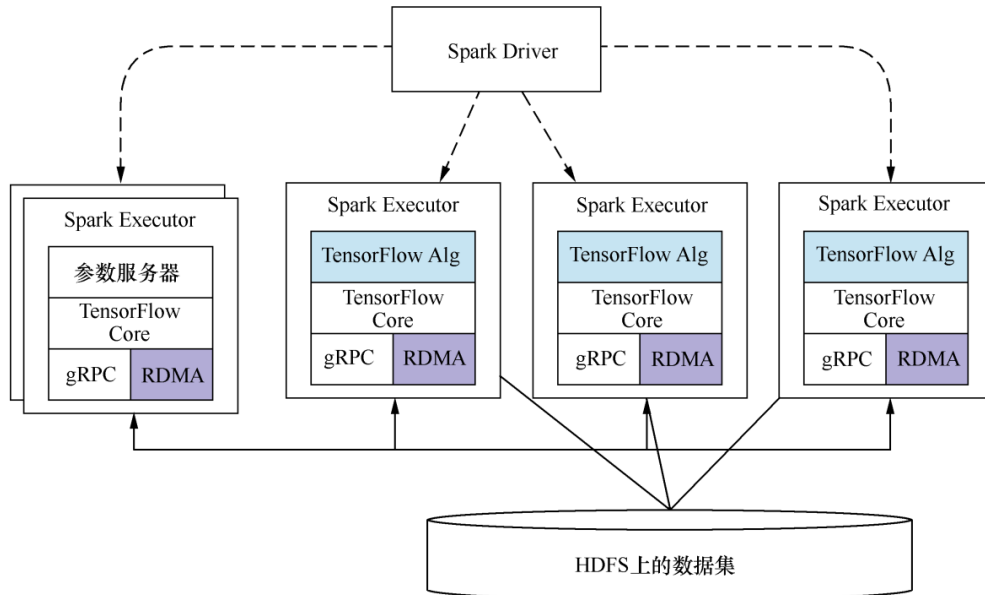
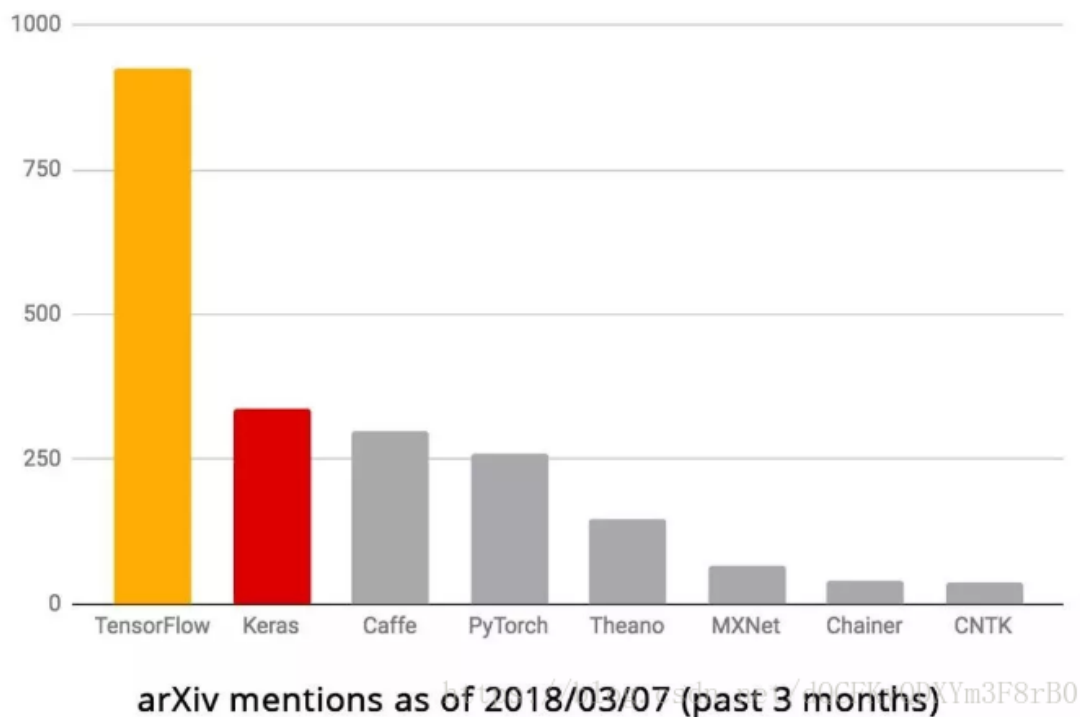


