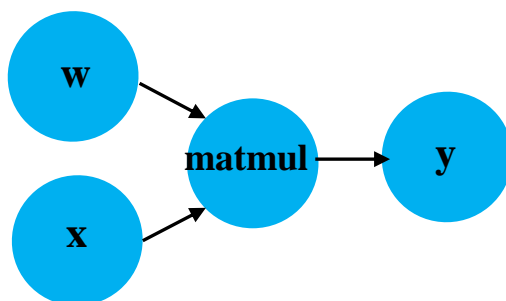


- 1、使用**计算图**来表示机器学习算法中所有的计算和状态
- 2、将所有的数据建模成**张量 (tensor)**
- 3、具体计算操作运行在**会话 (session)** 环境中
- 4、将多种类型的计算定义为**操作 (operation)**
- 5、通过**变量 (variable)** 存储计算图中的有状态参数，如模型参数
- 6、通过**占位符 (placeholder)** 将张量传递到会话中
- 7、通过**队列 (Queue)** 处理数据读取和计算图的异步执行



1、计算图

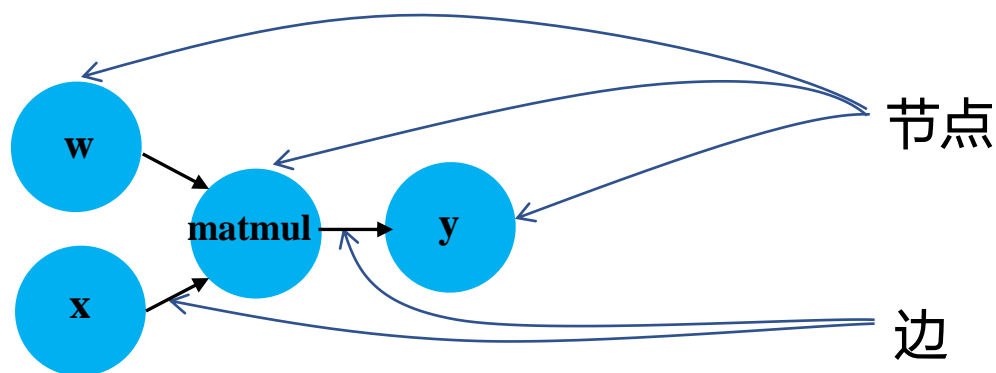
- TensorFlow中使用有向图来描述计算过程。有向图中包含一组**节点**和**边**
- 支持通过多种高级语言来构建计算图（C++/Python）
- 计算图对应了神经网络的结构
- 示例： $y=w*x$





节点和边

- **节点**一般用来表示各类操作，包括数学运算、变量读写、数据填充等，也可以表示输入数据、模型参数、输出数据
- **边**表示“节点”之间的输入输出关系。分为两类：
 - 一类是**传递具体数据**的边。传递的数据即为张量（tensor）。
 - 一类是**表示节点之间控制依赖关系**的边。这类边不传递数据，只表示节点执行的顺序：必须前序节点计算完成，后序节点才开始计算。

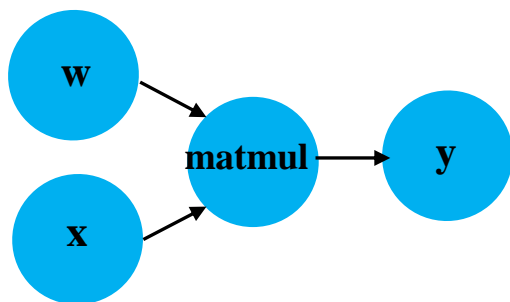




TensorFlow程序简单示例

- TensorFlow 程序一般分为两部分：构建计算图（Line3-6）、执行计算图（Line8-10）

构造计算图



```
1 import tensorflow as tf
2
3 x = tf.constant([[3., 3.]])
4 w = tf.constant([[2.],[2.]])
5
6 y = tf.matmul(x, w)
7
8 with tf.Session() as sess:
9     result = sess.run(y)
10    print(result)
```

创建会话，执行运算



- TensorFlow 1.x
 - 静态图，方便对整个计算图做全局优化，性能较高；但调试困难，影响开发效率
- TensorFlow 2.x
 - 动态图，调试简单，更适合快速开发；但运行效率低于静态图方法



2、操作

- 计算图中的每个计算节点即代表一个**操作（operation）**，其接收0个或多个tensor作为输入，产生0个或多个tensor作为输出
- 操作对应了神经网络中的具体计算

▶ 右图中包含了三个操作：

1) 给a赋常数值

2) 给b赋常数值

3) a、b相乘得到y

```
1 import tensorflow as tf
2
3 a = tf.constant([[3., 3.]])
4 b = tf.constant([[2.],[2.]])
5
6 y = tf.matmul(a, b)
7
8 with tf.Session() as sess:
9     result = sess.run(y)
10    print(result)
11
```



操作的主要属性

属性名	功能说明
tf.operation.name	操作的名称
tf.operation.type	操作的类型，如add
tf.operation.inputs	操作的输入
tf.operation.outputs	操作的输出
tf.operation.control_inputs	该操作的控制依赖列表
tf.operation.device	执行该操作所使用的设备
tf.operation.graph	操作所属的计算图
tf.operation.traceback	实例化该操作时的调用栈



TensorFlow中的常用操作

操作类型	常用算子
标量运算	add、subtract、multiply、div、greater、less、equal、abs、sign、square、pow、log、sin、cos
矩阵运算	matmul、matrix_inverse、matrix_determinant、matrix_transpose
逻辑操作	logical_and、is_finite
神经网络运算	convolution、max_pool、bias_add、softmax、dropout、sigmoid、relu
储存、恢复	save、restore
初始化操作	zeros_initializer、random_normal_initializer、orthogonal_initializer
随机运算	random_gamma、multinomial、random_normal、random_shuffle



3、张量（tensor）

- TensorFlow中，张量是计算图上的数据载体，用张量统一表示所有的数据，张量在计算图的节点之间传递
- 张量中并没有实际保存数据，而仅是对计算结果的引用，对应了神经网络中各个节点之间流动的数据
- 张量可以看做是 n 维的数组，数组的维数即为张量的阶数

阶数	对应数据形式
0	标量
1	向量
2	矩阵
n	n 维数组

- 例如，一张RGB图片可以表示成三阶张量，多张图片组成的数据集可以表示成四阶张量



tensor的常用属性

属性名	含义
dtype	tensor存储的数据类型
shape	tensor各阶的长度
name	tensor在计算图中的名称
op	计算出此tensor的操作
device	计算出此tensor所用的设备名
graph	包含此tensor的计算图



tensor中支持的dtype类型

TensorFlow数据类型	说明
int8/int16/int32/int64	8位/16位/32位/64位有符号整数
float16/float32/float64	半精度/单精度/双精度浮点数
bfloat16	裁短浮点数
uint8/uint16/uint32/uint64	8位/16位无符号整数
bool	布尔值
string	字符串
complex64/complex128	单精度/双精度复数
qint8/qint16/qint32	量化的8位/16位/32位有符号整数
quint8/quint16	量化的8位/16位无符号整数

