https://www.wandouip.com/t5i183316/

引言

原来引用过一个段子，这里还要再引用一次。是关于苹果的。大意是，苹果发布了新的开发语言Swift，有非常多优秀的特征，于是很多时髦的程序员入坑学习。不料，经过一段头脑体操一般的勤学苦练，发现使用Swift做开发，不仅要学习Swift，还要学习Swift2、Swift3、Swift4...
后来我发现，这个段子很有普遍性，并非仅仅苹果如此，今天的TensorFlow 2.0也有点这样的趋势。以至于我不得不专门写一个课程的续集，来面对使用新版本软件开始机器学习的读者。
事实上大多具有革命性的公司都是这样，一方面带来令人兴奋的新特征，另一方面则是高企不落的学习成本。

《从锅炉工到AI专家》一文中，已经对机器学习的基本概念做了很详细的介绍。所以在这里我们就省掉闲言絮语，直接从TensorFlow2.0讲起。
当然即便没有看过这个系列，假设你对TensorFlow 1.x很熟悉，也可以直接通过阅读本文，了解从TensorFlow 1.x迁移至2.x的知识。
如果你不了解机器学习的概念，试图通过直接学习TensorFlow 2.0开始AI开发，那可能比较困难。TensorFlow只是工具。没有技能，只凭工具，你恐怕无法踏上旅程。

安装

截至本文写作的时候，TensorFlow 2.0尚未正式的发布。pip仓库中仍然是1.13稳定版。所以如果想开始TensorFlow 2.0的学习，需要指定版本号来安装。此外由于Python2系列将于2020年元月停止官方维护，本文的示例使用Python3的代码来演示：

```
$ pip3 install tensorflow==2.0.0-alpha0
```

(注: 上面$是Mac/Linux的提示符，假如用Windows,你看到的提示符应当类似C:\Users\Administrator>这样子。)

如果希望使用GPU计算，安装的预先准备会麻烦一些，请参考这篇文档：
https://www.tensorflow.org/install/gpu。主要是安装CUDA/cuDNN等计算平台的工具包。其中CUDA可以使用安装程序直接安装。cuDNN是压缩包，如果不打算自己编译TensorFlow的话，放置到CUDA相同目录会比较省事。
这里提醒一下，除非自己编译TensorFlow，否则一定使用CUDA 10.0的版本，低了、高了都不成，因为官方的2.0.0-alpha0使用了CUDA 10.0的版本编译。
此外TensorFlow的安装请使用如下命令：

```
$ pip3 install tensorflow-gpu==2.0.0-alpha0
```

安装完成后，可以在Python的交互模式，来确认TensorFlow正常工作：

```
$ python3
Python 3.7.2 (default, Feb 13 2019, 13:59:29)
[Clang 10.0.0 (clang-1000.11.45.5)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow as tf
>>> tf.__version__
'2.0.0-alpha0'
>>>
```

本文中还会用到几个第三方的python扩展库，也是在机器学习中非常常用的，建议一起安装：

```
$ pip3 install numpy matplotlib pillow pandas seaborn sklearn
```

