深度学习与自然语言处理(6)_斯坦福cs224d 一起来学Tensorflow part1

标签： 自然语言处理 深度学习 Tensorflow 张量 神经网络 2016-07-10 13:12 19564人阅读 评论(9) 收藏 举报

分类： 深度学习与自然语言处理（7）

目录(?) [+]

内容翻译：@穆文(**微信**公众号 数据挖掘机养成记) && 寒小阳
校正调整：寒小阳 && 龙心尘
时间：2016年7月
出处：http://blog.csdn.net/han_xiaoyang/article/details/51871068

说明：本文为斯坦福大学CS224d课程的中文版内容笔记整理，已得到斯坦福大学课程@Richard Socher教授的授权翻译

0.前言

之前的课程里介绍了**自然语言**处理当中的一些问题，以及设计出来的一些相应的**算法**。research的东西还是落地到工程应用上比较有价值，之前也手撸过一些toy project，不过这些实现要用在工程中，总是有那么些虚的，毕竟稳定性和效率未必能够保证。所幸的是，**深度学习**热度持续升温的大环境下，各种大神和各家大厂也陆续造福民众，开源了一些深度学习框架，在这些开源框架的基础上去搭建和实现自己想要的深度学习网络结构就简单和稳定得多了。

有时候选择多了也是麻烦，对框架感兴趣的同学可以查看深度学习框架对比维基百科中对12个开源的package比对。这里简单提几个最常见和可能会用到的深度学习开源框架的特点。

caffe提供了很便捷的神经网络搭建和命令行工具，加之model zoo里面大量预训练好的模型(主要是图像相关的)可以做fine-tuning，因此使用在图像相关的研究和应用上非常方便。
Theano以及搭建于其之上的**Keras**和**Lasagne**似乎颇受research派系同学的偏爱，自动求导是它的优势之一。
MXnet对显存的利用率高，并且支持C++, **Python**, Julia, Matlab, **JavaScript**, **Go**, R, **Scala**这么多种语言，编写起来也比较简易。
Torch是facebook用的深度学习package，定义新网络层比较简单，不过Lua倒不算大家熟知的编程语言。
Tensorflow是Google提供资金研发的，比较全，支持分布式，同时有Google这样的亲爹在，我猜资源倾斜也是迟早的事情。

今天的重点自然是Tensorflow，其他的框架也都很好，大家可以自行尝试。

1.Tensorflow

首先提提Tensorflow和theano，它俩都是**python**封装的深度学习库，非常容易上手，说起来Tensorflow还是受Theano启发，借鉴了一部分它的思想。不同之处在于，Tensorflow 对分布式系统支持更好，同时还是Google提供资金研发的，而Theano 是一个学术性质的项目。

Tensorflow 可以对定义在张量(tensors,你可以先简单理解成标量、向量或者矩阵，一会儿会提到)上的函数自动求导，因此神经网络中BP算法可以很轻松地实现。

在开始Tensorflow之前，需要先让大家对Tensorflow的过程有个直观的理解。

在Tensorflow里：

- 使用 `张量(tensor)` 表示数据.
- 使用 `图(graph)` 来表示计算任务.
- 在被称之为 `会话(Session)` 的 `上下文 (context)` 中执行图.
- 通过 `变量 (Variable)` 维护状态.
- 使用 `feed` 和 `fetch` 可以为任意的操作(arbitrary operation)赋值或者从其中获取数据.