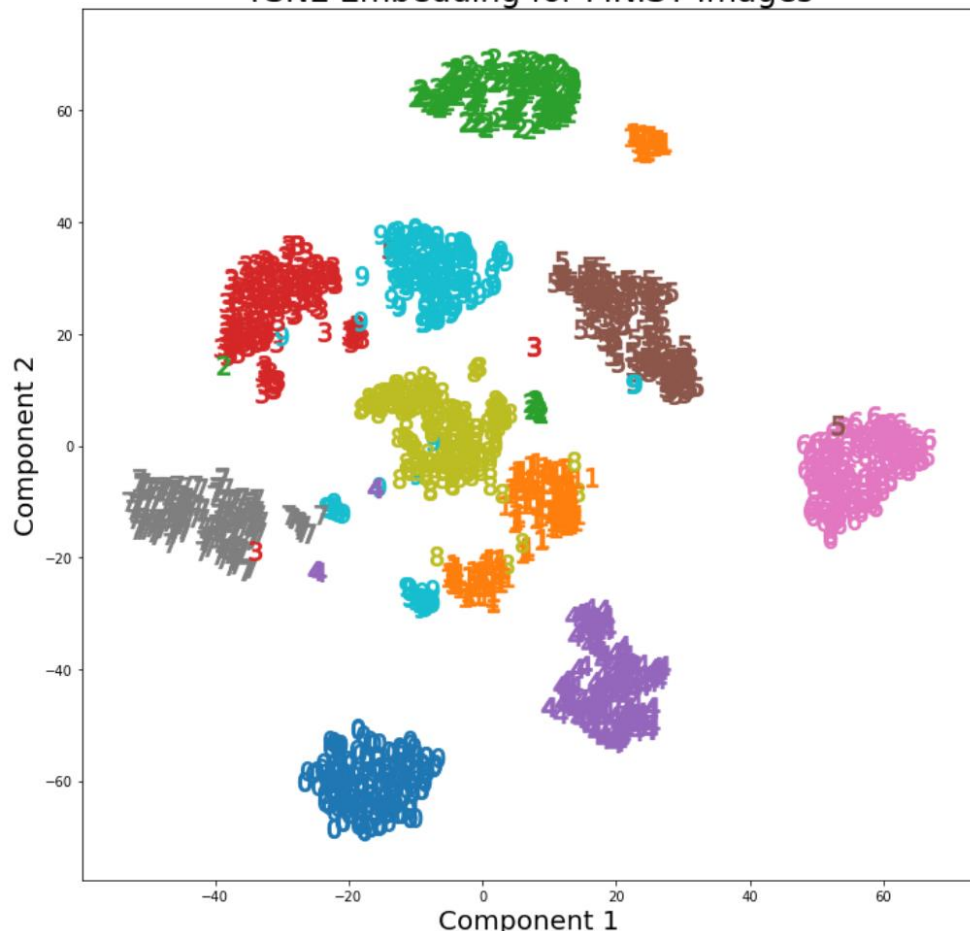
► 示例：`a=tf.constant([2,3],dtype=tf.float32)`



深度学习为什么不需要全部float32

- 深度学习算法特性

TSNE Embedding for MNIST Images



t-SNE可视化深度学习数据

- 这是一个典型的手写体识别任务的网络特征（神经元）二维可视化图
- 不同类别（颜色）的数据间距
- “大间距”意味着 容忍非精确计算



训练时不需要高位宽

- CNN网络，分类、检测、分割任务下的低位宽训练

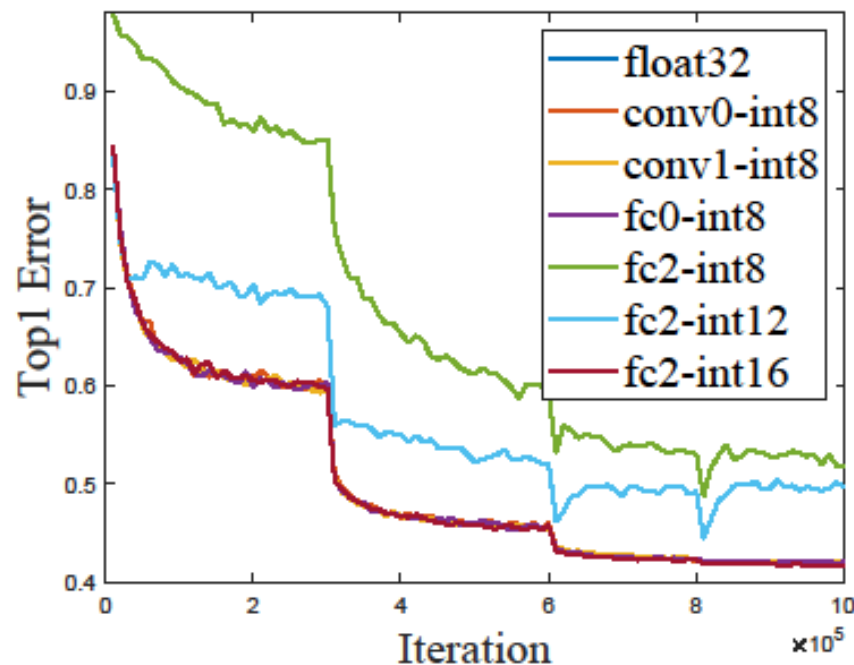
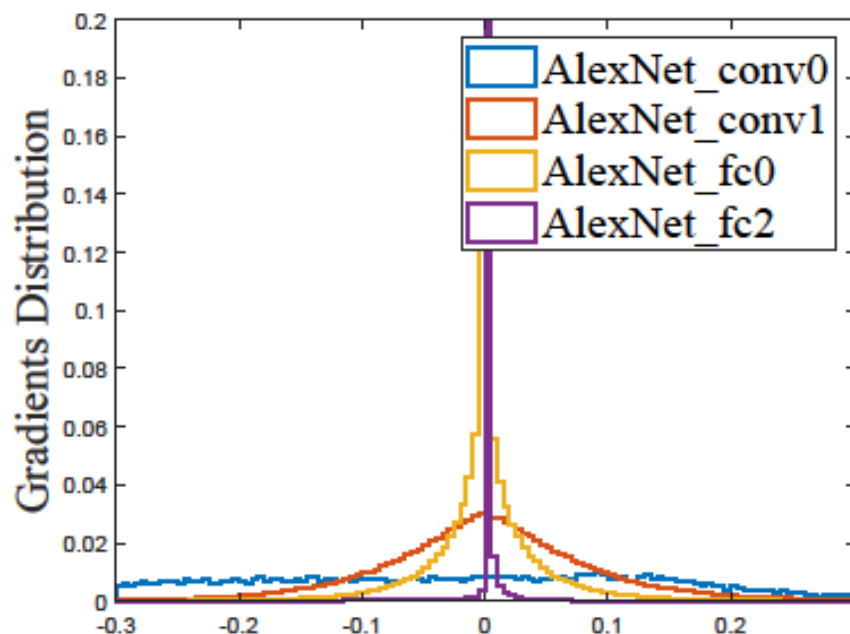
精度和估计精度对比 梯度精度 权重精度