TensorFlow On Spark (TFoS) 测试指南

1.1 TFoS 系统架构图



TF.learn 是关于 Tensorflow 的高阶 API 库，里面的 Log 同时可被 Tensorboard 可视化，因为它是高阶框架，所以只需要写很少的代码，就可以实现一个神经网络。和 TFLearn 类似的深度学习高阶框架，还有 [Keras](#) 和 [tensrolayer](#)。



1.2 开发编译必备软件

目前服务器上已安装软件的版本如下：

CentOS: 7.5
 java:jdk 1.8
 scala:2.11.8
 hadoop:2.7.3
 spark:2.0.1
 python:2.7.5
 Git:1.8

1.3 Tensorflow 编译安装

`pip install --target=d:\somewhere\other\than\the\default package_name`

在/opt/diyu/hadoop-system 目录下执行：

`pip install tensorflow-1.9.0-cp27-cp27mu-manylinux1_x86_64.whl`

```
[root@htl.r1.n11 hadoop-system]$ll
total 50000
drwx--x--x 11 root root      4096 Jan  6 13:31 apache-hive-2.3.4-bin
drwx--x--x  9 197608 197121    4096 Dec 25 16:36 apache-tomcat
drwx--x--x 11 197608 197121    4096 Dec 25 16:38 hadoop-2.7.3
drwx--x--x 10 197608 197121    4096 Feb 23 15:15 hive-2.1.1
drwx--x--x  8 197608 197121    4096 Dec 25 16:36 jdk1.8.0_131
drwx--x--x  2 197608 197121    4096 Dec 25 16:36 mysql-rpm
drwxr-xr-x  3 root root      4096 Feb 27 13:46 scala
drwx--x--x 13 197608 197121    4096 Jan 13 12:39 spark-2.0.1
drwx--x--x 13 root root      4096 Jan  6 13:31 spark-2.0.1-bin-hadoop2-without-hive
-rw-r--r--  1 root root    51152262 Feb 28 17:46 tensorflow-1.9.0-cp27-cp27mu-manylinux1_x86_64.whl
drwxr-xr-x 13 root root      4096 Feb 28 16:59 tensorflow-gpu
drwx--x--x 11 root root      4096 Jan  8 10:50 tpc-ds_2.9.0rc2
```

安装完成后查看一下tensorflow的版本和安装路径：

```
[root@htl.r1.n11 hadoop-system]$python
Python 2.7.5 (default, Oct 30 2018, 23:45:53)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-36)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow as tf
>>> tf.__version__
'1.9.0'
>>> tf.__path__
['/usr/lib/python2.7/site-packages/tensorflow']
>>> █
```

注意：网上安装教程tensorflow一般都使用0.12.1版本，在后面测试命令运行时会报AttributeError: 'module' object has no attribute 'data'的错误，必须使用1.4以后的版本。