严格意义上来说TensorFlow算是一个编程系统，它使用 `图` 来表示计算任务，图中的 `节点` 被称之为operation(可以缩写成 `op`)，一个节点获得0个或者多个 `张量` (tensor，下文会介绍到)，执行计算，产生0个或多个张量。TensorFlow的一个 `图` 描述了一个计算过程，为了进行计算， `图` 必须在 `会话(Session)` 里被启动， `会话(Session)` 将 `图` 的 `op` 分发到CPU或GPU之类的设备上，同时提供执行 `op` 的方法，这些方法执行后，将产生的 `张量(tensor)` 返回。返回的张量因语言不同而有不同，在python里是numpy ndarry对象；在C/C++语言中，是tensorflow::Tensor实例。

下面咱们来详细说说上面提到的概念。

1.1 什么是张量

既然Tensorflow里面的定义和运算都是基于张量这个概念，我们就先来看看，什么是张量。

- 张量的正式定义：从向量空间到实数域的多重现性映射(multilinear maps)（V是向量空间，V^*是对偶空间）
  - $$f : \underbrace{V^* \times \cdots V^*}_{\text{p copies}} \times \underbrace{V \times \cdots V}_{\text{q copies}} \rightarrow R$$
  - 标量是张量($f : R \rightarrow R, f(e_1) = c$)（*译者注: 标量是用实数表示零维空间的点*）
  - 向量是张量($f : R^n \rightarrow R, f(e_i) = v_i$)（*译者注: 向量是用实数表示一维空间的点，也即向量中的某个元素*）
  - 矩阵是张量($f : R^n \times R^m \rightarrow R, f(e_i, e_j) = A_{ij}$)（*译者注: 矩阵是用实数表示二维空间的点，也即矩阵的某个元素*）
  - 通常来说，**张量可以用多维数组来表示**

1.2 Tensorflow 与 Numpy

- 看似差别甚远的2个package，说起来可能也很少有人把这两者作对比，但他们“长得”确实很相似（都是提供N维数组的库）
- Numpy 有 Ndarray(N维数组) 支持，但不提供创建张量函数和自动求导的方法，也不提供GPU支持

1.3 Numpy 与 Tensorflow 定义与操作对比

```
# numpy定义与操作
In [23]: import numpy as np
In [24]: a = np.zeros((2,2)); b = np.ones((2,2))
In [25]: np.sum(b, axis=1)
Out[25]: array([ 2.,  2.])
In [26]: a.shape
Out[26]: (2, 2)
In [27]: np.reshape(a, (1,4))
Out[27]: array([[ 0.,  0.,  0.,  0.]])
```

```
# 对应的Tensorflow定义与操作
In [31]: import tensorflow as tf
In [32]: tf.InteractiveSession()
In [33]: a = tf.zeros((2,2)); b = tf.ones((2,2))
In [34]: tf.reduce_sum(b, reduction_indices=1).eval()
Out[34]: array([ 2.,  2.], dtype=float32)
In [35]: a.get_shape()
Out[35]: TensorShape([Dimension(2), Dimension(2)])
In [36]: tf.reshape(a, (1, 4)).eval()
Out[36]: array([[ 0.,  0.,  0.,  0.]], dtype=float32)
```

以上代码中提到的`session`和`.eval()`将在下文细述，而关于`TensorShape`，大家可以简单理解成类似Python中tuple的类型。

为了方便记忆，我们把numpy和Tensorflow中的部分定义和操作做成了一张一一对应的表格，方便大家查看。

Numpy	**Tensorflow**
a = np.zeros((2,2)); b = np.ones((2,2))	a = tf.zeros((2,2)), b = tf.ones((2,2))
np.sum(b, axis=1)	tf.reduce_sum(a,reduction_indices=[1])
a.shape	a.get_shape()
np.reshape(a, (1,4))	tf.reshape(a, (1,4))
b*5+1	b*5+1
np.dot(a,b)	tf.matmul(a, b)
a[0,0], a[:,0], a[0,:]	a[0,0], a[:,0], a[0,:]

Tensorflow的输出要稍微注意一下，我们需要显式地输出(evaluation，也就是说借助eval()函数)！

```
In [37]: a = np.zeros((2,2))
In [38]: ta = tf.zeros((2,2))
In [39]: print(a)
[[ 0.  0.]
 [ 0.  0.]]
In [40]: print(ta)
Tensor("zeros_1:0", shape=(2, 2), dtype=float32)
In [41]: print(ta.eval())
[[ 0.  0.]
[ 0. 0.]]
```