Classification Network	float32 Acc	Adaptive Acc	Activation int8	Weight int8	Activation Gradient int8	Gradient int16
AlexNet	58.0	58.22	100%	100%	22.5%	77.5%
VGG16	71.0	70.6	100%	100%	31.3%	68.7%
Inception_BN	73.0	72.8	100%	100%	4.5%	95.5%
ResNet50	76.4	76.2	100%	100%	0.8%	99.2%
ResNet152	78.8	78.2	100%	100%	1.7%	98.3%
MobileNet v2	72.0	70.5	100%	100%	0.7%	99.2%
SSD Detection Network	float32 mAP	Adaptive mAP	Activation int8	Weight int8	Activation Gradient int8	Gradient int16
COCO_VGG	43.1	42.4	100%	100%	31.4%	68.6%
VOC_VGG	77.3	77.2	100%	100%	34.3%	65.7%
VOC_ResNet101	73.4	73.1	100%	100%	7.4%	83.6%
IMGDET_ResNet101	44.1	44.4	100%	100%	28.6%	71.4%
Segmentation Network	float32 meanIoU	Adaptive meanIoU	Activation int8	Weight int8	Activation Gradient int8	Gradient int16
deeplab-v1	70.1	69.9	100%	100%	1.0%	99.0%



数据位宽与算法精度

- 不同数据的位宽需求是不同的



每层数据都有其保持网络收敛的最低位宽需求，
每层数据的位宽需求与数据分布之间存在关系



tensor的shape属性

- 表示tensor每一阶的长度

1

1	1	1
---	---	---

1	1	1
1	1	1
1	1	1

1	1	1
1	1	1
1	1	1

1	1	1
1	1	1
1	1	1

阶数

0阶

1阶

2阶

3阶

对应数
据形式

标量

向量

矩阵

三维数组

shape

()

(3)

(3,3)

(2,3,3)



tensor的device属性

- `tf.device(device_name)`指定计算出此tensor所用的设备名
- TensorFlow不区分CPU，所有CPU均使用`/cpu:0`作为设备名称
- 用`/gpu:n`表示第n个GPU设备，用`/mlu:n`表示第n个深度学习处理器
- 示例程序：

```
1 import tensorflow as tf
2
3 with tf.device('/cpu:0'):
4     a = tf.constant([[3.,3.]])
5     b = tf.constant([[2.],[2.]])
6
7 with tf.device('/mlu:0'):
8     y = tf.matmul(a,b)
9
10 with tf.Session() as sess:
11     result = sess.run(y)
12     print(result)
13
```

▶ Line 3: 指定使用CPU来执行Line 4-5

▶ Line 7: 指定使用第0号MLU来执行Line 8



Tensor的常用操作(op)命令

函数名称	功能
<code>tf.shape(tensor)</code>	返回tensor的shape值
<code>tf.to_double(x,name='ToDouble')</code> <code>tf.to_float(x,name='ToFloat')</code> <code>tf.to_int32(x,name='ToInt32')</code> <code>tf.to_int64(x,name='ToInt64')</code>	将x转为64位浮点类型 将x转为32位浮点类型 将x转为32位整型 将x转为64位整型
<code>tf.cast(x,dtype)</code>	将x转换为dtype类型数据
<code>tf.reshape(tensor,shape)</code>	修改tensor各阶的长度为shape 如：假设 <code>a=[1,2,3,4]</code> ，则 <code>tf.reshape(a,[2,2])</code> 命令将输出： <code>[[1,2],[3,4]]</code>
<code>tf.slice(input,begin,size)</code>	从由begin指定位置开始的input中提取一个尺寸为size的切片
<code>tf.split(value,num_or_size_splits,axis)</code>	沿着第axis阶对value进行切割，切割成num_or_size_splits份
<code>tf.concat(values,axis)</code>	沿着第axis阶对value进行连接操作



4、会话 (session)

- TensorFlow中的计算图描述了计算执行的过程，但并没有真正给输入赋值并执行计算
- 真正的神经网络计算过程需要在TensorFlow程序的session部分中定义并执行

- ▶ **session**为程序提供求解张量、执行操作的运行环境。将计算图转化为不同设备上的执行步骤

```
1 import tensorflow as tf
2
3 with tf.device('/cpu:0'):
4     a = tf.constant([[3., 3.]])
5     b = tf.constant([[2.],[2.]])
6
7 with tf.device('/gpu:0'):
8     y = tf.matmul(a, b)
9
10 with tf.Session() as sess:
11     result = sess.run(y)
12     print(result)
13
```

session



创建会话（tf.Session）

- Session的输入参数：

参数	功能说明
target	会话连接的执行引擎，默认为进程内引擎
graph	执行计算时加载的计算图。默认值为当前代码中唯一的计算图。当代码中定义了多幅计算图时，使用graph指定待加载的计算图
config	指定相关配置项，如设备数量、并行线程数、GPU配置参数等

- 示例：

```
sess=tf.Session(target='',graph=None,config=None)
```



执行会话

- 基于计算图和输入数据，求解张量或执行计算
- 输入参数：

参数	功能说明
<code>fetches</code>	本会话需计算的张量或操作
<code>feed_dict</code>	指定会话执行时需填充的张量或操作，及对应的填充数据
<code>options</code>	设置会话运行时的控制选项
<code>run_metadata</code>	设置会话运行时的非张量信息输出



关闭会话

- 会话的执行会占用大量硬件资源，因此会话结束时需要关闭会话，以释放这些资源
- 关闭会话的两种方式

1) 使用close语句显式关闭会话

```
1 sess.close()
```

2) 使用with语句隐式关闭会话

```
1 import tensorflow as tf
2
3 a = tf.constant([[3., 3.]])
4 b = tf.constant([[2.],[2.]])
5
6 y = tf.matmul(a, b)
7
8 with tf.Session() as sess:
9     result = sess.run(y)
10    print(result)
11
```



session.run与tensor.eval的区别

- tensor.eval()输入参数:

参数名	说明
feed_dict	指定需填充的张量或操作，及对应的填充数据
session	指定求解此tensor或操作的会话

- tensor.eval()函数使用前，均需要显式指定求解张量或操作的会话，如用with语句定义会话
- 处理单个tensor时， tensor.eval()与session.run()等价
- session.run可以一次进行多个tensor或操作的计算



简单程序示例

- `tensor.eval()`: 每次仅能计算一个tensor
- `sess.run()`: 每次可计算多个tensor

```
1 import tensorflow as tf
2
3 a = tf.constant([[1.0,2.0]])
4 b = tf.constant([[3.0],[4.0]])
5 c = tf.constant([[5.0],[6.0]])
6
7 y1 = tf.matmul(a,b)
8 y2 = tf.matmul(a,c)
9
10 with tf.Session() as sess:
11     y1.eval()
12     y2.eval()
13     sess.run([y1,y2])
```



5、变量

- ▶ 大多数计算中计算图被执行多次，每次执行后其中的值即被释放
- ▶ 变量 (variable) 是计算图中的一种有状态节点，用于在多次执行同一计算图时存储并更新指定参数，对应了机器学习或深度学习算法中的模型参数
- ▶ 作为有状态节点，其输出由输入、节点操作、节点内部已保存的状态值共同作用



变量的常用属性

属性名	含义
dtype	变量的数据类型
shape	变量的长度
name	变量在计算图中的名称
op	变量操作
device	存储此变量所用的设备名
graph	包含此变量的计算图
initialized_value	变量的初值
initializer	为变量赋值的初始化操作
trainable	是否在训练时被优化器更新



常用于构建变量的TensorFlow操作

TensorFlow操作	说明
<code>tf.zeros()</code>	产生一个全为0的张量
<code>tf.ones()</code>	产生一个全为1的张量
<code>tf.random_normal()</code>	产生正态分布的随机数
<code>tf.truncated_normal()</code>	从截断的正态分布中输出随机数
<code>tf.random_uniform()</code>	产生满足平均分布的随机数
<code>tf.random_gamma()</code>	产生满足Gamma分布的随机数
<code>tf.fill()</code>	产生一个全为给定值的张量
<code>tf.constant()</code>	产生常量
<code>variable.initialized_value()</code>	产生一个变量的初值



更新变量

- 变量是计算图中的有状态节点，其输出受到输入、操作、节点内部已保存状态的共同影响
- 变量更新可以通过优化器自动更新完成，也可以通过自定义方法强制赋值更新

更新函数	说明
强制赋值更新	
<code>tf.assign()</code>	更新变量的值
<code>tf.assign_add()</code>	加法赋值
<code>tf.assign_sub()</code>	减法赋值
自动更新	
<code>tf.train.**Optimizer</code>	使用多种优化方法自动更新参数



6、占位符(placeholder)

- 训练神经网络模型时需要大量的样本输入，如果每个输入都用常量表示，则每个常量都需要在计算图中增加一个节点，最终的计算图会非常大
- 计算图表达的是计算的拓扑结构，在向计算图填充数据前，计算图并没有真正执行运算
- TensorFlow使用占位符来构建计算图中的样本输入节点，而不需要实际分配数据
- 占位符本身没有初始值，只是在程序中分配了内存
- 使用占位符则只会在计算图中增加一个节点，并只在执行时向其填充数据



- tf.placeholder()的输入参数

输入参数	说明
name	在计算图中的名字
dtype	填充数据的数据类型
shape	填充数据的shape值

```
#因为需要重复输入x，而每建一个x就会生成一个节点，计算图的效率会低。所以使用占位符  
x=tf.placeholder(tf.float32,shape=(1,2))
```

- 使用时需与feed_dict参数配合，用feed_dict提交数据，进行参数传递

```
#运行y时将占位符填上，feed_dict为字典，变量名不可变  
y_1=sess.run(a,feed_dict={x:[[0.7,0.9]]})
```



7、队列(Queue)

- TensorFlow提供了队列（queue）机制，通过多线程将读取数据与计算数据分开
- 队列是一种有状态的操作机制，用来处理数据读取
- 为了加快训练速度，我们可以采用多个线程读取数据，一个线程消耗数据
- 队列操作包含了入队、出队操作
- TensorFlow 提供多种队列机制，如FIFOQueue 和 RandomShuffleQueue



FIFOQueue

- 先入先出队列，支持入队、出队操作
- 入队操作将输入输出放到FIFOQueue队尾，出队操作将FIFOQueue队列中首个元素取出
- 当队列满时入队操作会被阻塞，当队列为空时出队操作会被阻塞

```
1 import tensorflow as tf
2
3 # 创建一个先进先出队列，指定队列中可以保存三个元素，并指定类型为float
4 q = tf.FIFOQueue(3, 'float')
5 # 使用enqueue_many函数来初始化队列中的元素
6 init = q.enqueue_many([0., 0., 0.],)
7 # 使用Dequeue函数将队列中的第一个元素出队列。这个元素值，将被存在变量x中
8 x = q.dequeue()
9 y = x + 1
10 # 将加1后的值在重新加入队列中
11 q_inc = q.enqueue([y])
12
13 init.run()
14 q_inc.run()
15 q_inc.run()
16 q_inc.run()
17 q_inc.run()
```



Client

```
q = tf.FIFOQueue(3, "float")  
init = q.enqueue_many([[0.,0.,0.]])
```

```
x = q.dequeue()  
y = x+1  
q_inc = q.enqueue([y])
```

```
init.run()  
q_inc.run()  
q_inc.run()  
q_inc.run()  
q_inc.run()
```