1.4 下载 TensorflowOnSpark 源码

```
git clone https://github.com/yahoo/TensorFlowOnSpark.git
cd TensorFlowOnSpark
export TFoS_HOME=$(pwd)
zip -r tfspark.zip tensorflowonspark/*
```

```

[root@ht1.r1.n11 hadoop-system]$cd TensorFlowOnSpark
[root@ht1.r1.n11 TensorFlowOnSpark]$pwd
/opt/wotung/hadoop-system/TensorFlowOnSpark
[root@ht1.r1.n11 TensorFlowOnSpark]$ll
total 136
-rw-r--r-- 1 root root 3355 Feb 28 15:46 1.txt
-rw-r--r-- 1 root root 7510 Feb 27 14:23 Code-of-Conduct.md
-rw-r--r-- 1 root root 1665 Feb 27 14:23 Contributing.md
drwxr-xr-x 3 root root 4096 Feb 27 14:23 docs
drwxr-xr-x 7 root root 4096 Feb 28 13:40 examples
-rw-r--r-- 1 root root 1 Feb 28 17:00 executor_id
drwxr-xr-x 2 root root 4096 Feb 27 14:23 lib
-rw-r--r-- 1 root root 9210 Feb 27 14:23 LICENSE
-rwxr-xr-x 1 root root 6040 Feb 28 16:35 mnist_dist.py
-rw-r--r-- 1 root root 5431 Feb 28 16:59 mnist_dist.pyc
drwxr-xr-x 2 root root 4096 Feb 28 15:58 mnist_model
-rwxr-xr-x 1 root root 3245 Feb 27 17:10 mnist_spark.py
-rw-r--r-- 1 root root 7137 Feb 27 14:23 pom.xml
-rw-r--r-- 1 root root 4392 Feb 27 14:23 README.md
-rw-r--r-- 1 root root 44 Feb 27 14:23 requirements.txt
drwxr-xr-x 3 root root 4096 Feb 27 14:23 scripts
-rw-r--r-- 1 root root 69 Feb 27 14:23 setup.cfg
-rw-r--r-- 1 root root 1029 Feb 27 14:23 setup.py
drwxr-xr-x 4 root root 4096 Feb 27 14:23 src
drwxr-xr-x 2 root root 4096 Feb 28 17:00 tensorboard_1
drwxr-xr-x 2 root root 4096 Feb 28 16:45 tensorflowonspark
drwxr-xr-x 2 root root 4096 Feb 27 14:23 test
-rw-r--r-- 1 root root 16409 Feb 27 14:30 tfspark.zip
[root@ht1.r1.n11 TensorFlowOnSpark]$

```

1.5 测试数据准备

下载 mnist 测试数据集

```

wget http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz
wget http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz
wget http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz
wget http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz

```

数据文件放到 TensorFlowOnSpark/examples/mnist 中

```

[root@ht1.r1.n11 mnist]$ll
total 30244
drwxr-xr-x 2 root root 4096 Feb 27 15:06 csv
drwxr-xr-x 2 root root 4096 Feb 27 14:23 estimator
drwxr-xr-x 2 root root 4096 Feb 27 14:23 keras
-rw-r--r-- 1 root root 5743 Feb 27 14:23 mnist_data_setup.py
-rw-r--r-- 1 root root 20082 Feb 27 14:23 mnist_pipeline.ipynb
-rw-r--r-- 1 root root 10872 Feb 27 14:23 mnist_spark.ipynb
-rw-r--r-- 1 root root 9722 Feb 27 14:23 mnist_tf.ipynb
-rw-r--r-- 1 root root 437 Feb 27 14:23 README.md
drwxr-xr-x 2 root root 4096 Feb 27 17:13 spark
drwxr-xr-x 2 root root 4096 Feb 27 14:23 streaming
-rw-r--r-- 1 root root 4422102 Feb 27 14:57 t10k-images-idx3-ubyte.gz
-rw-r--r-- 1 root root 5148 Feb 27 14:57 t10k-labels-idx1-ubyte.gz
drwxr-xr-x 2 root root 4096 Feb 27 14:23 tf
-rw-r--r-- 1 root root 26421880 Feb 27 14:57 train-images-idx3-ubyte.gz
-rw-r--r-- 1 root root 29515 Feb 27 14:57 train-labels-idx1-ubyte.gz
[root@ht1.r1.n11 mnist]$pwd
/opt/wotung/hadoop-system/TensorFlowOnSpark/examples/mnist
[root@ht1.r1.n11 mnist]$

```

我们在解 tar 时，包内已含所有 example 的例子：

Under /opt/diyu/Hadoop-system:

```
tar -xvf TensorFlowOnSpark.tar.gz
```

1.6 设置 TensorFlowOnSpark 根目录的环境变量

```
cd TensorFlowOnSpark
```

```
export TFoS_HOME=$(pwd)
```