上面是一个示例的代码，大家可以理解Tensorflow是通过计算图（computation graph）定义一个计算过程的，这个过程不产生数值结果，那想看到具体内容怎么办呢？我们要借助`.eval()`函数输出。

1.4 Tensorflow 的计算图

用Tensorflow编写的程序一般由两部分构成，一是构造部分，包含了计算流图，二是执行部分，通过session 来执行图中的计算，具体可以参考Tensorflow文档

我们先来看看怎么构建图。构件图的第一步是创建`源节点`(source op)。`源节点`不需要任何输入，它的输出传递给其它节点(op)做运算。python库中，节点构造器的返回值即当前节点的输出，这些返回值可以传递给其它节点(op)作为输入。

TensorFlow Python库中有一个`默认图`(default graph)，在默认图的基础上，`节点构造器(op 构造器)`可以为其增加节点。这个默认图对许多程序来说已经足够用了，更多管理视图的细节可以阅读官方Graph类文档。

我们来看一个简单的构建图例子：

```
import tensorflow as tf
# 创建一个常量节点， 产生一个1x2矩阵，这个op被作为一个节点
# 加到默认视图中
# 构造器的返回值代表该常量节点的返回值
matrix1 = tr.constant([[3., 3.]])

# 创建另一个常量节点，产生一个2x1的矩阵
matrix2 = tr.constant([[2.], [2.]])

# 创建一个矩阵乘法matmul节点，把matrix1和matrix2作为输入：
product = tf.matmul(matrix1, matrix2)
```

上面代码里的默认图现在有三个节点，两个constant()节点和matmul() 节点。不过这仅仅是构建图，为了真正进行矩阵的乘法，你必须在`会话(Session，马上提到)`里启动这个图。

1.5 Tensorflow与Session对象

上面我们知道了Tensorflow需要先构造一个图用于计算，但是图怎么启动呢？启动图的第一步需要创建一个Session对象。比如：

```
# 创建session，启动默认图
sess = tf.Session()

# 调用sess的'run()' 方法来执行矩阵乘法节点操作，传入'product'作为该方法的参数。'product'代表了矩阵乘法节点的输出，传入它是告诉方法我们希望取

#整个执行过程是自动化的，会话负责传递节点所需的全部输入。节点通常是并发执行的。

# 函数调用'run(product)'会触发图中三个节点（上面例子里提到的两个常量节点和一个矩阵乘法节点）的执行。

# 返回值'result'是一个numpy 'ndarray'对象。

result = sess.run(product)
print result
# 结果为[[12.]]

# 完成任务，记得关闭会话
sess.close()
```

Session对象在使用完成后，记得关闭以释放资源，当然，除了显式调用close关闭外，也可以使用with代码来自动完成关闭动作：

```
# 用with代码来自动完成session里的图运算并关闭
with tf.Session() as sess:
  result = sess.run([product])
  print result
```

为了便于使用像IPython这样的python交互环境，可以使用InteractiveSession代替Session类，使用Tensor.eval()和Operation.run()方法代替Session.run()。这样做的好处是可以在ipython中保持默认session处于打开状态：

```
# 进入一个交互式Tensorflow会话
import tensorflow as tf
sess = tf.InteractiveSession()

x = tf.Variable([1.0, 2.0])
a = tf.constant([3.0, 3.0]);
```

```
7
8  # 使用初始化器的run()方法初始化x
9  x.initializer.run()
10
11 # 增加一个减法节点，从x减去a。运行减法op，输出结果
12 sud = tf.sub(x, a)
13 print sub.eval()
14 # 结果为[-2. -1.]
```

1.6 关于session和多GPU运算

我们一直在说，Tensorflow是支持分布式的深度学习框架/包，这是因为它能将图定义转换成分布式执行的操作，以充分利用可以利用的计算资源（如CPU或GPU）。不过一般情况下，你不需要显式指定使用CPU还是GPU，Tensorflow能自动检测。如果检测到GPU，Tensorflow会优先使用找到的第一个GPU来执行操作。