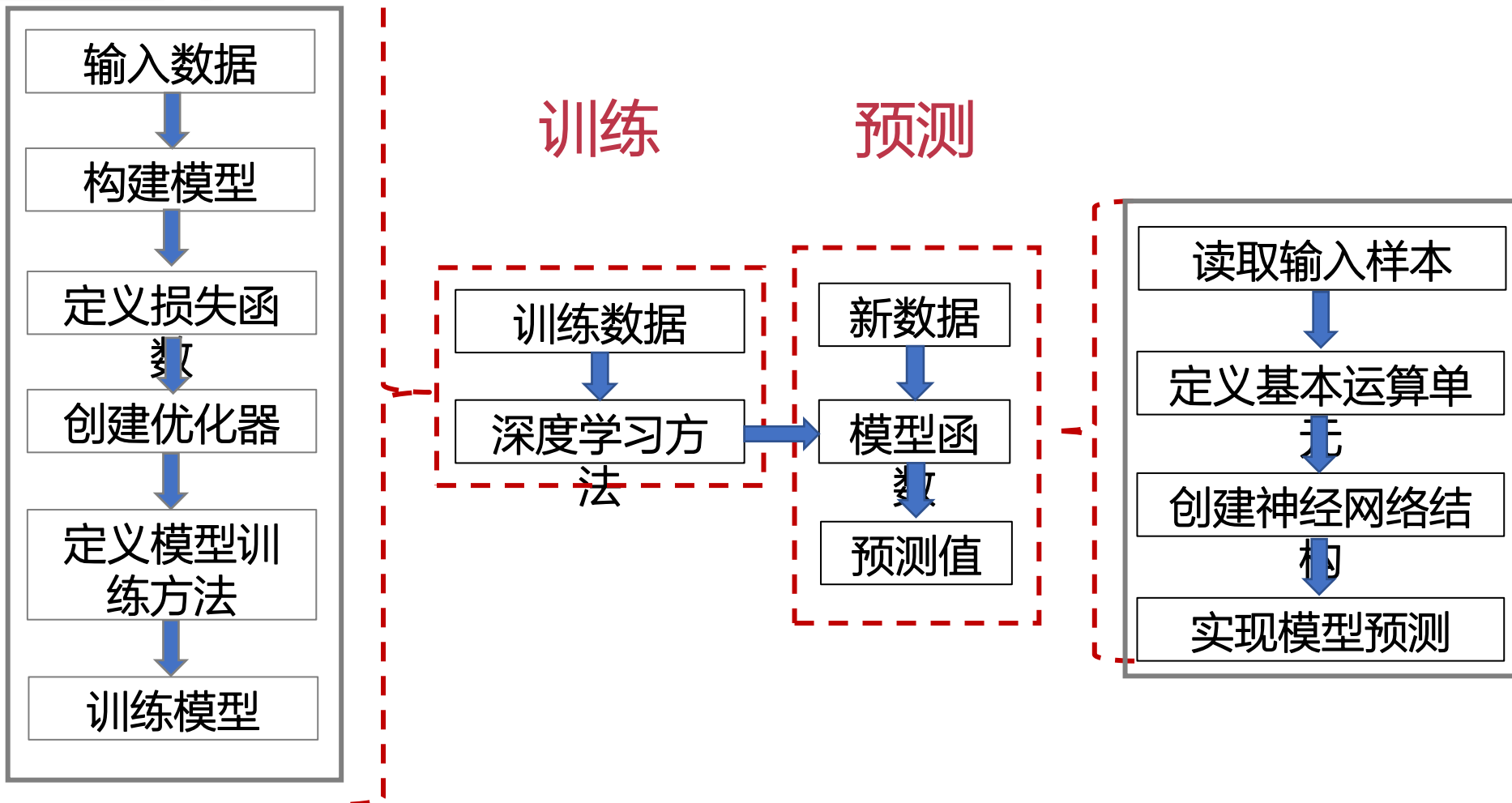


Tensorflow基础总结

- 计算图：对应神经网络结构
- 操作：对应神经网络具体计算
- 张量：对应神经网络中的数据
- 会话：执行神经网络真正的训练和预测
- 变量：对应神经网络参数
- 占位符：对应神经网络的训练或预测输入
- 队列：对应神经网络训练样本的多线程并行处理

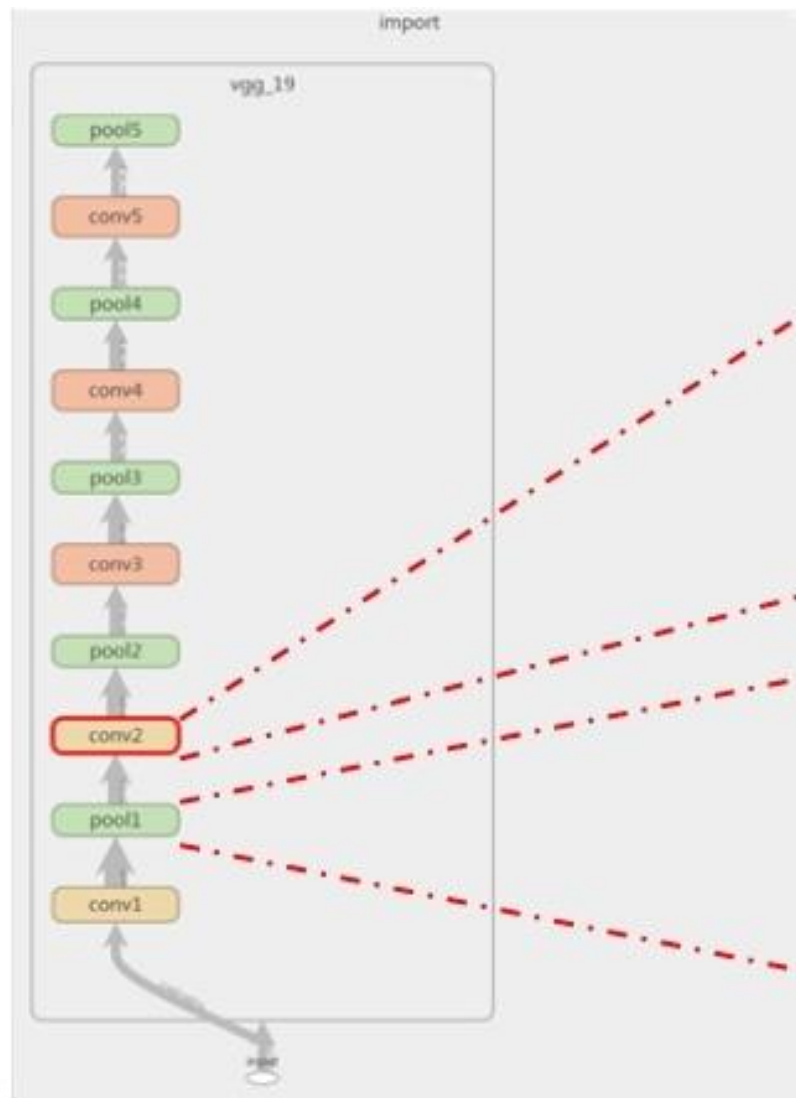


基于TensorFlow的训练





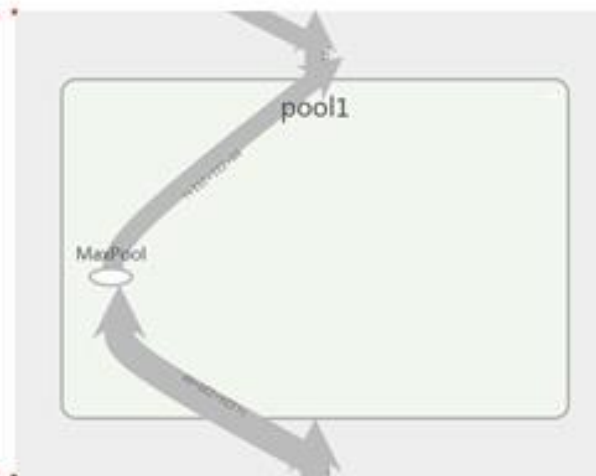
利用TensorFlow实现简单的模型预测



(a)