第一个例子：房价预测

本示例中的源码来自于《从锅炉工到AI专家》系列2，使用了最简单的线性函数来做房价预测。原始TensorFlow 1.x/ Python 2.x代码如下：

```
#!/usr/bin/env python
# -*- coding=UTF-8 -*-

#本代码在mac电脑，python2.7环境测试通过
#第一行是mac/Linux系统脚本程序的标志，表示从环境参量中寻找python程序解释器来执行本脚本
#省去了每次在命令行使用 python <脚本名> 这样的执行方式
#第二行表示本脚本文本文件存盘使用的代码是utf-8,并且字符串使用的编码也是utf-8,
#在本源码中，这一点其实没有什么区别，但如果需要中文输出的时候，这一行就必须要加了。

#引入TensorFlow库
import tensorflow as tf
#引入数值计算库
import numpy as np

#使用 NumPy 生成假数据集x,代表房间的平米数，这里的取值范围是0-1的浮点数，
#原因请看正文中的说明，属于是"规范化"之后的数据
# 生成的数据共100个，式样是100行，每行1个数据
x = np.float32(np.random.rand(100,1))
#我们假设每平米0.5万元，基础费用0.7万，这个数值也是规范化之后的，仅供示例
#最终运行的结果，应当求出来0.5/0.7这两个值代表计算成功
#计算最终房价y，x和y一同当做我们的样本数据
# np.dot的意思就是向量x * 0.5
y = np.dot(x,0.5) + 0.7
#---------------------------------数据集准备完成
#以下使用TensorFlow构建数学模型，在这个过程中，
#直到调用.run之前，实际上都是构造模型，而没有真正的运行。
#这跟上面的numpy库每一次都是真正执行是截然不同的区别
# 请参考正文，我们假定房价的公式为：y=a*x+b

#tf.Variable是在TensorFlow中定义一个变量的意思
#我们这里简单起见，人为给a/b两个初始值，都是0.3，注意这也是相当于规范化之后的数值
b = tf.Variable(np.float32(0.3))
a = tf.Variable(np.float32(0.3))

#这是定义主要的数学模型，模型来自于上面的公式
#注意这里必须使用tf的公式，这样的公式才是模型
#上面使用np的是直接计算，而不是定义模型
# TensorFlow的函数名基本就是完整英文，你应当能读懂
y_value = tf.multiply(x,a) + b

# 这里是代价函数，同我们文中所讲的唯一区别是用平方来取代求绝对值，
#目标都是为了得到一个正数值，功能完全相同，
#平方计算起来会更快更容易,这种方式也称为"方差"
loss = tf.reduce_mean(tf.square(y_value - y))
# TensorFlow内置的梯度下降算法，每步长0.5
optimizer = tf.train.GradientDescentOptimizer(0.5)
# 代价函数值最小化的时候，代表求得解
train = optimizer.minimize(loss)

# 初始化所有变量，也就是上面定义的a/b两个变量
init = tf.global_variables_initializer()

#启动图
sess = tf.Session()
#真正的执行初始化变量，还是老话，上面只是定义模型，并没有真正开始执行
sess.run(init)
```

```
#重复梯度下降200次，每隔5次打印一次结果
for step in xrange(0, 200):
    sess.run(train)
    if step % 5 == 0:
        print step, sess.run(loss),sess.run(a), sess.run(b)
```

代码保留了原始的注释，希望如果概念已经没有问题的话，可以让你不用跑回原文去看详细讲解。
程序使用numpy生成了一组样本集，样本集是使用线性函数生成的。随后使用TensorFlow学习这些样本，从而得到线性函数中未知的权重(Weight)和偏移(Bias)值。
原文中已经说了，这个例子并没有什么实用价值，只是为了从基础开始讲解"机器学习"的基本原理。

使用2.0中的v1兼容包来沿用1.x代码

TensorFlow 2.0中提供了tensorflow.compat.v1代码包来兼容原有1.x的代码，可以做到几乎不加修改的运行。社区的contrib库因为涉及大量直接的TensorFlow引用代码或者自己写的Python扩展包，所以无法使用这种模式。TensorFlow 2.0中也已经移除了contrib库，这让人很有点小遗憾的。
使用这种方式升级原有代码，只需要把原有程序开始的TensorFlow引用:

```
import tensorflow as tf
```

替换为以下两行就可以正常的继续使用：

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
```

其它代码无需修改。个人觉得，如果是稳定使用中、并且没有重构意愿的代码，这种方式算的上首选。

使用迁移工具来自动迁移1.x代码到2.0

TensorFlow 2.0中提供了命令行迁移工具，来自动的把1.x的代码转换为2.0的代码。工具使用方法如下(假设我们的程序文件名称为first-tf.py)：