如果机器上有超过一个可用的GPU，默认状况下除了第一个外的其他GPU是不参与计算的。为了让Tensorflow使用这些GPU，你必须将节点运算明确地指派给它们执行。其中with...Device语句用来指派特定的CPU或GPU操作：

```
1  # 手动指定给某个gpu执行
2  with tf.Session() as sess:
3    with tf.device("/gpu:1"):
4      matrix1 = tf.constant([[3., 3.]])
5      matrix2 = tf.constant([[2.], [2.]])
6      product = tf.matmul(matrix1, matrix2)
```

指定设备的书写格式如下：

- `/cpu:0`:机器的CPU
- `/gpu:0`:机器的第一个GPU，如果有的话
- `/gpu:1`:机器的的第二个GPU，其他GPU以此类推

1.7 Tensorflow的变量(Variables)

我们训练一个模型的时候，会用到Tensorflow中的`变量(Variables)`，我们需要它来保持和更新参数值，和`张量`一样，`变量`也保存在内存缓冲区当中。

有很多同学会问，前面不是提到了一个概念叫做张量，为什么还需要这个新的变量呢？需要说明一下的是，如果大家仔细看之前的代码，会发现我们所用到的张量都是常值张量(constant tensors)，而非变量，而参数值是需要动态调整的内容。

比如下面的代码里我们设定了一组权重为变量：

```
1  In [32]: W1 = tf.ones((2,2))
2  In [33]: W2 = tf.Variable(tf.zeros((2,2)), name="weights")
3  In [34]: with tf.Session() as sess:
4             print(sess.run(W1))
5             sess.run(tf.initialize_all_variables())
6             print(sess.run(W2))
7     ....:
8  [[ 1.  1.]
9   [ 1.  1.]]
10 [[ 0.  0.]
11 [ 0.  0.]]
```

说一个小细节，注意到上面第34步`tf.initialize_all_variables`，我们要预先对变量初始化(initialization)

Tensorflow 的变量必须先初始化然后才有值！而常值张量是不需要的

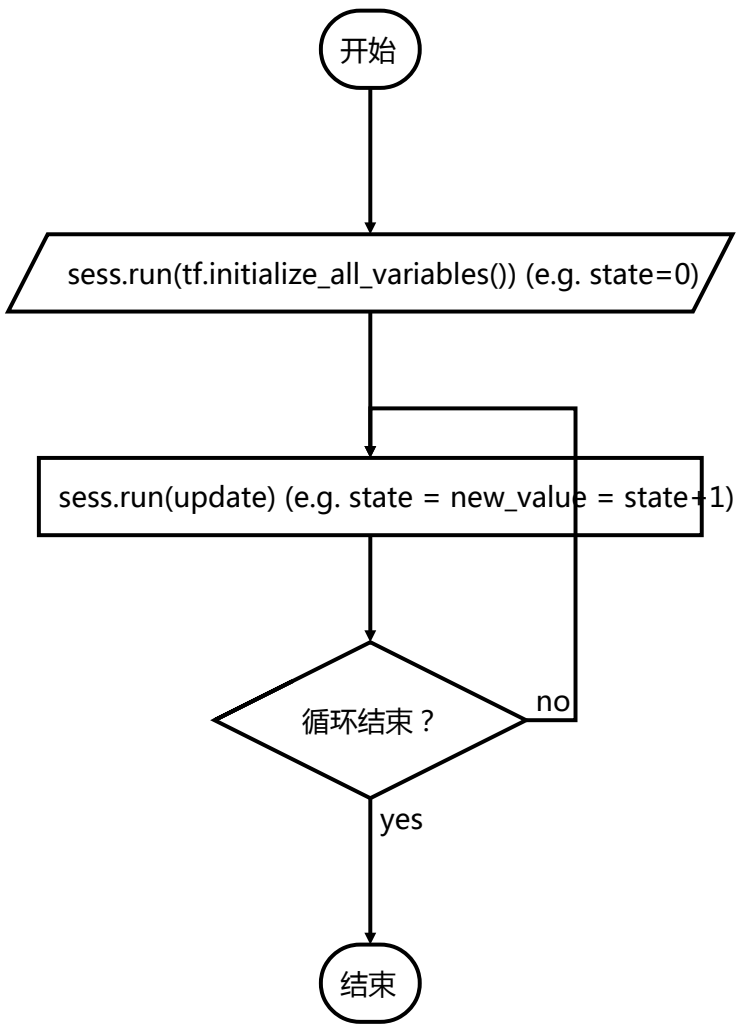
再具体一点，比如下面的代码，其实38和39步，我们初始化定义初值是可以通过常数或者随机数等任何一种方式初始化的，但是直到第40步才真正通过Tensorflow的initialize_all_variables对这些变量赋初值。

```
1  In [38]: W = tf.Variable(tf.zeros((2,2)), name="weights")
2  In [39]: R = tf.Variable(tf.random_normal((2,2)), name="random_weights")
3  In [40]: with tf.Session() as sess:
4     ....:     sess.run(tf.initialize_all_variables())
5     ....:     print(sess.run(W))
6     ....:     print(sess.run(R))
7     ....:
```