(c)



(b)



利用TensorFlow实现简单的模型预测

• 1、读取输入样本

```
1 import cv2
2 import numpy as np
3
4 def load_image(path):
5     img = cv2.imread(path, cv2.IMREAD_COLOR)
6     resize_img = cv2.resize(img, (224, 224))
7     norm_img = resize_img / 255.0
8     return np.reshape(norm_img, (1, 224, 224, 3))
9
```

- ▶ 支持h5、 npy, npz, ckpt和pb等格式
- ▶ 按照NHWC格式 (batch、 height、 width、 channel)



tf.nn模块定义基本运算单元

- TensorFlow中用于深度学习计算的核心模块，提供神经网络相关操作的支持，包括卷积、池化、损失、分类等
- 卷积函数

操作	说明
<code>tf.nn.conv2d(input, filter, strides, padding)</code>	在给定的 input与 filter下计算卷积
<code>tf.nn.depthwise_conv2d(input, filter, strides, padding)</code>	卷积核能相互独立的在自己的通道上面进行卷积操作。
<code>tf.nn.separable_conv2d(input, depthwise_filter, pointwise_filter, strides, padding)</code>	在纵深卷积 depthwise filter 之后进行逐点卷积 separable filter
<code>tf.nn.bias_add(value,bias)</code>	对输入加上偏置



• 激活函数

操作	说明
<code>tf.nn.relu(features)</code>	计算relu函数
<code>tf.nn.elu(features)</code>	计算elu函数
<code>tf.nn.dropout(x, keep_prob)</code>	计算dropout, keep_prob为keep概率,
<code>tf.sigmoid(x)</code>	计算sigmoid函数
<code>tf.tanh(x)</code>	计算tanh函数





• 池化函数与损失函数

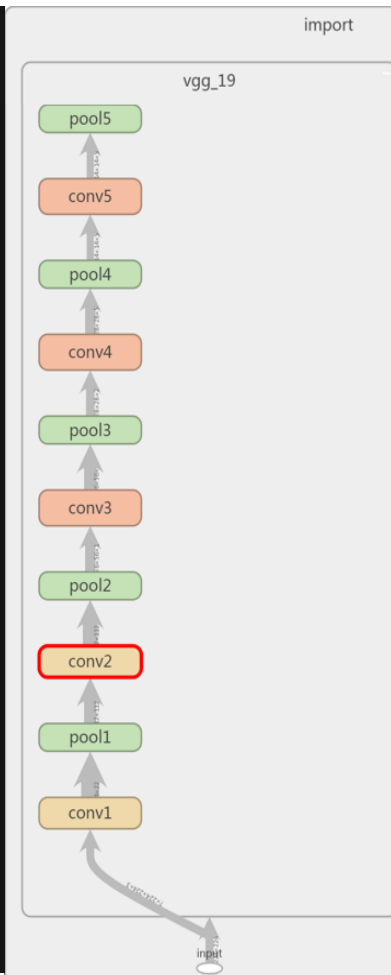
操作	说明
<code>tf.nn.avg_pool(value, ksize, strides, padding)</code>	平均方式池化
<code>tf.nn.max_pool(value, ksize, strides, padding)</code>	最大值方法池化
<code>tf.nn.max_pool_with_argmax(input, ksize, strides, padding)</code>	返回一个二维元组(output, argmax), 最大值pooling, 返回最大值及其相应的索引
<code>tf.nn.l2_loss(t)</code>	$\text{output} = \text{sum}(t ** 2) / 2$

惩罚项，正则化，防止过拟合



• 3、创建神经网络结构

```
1 def build_vggnet(vgg19_npy_path):
2     models = {}
3     models['input'] = tf.Variable(np.zeros((1, 224, 224, 3)).astype('float32'))
4     models['conv1_1'] = basic_calc('conv', models['input'], read_wb(vgg19_npy_path, 'conv1_1'))
5     models['conv1_2'] = basic_calc('conv', models['conv1_1'], read_wb(vgg19_npy_path, 'conv1_2'))
6     models['pool1'] = basic_calc('pool', models['conv1_2'])
7     models['conv2_1'] = basic_calc('conv', models['pool1'], read_wb(vgg19_npy_path, 'conv2_1'))
8     models['conv2_2'] = basic_calc('conv', models['conv2_1'], read_wb(vgg19_npy_path, 'conv2_2'))
9     models['pool2'] = basic_calc('pool', models['conv2_2'])
10    models['conv3_1'] = basic_calc('conv', models['pool2'], read_wb(vgg19_npy_path, 'conv3_1'))
11    models['conv3_2'] = basic_calc('conv', models['conv3_1'], read_wb(vgg19_npy_path, 'conv3_2'))
12    models['conv3_3'] = basic_calc('conv', models['conv3_2'], read_wb(vgg19_npy_path, 'conv3_3'))
13    models['conv3_4'] = basic_calc('conv', models['conv3_3'], read_wb(vgg19_npy_path, 'conv3_4'))
14    models['pool3'] = basic_calc('pool', models['conv3_4'])
15    models['conv4_1'] = basic_calc('conv', models['pool3'], read_wb(vgg19_npy_path, 'conv4_1'))
16    models['conv4_2'] = basic_calc('conv', models['conv4_1'], read_wb(vgg19_npy_path, 'conv4_2'))
17    models['conv4_3'] = basic_calc('conv', models['conv4_2'], read_wb(vgg19_npy_path, 'conv4_3'))
18    models['conv4_4'] = basic_calc('conv', models['conv4_3'], read_wb(vgg19_npy_path, 'conv4_4'))
19    models['pool4'] = basic_calc('pool', models['conv4_4'])
20    models['conv5_1'] = basic_calc('conv', models['pool4'], read_wb(vgg19_npy_path, 'conv5_1'))
21    models['conv5_2'] = basic_calc('conv', models['conv5_1'], read_wb(vgg19_npy_path, 'conv5_2'))
22    models['conv5_3'] = basic_calc('conv', models['conv5_2'], read_wb(vgg19_npy_path, 'conv5_3'))
23    models['conv5_4'] = basic_calc('conv', models['conv5_3'], read_wb(vgg19_npy_path, 'conv5_4'))
24    models['pool5'] = basic_calc('pool', models['conv5_4'])
25    return models
```