接着，启动 Spark 主节点（master）：

```
${SPARK_HOME}/sbin/start-master.sh
```

配置两个工作节点（worker）实例，通过 master-spark-URL 和主节点连接：

```
export MASTER=spark://$(hostname):7077
```

```
export SPARK_WORKER_INSTANCES=2
```

```
export CORES_PER_WORKER=1
```

```
export TOTAL_CORES=$(( ${CORES_PER_WORKER} * ${SPARK_WORKER_INSTANCES} ))
```

```
${SPARK_HOME}/sbin/start-slave.sh -c $CORES_PER_WORKER -m 3G $MASTER
```

1.7 MNIST 的 Zip 原始数据集转换为 HDFS 的 RDD csv 格式（用户执行）

```
cd ${TFoS_HOME}
```

```
rm -rf examples/mnist/csv
```

```
cd examples
```

```
/opt/diyu/hadoop-system/spark-2.0.1/bin/spark-submit --master=local[*]
```

```
${TFoS_HOME}/examples/mnist/mnist_data_setup.py --output
```

```
examples/mnist/csv --format csv
```

(local[*]改成 yarn for the cluster environment)

(Local[*] 改成\${MASTER} standalone)

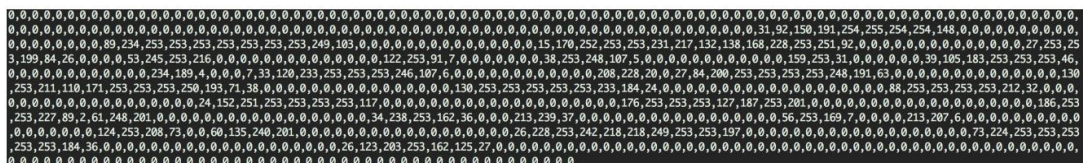
运行完成后可以看生成的文件（图片和标记向量）：

```
[root@htl.r1.n11 hadoop-system]$hadoop fs -ls /mnist/csv
Found 2 items
drwxrwxrwx - root root 4096 2019-03-01 10:33 /mnist/csv/test
drwxrwxrwx - root root 4096 2019-03-01 10:31 /mnist/csv/train
[root@htl.r1.n11 hadoop-system]$hadoop fs -ls /mnist/csv/test
Found 2 items
drwxrwxrwx - root root 4096 2019-03-01 10:33 /mnist/csv/test/images
drwxrwxrwx - root root 4096 2019-03-01 10:33 /mnist/csv/test/labels
[root@htl.r1.n11 hadoop-system]$hadoop fs -ls /mnist/csv/test/images
Found 11 items
-rw-r--r-- 3 root root 0 2019-03-01 10:33 /mnist/csv/test/images/_SUCCESS
-rw-r--r-- 3 root root 2221759 2019-03-01 10:33 /mnist/csv/test/images/part-00000
-rw-r--r-- 3 root root 2216293 2019-03-01 10:33 /mnist/csv/test/images/part-00001
-rw-r--r-- 3 root root 2220481 2019-03-01 10:33 /mnist/csv/test/images/part-00002
-rw-r--r-- 3 root root 2210866 2019-03-01 10:33 /mnist/csv/test/images/part-00003
-rw-r--r-- 3 root root 2223370 2019-03-01 10:33 /mnist/csv/test/images/part-00004
-rw-r--r-- 3 root root 2217034 2019-03-01 10:33 /mnist/csv/test/images/part-00005
-rw-r--r-- 3 root root 2207009 2019-03-01 10:33 /mnist/csv/test/images/part-00006
-rw-r--r-- 3 root root 2216107 2019-03-01 10:33 /mnist/csv/test/images/part-00007
-rw-r--r-- 3 root root 2214443 2019-03-01 10:33 /mnist/csv/test/images/part-00008
-rw-r--r-- 3 root root 2228709 2019-03-01 10:33 /mnist/csv/test/images/part-00009
[root@htl.r1.n11 hadoop-system]$hadoop fs -ls /mnist/csv/train
Found 2 items
drwxrwxrwx - root root 4096 2019-03-01 10:31 /mnist/csv/train/images
drwxrwxrwx - root root 4096 2019-03-01 10:32 /mnist/csv/train/labels
[root@htl.r1.n11 hadoop-system]$hadoop fs -ls /mnist/csv/train/images
Found 11 items
-rw-r--r-- 3 root root 0 2019-03-01 10:32 /mnist/csv/train/images/_SUCCESS
-rw-r--r-- 3 root root 11325762 2019-03-01 10:32 /mnist/csv/train/images/part-00000
-rw-r--r-- 3 root root 13604125 2019-03-01 10:32 /mnist/csv/train/images/part-00001
-rw-r--r-- 3 root root 13593080 2019-03-01 10:32 /mnist/csv/train/images/part-00002
-rw-r--r-- 3 root root 13613443 2019-03-01 10:32 /mnist/csv/train/images/part-00003
-rw-r--r-- 3 root root 13600331 2019-03-01 10:32 /mnist/csv/train/images/part-00004
-rw-r--r-- 3 root root 13610984 2019-03-01 10:32 /mnist/csv/train/images/part-00005
-rw-r--r-- 3 root root 13591592 2019-03-01 10:32 /mnist/csv/train/images/part-00006
-rw-r--r-- 3 root root 13605529 2019-03-01 10:32 /mnist/csv/train/images/part-00007
-rw-r--r-- 3 root root 13630341 2019-03-01 10:32 /mnist/csv/train/images/part-00008
-rw-r--r-- 3 root root 12713686 2019-03-01 10:32 /mnist/csv/train/images/part-00009
[root@htl.r1.n11 hadoop-system]$
```