比如我们来看一个`计算图`中变量的状态更新过程，代码如下：

```
In [63]: state = tf.Variable(0, name="counter")
In [64]: new_value = tf.add(state, tf.constant(1))
In [65]: update = tf.assign(state, new_value)
In [66]: with tf.Session() as sess:
    sess.run(tf.initialize_all_variables())
    print(sess.run(state))
    for _ in range(3):
        sess.run(update)
        print(sess.run(state))
0
1
2
3
```

上面的代码定义了一个如下的计算图，同时其中变量的状态是循环变化的：*



1.8 Tensorflow的Fetch(获取)操作

如果想取回定义的计算图中的节点运算输出结果，可以在使用Session对象的run()调用执行图时，传入一些张量，这些张量可以帮助你取回结果。而且不仅仅是单个节点的状态或者结果，可以输出多个节点的结果，比如下面这个简单例子：

```
input1 = tf.constant(3.0)
input2 = tf.constant(4.0)
input3 = tf.constant(5.0)
intermed = tf.add(input2, input3)
mul = tf.mul(input1, intermed)

with tf.Session() as sess:
  result = sess.run([mul, intermed])
  print result

# print
# 输出最后的乘法结果，和之前的加法结果[27.0, 9.0]
```

1.9 Tensorflow与Feed(传入)操作

1.8里我们提到了在计算图中引入张量，以获取节点状态或者输出结果。Tensorflow还提供了feed机制，该机制可以临时替代图中的任意操作中的张量，也就是说，可以对图中任何操作提交补丁，直接插入一个新的张量。

feed可以使用一个张量值临时替换某个操作的输出结果，你只需要提供feed数据作为run()调用的参数。需要说明的是，feed只在调用它的方法内有效，方法结束则feed就会消失。最常见的用例是将某些特殊的操作指定为feed操作，标记的方法是使用tf.placeholder()为这些操作创建占位符(可以先想成一个容器，这个在之后的内容里会提到，不要着急)。

```
input1 = tf.placeholder(tf.types.float32)
input2 = tf.placeholder(tf.types.float32)
output = tf.mul(input1, input2)

# 手动提供feed数据作为run的参数
with tf.Session() as see:
  print sess.run([output], feed_dict={input:[7.],  input2:[2.]})

# print
# 结果是[array([ 14.], dtype=float32)]
```

2.结语

这个部分呢，就先简单给大家介绍Tensorflow的一些常用对象，基本操作和设计思想。之后会针对常见的问题(回归，图像分类/CNN， 自然语言处理/RNN)逐个进行讲解。欢迎大家继续关注。