• 4、计算模型输出

```
1 #模型文件路径，需要加载
2 vgg19_npy_path = './vgg_models.npy'
3 #获取输入的内容图像
4 img_content = load_image('./content.jpg')
5
6 with tf.Session() as sess:
7     sess.run(tf.global_variables_initializer())
8     models = build_vggnet(vgg19_npy_path)
9
10    sess.run(models['input'].assign(img_content))
11    res = sess.run(models['pool5'])
12
13    # other process on res
14    ...
```




构建模型

输入数据

定义损失函数

创建优化器

定义模型训练方法

训练模型

训练

预测

训练数据

深度学习方法

新数据

模型函数

预测值

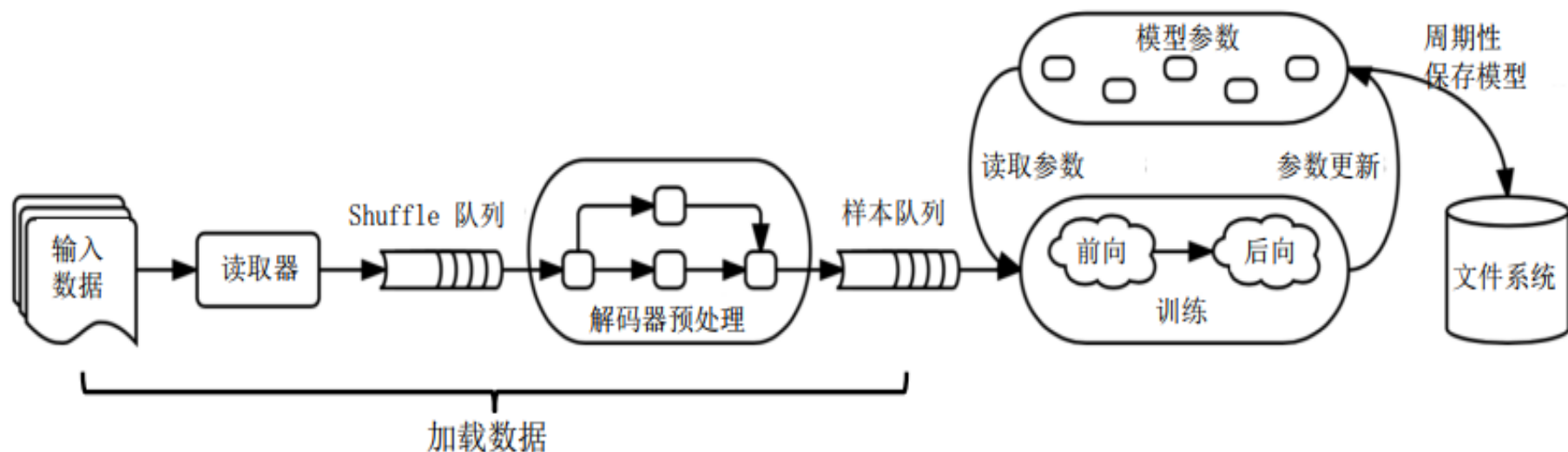
定义基本运算单元

创建神经网络结构

实现模型预测



利用TensorFlow实现简单的模型训练





```
1 def train_vgg():
2     sess = tf.Session()
3     #构建模型，使用与模型推理时相同的网络结构
4     models = build_vggnet(vgg19_npy_path)
5     #获取输入的内容图像、风格图像
6     img_content = load_image('./content.jpg')
7     img_style = load_image('./style.jpg')
8     #生成噪声图像img_random
9     img_random = get_random_img(img_content)
10    sess.run(tf.global_variables_initializer())
11    #定义损失函数
12    total_loss = loss(sess, models, img_content, img_style)
13    #创建优化器
14    optimizer = tf.train.AdamOptimizer(2.0)
15    #定义模型训练方法
16    train_op = optimizer.minimize(total_loss)
17    sess.run(tf.global_variables_initializer())
18    #使用噪声图像img_random进行训练
19    sess.run(models['input'].assign(img_random))
20
21    for i in range(3000):
22        #完成一次反向传播
23        sess.run(train_op)
24        if i % 100 == 0:
25            #每完成100次训练即打印中间结果，从而监测训练效果
26            img_transfer = sess.run(models['input'])
27            print('Iteration %d' % (i))
28            print('cost: ', sess.run(total_loss))
29
30    #训练结束，保存训练结果，显示图像
31    ...
32
33 if __name__ == '__main__':
34     train_vgg()
```

构建模型

1、加载数据

2、定义损失函数

3、创建优化器

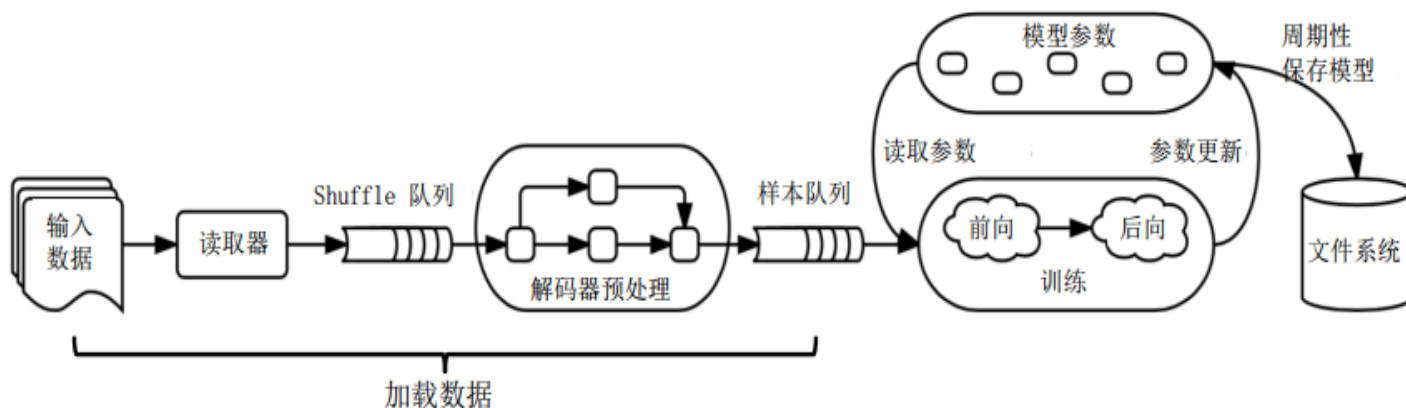
4、定义模型训练方法

训练模型



1、加载数据

- 注入（feeding）：利用 `feed_dict` 直接传递输入数据），配合 `placeholder`。
- 预取（`pre_load`）：利用 `Const`和 `Variable`直接读取输入数据
 - 利用常量进行数据预取
 - 数据量小的时候适用
 - 内联，多次使用重复复制，浪费内存空间
- 基于队列API：基于队列相关的API来构建输入流水线（`Pipeline`），配合 `placeholder`。
- `tf.data` API：利用 `tf.data` API来构建输入流水线