```
tf_upgrade_v2 --infile first-tf.py --outfile first-tf-v2.py
```

迁移工具还可以对整个文件夹的程序做升级，请参考工具自身的帮助文档。
使用迁移工具升级的代码，实质上也是使用了tensorflow.compat.v1兼容包来提供在TensorFlow 2.0环境中执行1.x的代码。这里贴出自动转换后的新代码供你对比参考：

```
#引入TensorFlow库
import tensorflow as tf
#import tensorflow.compat.v1 as tf
tf.compat.v1.disable_v2_behavior()

#引入数值计算库
import numpy as np

#使用 NumPy 生成假数据集x,代表房间的平米数，这里的取值范围是0-1的浮点数，
#原因请看正文中的说明，属于是"规范化"之后的数据
# 生成的数据共100个，式样是100行，每行1个数据
x = np.float32(np.random.rand(100,1))
#我们假设每平米0.5万元，基础费用0.7万，这个数值也是规范化之后的，仅供示例
#最终运行的结果，应当求出来0.5/0.7这两个值代表计算成功
#计算最终房价y，x和y一同当做我们的样本数据
# np.dot的意思就是向量x * 0.5
y = np.dot(x,0.5) + 0.7
#---------------------------------数据集准备完成
#以下使用TensorFlow构建数学模型，在这个过程中，
#直到调用.run之前，实际上都是构造模型，而没有真正的运行。
#这跟上面的numpy库每一次都是真正执行是截然不同的区别
```

```
# 请参考正文，我们假定房价的公式为：y=a*x+b

#tf.Variable是在TensorFlow中定义一个变量的意思
#我们这里简单起见，人为给a/b两个初始值，都是0.3，注意这也是相当于规范化之后的数值
b = tf.Variable(np.float32(0.3))
a = tf.Variable(np.float32(0.3))

#这是定义主要的数学模型，模型来自于上面的公式
#注意这里必须使用tf的公式，这样的公式才是模型
#上面使用np的是直接计算，而不是定义模型
# TensorFlow的函数名基本就是完整英文，你应当能读懂
y_value = tf.multiply(x,a) + b

# 这里是代价函数，同我们文中所讲的唯一区别是用平方来取代求绝对值，
#目标都是为了得到一个正数值，功能完全相同，
#平方计算起来会更快更容易,这种方式也称为"方差"
loss = tf.reduce_mean(input_tensor=tf.square(y_value - y))
# TensorFlow内置的梯度下降算法，每步长0.5
optimizer = tf.compat.v1.train.GradientDescentOptimizer(0.5)
# 代价函数值最小化的时候，代表求得解
train = optimizer.minimize(loss)

# 初始化所有变量，也就是上面定义的a/b两个变量
init = tf.compat.v1.global_variables_initializer()

#启动图
sess = tf.compat.v1.Session()
#真正的执行初始化变量，还是老话，上面只是定义模型，并没有真正开始执行
sess.run(init)

#重复梯度下降200次，每隔5次打印一次结果
for step in range(0, 200):
    sess.run(train)
    if step % 5 == 0:
        print(step, sess.run(loss),sess.run(a), sess.run(b))
```

转换之后，代码中的注释部分会完美的保留，喜欢用代码来代替文档的程序员可以放心。所有2.0中变更了的类或者方法，转换工具将使用tensorflow.compat.v1中的对应类或方法来替代，比如：