2、定义损失函数

$$L_{total}(p, a, x) = \alpha L_{content}(p, x) + \beta L_{style}(a, x)$$

10^{-4}



10^{-3}



10^{-2}



10^{-1}



α/β 越大

内容越具象



2、定义损失函数

- 基本函数

类别	基本函数及功能说明
四则运算	tf.add(), tf.sub(), tf.mul()
科学计算	tf.abs(), tf.square(), tf.sin()
比较操作	tf.greater() (返回True或False)
条件判断	tf.where(condition, tensor_x, tensor_y) (condition为True时返回tensor_x, 否则返回tensor_y)
降维操作	tf.reduce_sum(), tf.reduce_mean() (将高维矩阵元素以求和或均值的方式变为一维)

```
1 loss = tf.reduce_mean(tf.square(y-y_data))
2 #实际值和预测值的差值平方再求平均值, 训练的目的就是要让这个loss越来越小
3
```



TensorFlow内置的4个损失函数

- 1) softmax交叉熵:

`tf.nn.softmax_cross_entropy_with_logits(labels, logits)`

- 2) 加了稀疏的softmax交叉熵

`tf.nn.sparse_softmax_cross_entropy_with_logits(labels, logits)`

- 3) sigmoid交叉熵

`tf.nn.sigmoid_cross_entropy_with_logits(labels, logits)`

- 4) 带权重的sigmoid交叉熵:

`tf.nn.weighted_cross_entropy_with_logits(targets, logits, pos_weight)`



2、创建优化器

- 优化器的功能是实现优化算法，可以自动为用户计算模型参数的梯度值
- TensorFlow中支持的优化器函数
 - `tf.train.Optimizer` 基类
 - `tf.train.GradientDescentOptimizer`: 梯度下降优化器
 - `tf.train.AdadeltaOptimizer`
 - `tf.train.AdagradOptimizer`
 - `tf.train.AdagradDAOptimizer`
 - `tf.train.MomentumOptimizer`: 动量梯度下降优化器
 - `tf.train.AdamOptimizer`: Adam算法优化器
 - `tf.train.FtrlOptimizer`
 - `tf.train.ProximalGradientDescentOptimizer`
 - `tf.train.ProximalAdagradOptimizer`
 - `tf.train.RMSPropOptimizer`



- `tf.train.AdamOptimizer`: Adam算法优化器
 - 返回一个使用Adam算法的优化器，参数为`learningRate`
 - **Adam算法**: 综合了 Momentum 和 RMSProp 方法，根据损失函数对每个参数的梯度的一阶矩估计和二阶矩估计动态调整针对于每个参数的学习率
 - 用法:

```
train=tf.train.AdamOptimizer(learningRate)
```



3、定义模型训练方法

- 一般采用最小化损失函数（minimize）的方法

```
1 train_op = tf.train.Optimizer.minimize(loss,global_step=None,var_list=None)
```

操作	功能
tf.train.Optimizer.minimize(loss,global_step=None,var_list=None)	使用最小化损失函数的方法来训练模型。执行该操作时会内部依次调用compute_gradients和apply_gradients操作
tf.train.Optimizer.compute_gradients(loss,var_list=None)	对var_list中列出的模型参数计算梯度，返回（梯度，模型参数）组成的列表
tf.train.Optimizer.apply_gradients(grads_and_vars)	将计算出的梯度更新到模型参数上，返回更新参数的操作



4、模型保存

- 在模型训练过程中，使用`tf.train.Saver()`来保存模型中的所有变量

```
1 import tensorflow as tf
2 weights = tf.Variable(tf.random_normal([30,60],stddev=0.35),name="weights")
3 w2 = tf.Variable(weights.initialized_value(),name="w2")
4
5 #实例化saver对象
6 saver = tf.train.Saver()
7
8 with tf.Session() as sess:
9     sess.run(tf.global_variables_initializer())
10    for step in xrange(1000000):
11        #执行模型训练
12        sess.run(training_op)
13        if step % 1000 == 0:
14            # 将训练得到的变量值保存到检查点文件中
15            saver.save(sess, './ckpt/my-model')
```



恢复模型

- 当需要基于某个checkpoint继续训练模型参数时，需要从.ckpt文件中恢复出已保存的变量
- 同样使用tf.train.Saver()来恢复变量，恢复变量时不需要先初始化变量

```
1 import tensorflow as tf
2
3 weights = tf.Variable(tf.random_normal([30,60],stddev=0.35),name="weights")
4 w2 = tf.Variable(weights.initialized_value(),name="w2")
5
6 #模型路径只需要给出文件夹名称
7 model_path = "./ckpt"
8
9 #实例化saver对象
10 saver = tf.train.Saver()
11
12 with tf.Session() as sess:
13     #找到存储变量值的位置
14     ckpt = tf.train.latest_checkpoint(model_path)
15     #恢复变量
16     saver.restore(sess,ckpt)
17     print(sess.run(weights))
18     print(sess.run(w2))
```



图像风格迁移训练的实现

```
1 def train_vgg():
2     sess = tf.Session()
3     #构建模型，使用与模型推理时相同的网络结构
4     models = build_vggnet(vgg19_npy_path)
5     #获取输入的内容图像、风格图像
6     img_content = load_image('./content.jpg')
7     img_style = load_image('./style.jpg')
8     #生成噪声图像img_random
9     img_random = get_random_img(img_content)
10    sess.run(tf.global_variables_initializer())
11    #定义损失函数
12    total_loss = loss(sess, models, img_content, img_style)
13    #创建优化器
14    optimizer = tf.train.AdamOptimizer(2.0)
15    #定义模型训练方法
16    train_op = optimizer.minimize(total_loss)
17    sess.run(tf.global_variables_initializer())
18    #使用噪声图像img_random进行训练
19    sess.run(models['input'].assign(img_random))
20
21    for i in range(3000):
22        #完成一次反向传播
23        sess.run(train_op)
24        if i % 100 == 0:
25            #每完成100次训练即打印中间结果，从而监测训练效果
26            img_transfer = sess.run(models['input'])
27            print('Iteration %d' % (i))
28            print('cost: ', sess.run(total_loss))
29
30    #训练结束，保存训练结果，显示图像
31    ...
32
33 if __name__ == '__main__':
34     train_vgg()
```



小结

- 深度学习编程框架的概念

深度学习编程框架的概念及分类

- TensorFlow概述

TensorFlow的历史、发展历程

- TensorFlow编程模型及基本用法

TensorFlow中常用的计算图、张量、操作、会话、变量、占位符、队列等概念

- 基于TensorFlow的训练及预测实现

TensorFlow模型训练、预测等

谢谢大家！