```
optimizer = tf.compat.v1.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)
init = tf.compat.v1.global_variables_initializer()
```

所以从本质上，这种方式跟第一种方法，采用tensorflow.compat.v1包作为tensorflow替代包的方式是完全相同的。

编写原生的**TensorFlow 2.0**程序

推荐的演进方式，当然还是学习TensorFlow 2.0的相关特征，重构原有代码为新版本代码才是正路。平心而论，毕竟绝大多数系统的升级都是为了提供更多功能和降低使用门槛。TensorFlow 2.0也是大幅的降低了使用门槛的。大多数的工作比起1.x版本来，都能使用更少的代码量来完成。
首先了解一下TensorFlow 2.0同1.x之间的重要区别：

- 在API层面的类、方法有了较大的变化，这个需要在使用中慢慢熟悉
- 取消了Session机制，每一条命令直接执行，而不需要等到Session.run
- 因为取消了Session机制，原有的数学模型定义，改为使用Python函数编写。原来的feed_dict和tf.placeholder，成为了函数的输入部分；原来的fetches，则成为了函数的返回值。

- 使用keras的模型体系对原有的TensorFlow API进行高度的抽象，使用更容易
- 使用tf.keras.Model.fit来替代原有的训练循环。

正常情况下，最后一项tf.keras.Model.fit能够大大的降低训练循环的代码量。但在本例中，我们模拟了一个现实中并不适用的例子，keras中并未对这种情形进行优化。所以在本例中反而无法使用tf.keras.Model.fit（实际上一定要使用也是可以的，不过要自定义模型，工作量更不划算）。因此本例中仍然要自己编写训练循环。并且因为2.0中API的变化，代码更复杂了。不过相信我，等到比较正式应用中，使用神经网络、卷积等常用算法，代码就极大的简化了。
使用TensorFlow 2.0原生代码的程序代码如下：

```
#!/usr/bin/env python3
#上面一行改为使用python3解释本代码

#引入python新版本的语言特征
from __future__ import absolute_import, division, print_function

#引入TensorFlow库,版本2.0
import tensorflow as tf

#引入数值计算库
import numpy as np

#使用 NumPy 生成假数据集x,代表房间的平米数，这里的取值范围是0-1的浮点数，
#原因请看正文中的说明，属于是"规范化"之后的数据
# 生成的数据共100个，式样是100行，每行1个数据
x = np.float32(np.random.rand(100,1))
#我们假设每平米0.5万元，基础费用0.7万，这个数值也是规范化之后的，仅供示例
#最终运行的结果，应当求出来0.5/0.7这两个值代表计算成功
#计算最终房价y，x和y一同当做我们的样本数据
y = np.dot(x,0.5) + 0.7
#---------------------------------数据集准备完成
# 请参考正文，我们假定房价的公式为：y=a*x+b
#定义tensorflow变量，a是权重，b是偏移
b = tf.Variable(np.float32(0.3))
a = tf.Variable(np.float32(0.3))

#以上代码基本同tensorflow1.x版本一致
#以下有了区别
#使用python语言定义数学模型，模型来自于上面的公式
#上面使用np的是直接计算得到训练样本，而不是定义模型
#模型中并非必须使用tensorflow的计算函数来代替python的乘法运算
@tf.function
def model(x):
        return a*x+b

#定义代价函数，也是python函数
def loss(predicted_y, desired_y):
    return tf.reduce_sum(tf.square(predicted_y - desired_y))

# TensorFlow内置Adam算法，每步长0.1
optimizer = tf.optimizers.Adam(0.1)
# 还可以选用TensorFlow内置SGD(随机最速下降)算法，每步长0.001
#不同算法要使用适当的步长，步长过大会导致模型无法收敛
#optimizer = tf.optimizers.SGD(0.001)

#重复梯度下降200次，每隔5次打印一次结果
for step in range(0, 200):
```

```
        with tf.GradientTape() as t:
                outputs = model(x)  #进行一次计算
                current_loss = loss(outputs, y) #得到当前损失值
                grads = t.gradient(current_loss, [a, b])     #调整模型中的权重、偏移值
                optimizer.apply_gradients(zip(grads,[a, b]))     #调整之后的值代回到模型
        if step % 5 == 0:    #每5次迭代显示一次结果
                print( "Step:%d loss:%%%2.5f weight:%2.7f bias:%2.7f " %
                        (step,current_loss.numpy(), a.numpy(), b.numpy()))
```

程序在升级中所做的修改和特殊的处理，都使用注释保留在了源码中。我觉得这种方式比打散摘出来讲解的能更透彻。
最后看一下新版程序的执行结果：

```
Step:0 loss:%25.78244 weight:0.4000000 bias:0.4000000
Step:5 loss:%2.71975 weight:0.7611420 bias:0.7740188
Step:10 loss:%3.09600 weight:0.6725605 bias:0.7224629
Step:15 loss:%0.87834 weight:0.4931822 bias:0.5800986
Step:20 loss:%1.24737 weight:0.4960071 bias:0.6186275
Step:25 loss:%0.22444 weight:0.5730734 bias:0.7264798
Step:30 loss:%0.47145 weight:0.5464076 bias:0.7252067
Step:35 loss:%0.09156 weight:0.4736322 bias:0.6712209
Step:40 loss:%0.14845 weight:0.4771673 bias:0.6866464
Step:45 loss:%0.06199 weight:0.5101752 bias:0.7255269
Step:50 loss:%0.03108 weight:0.4946054 bias:0.7112849
Step:55 loss:%0.04115 weight:0.4770990 bias:0.6918764
Step:60 loss:%0.00145 weight:0.4950625 bias:0.7060429
Step:65 loss:%0.01781 weight:0.5029647 bias:0.7096580
Step:70 loss:%0.00211 weight:0.4934593 bias:0.6963260
Step:75 loss:%0.00298 weight:0.4983235 bias:0.6982682
Step:80 loss:%0.00345 weight:0.5049748 bias:0.7031375
Step:85 loss:%0.00004 weight:0.5001755 bias:0.6976562
Step:90 loss:%0.00102 weight:0.5002422 bias:0.6978318
Step:95 loss:%0.00065 weight:0.5029225 bias:0.7010939
Step:100 loss:%0.00001 weight:0.5000774 bias:0.6990223
Step:105 loss:%0.00021 weight:0.4996552 bias:0.6993059
Step:110 loss:%0.00015 weight:0.5007215 bias:0.7008768
Step:115 loss:%0.00000 weight:0.4993480 bias:0.6997767
Step:120 loss:%0.00003 weight:0.4995552 bias:0.7000407
Step:125 loss:%0.00004 weight:0.5001000 bias:0.7004969
Step:130 loss:%0.00001 weight:0.4995880 bias:0.6998325
Step:135 loss:%0.00000 weight:0.4999941 bias:0.7000810
Step:140 loss:%0.00001 weight:0.5001197 bias:0.7000892
Step:145 loss:%0.00000 weight:0.4999250 bias:0.6998329
Step:150 loss:%0.00000 weight:0.5001498 bias:0.7000451
Step:155 loss:%0.00000 weight:0.5000388 bias:0.6999565
Step:160 loss:%0.00000 weight:0.4999948 bias:0.6999494
Step:165 loss:%0.00000 weight:0.5000526 bias:0.7000424
Step:170 loss:%0.00000 weight:0.4999576 bias:0.6999717
Step:175 loss:%0.00000 weight:0.4999971 bias:0.7000214
Step:180 loss:%0.00000 weight:0.4999900 bias:0.7000131
Step:185 loss:%0.00000 weight:0.4999775 bias:0.6999928
Step:190 loss:%0.00000 weight:0.5000094 bias:0.7000152
Step:195 loss:%0.00000 weight:0.4999923 bias:0.6999906
```

模型通过学习后，得到的结果是很接近我们的预设值的。
程序中还可以考虑使用随机快速下降算法(SGD)，你可以把当前的Adam算法使用注释符屏蔽上，打开SGD算法的注释屏蔽来尝试一下。对于本例中的数据集来讲，SGD的下降步长需要的更小，同样循环次数下，求解的精度

会低一些。可以看出对于本例，Adam算法显然是更有效率的。而在TensorFlow的支持下，对于同样的数据集和数学模型，变更学习算法会很容易。