MNIST 训练集共有 60 000 条数据，有 10 个文件，每个文件有 6 000 条数据左右，里面存储的格式和标记向量格式如图所示。



图片向量格式如图所示。



这里，我们主要是把训练集和测试集分别保存成 RDD 数据，具体代码本见\${TFoS_HOME}/

examples/mnist/mnist_data_setup.py。关键代码如下：

```
writeMNIST(sc, "mnist/train-images-idx3-ubyte.gz", "mnist/train-labels-idx1-ubyte.gz", args.output + "/train", args.format, args.num_partitions)
writeMNIST(sc, "mnist/t10k-images-idx3-ubyte.gz", "mnist/t10k-labels-idx1-ubyte.gz", args.output + "/test", args.format, args.num_partitions)
```

调用 writeMNIST 函数，将 RDDs 保存为特定格式：

```
def writeMNIST(sc, input_images, input_labels, output, format, num_partitions):
```

```
    """将 MNIST 图像和标记向量写入 HDFS 上"""
```

```
    with open(input_images, 'rb') as f:
```

```
        images = numpy.array(mnist.extract_images(f))
```

```
    with open(input_labels, 'rb') as f:
```

```
        labels = numpy.array(mnist.extract_labels(f, one_hot=True))
```

```
    shape = images.shape
```

```
    print("images.shape: {0}".format(shape)) # 60000 x 28 x 28
```

```
    print("labels.shape: {0}".format(labels.shape)) # 60000 x 10
```

```
    imageRDD = sc.parallelize(images.reshape(shape[0], shape[1] * shape[2]),
```

```
    num
```

```
    _partitions)
```

```
    labelRDD = sc.parallelize(labels, num_partitions)
```

```
    output_images = output + "/images"
```

```
    output_labels = output + "/labels"
```

```
    # 将 RDDs 保存为特定格式
```

```
    if format == "pickle":
```

```
        imageRDD.saveAsPickleFile(output_images)
```

```
        labelRDD.saveAsPickleFile(output_labels)
```

```
    elif format == "csv":
```

```
        imageRDD.map(toCSV).saveAsTextFile(output_images)
```

```
        labelRDD.map(toCSV).saveAsTextFile(output_labels)
```

接着，提交训练任务，开始训练，命令如下，我们最终在 HDFS 上生成了 mnist_model：

3.3 训练前的准备工作 (代码已改好)

```
cd ${TFoS_HOME}
```

```
cp examples/mnist/spark/mnist_dist.py ./
```

```
cp examples/mnist/spark/mnist_spark.py ./
```

mnist_dist.py 和 mnist_spark.py 与 tensorflowonspark 目录在同一层目录下，否则执行 mnist_spark.py 的 17 行

from tensorflowonspark import TFCluster 会报错。

```
[root@ht1.r1.n11 TensorFlowOnSpark]$ll
total 132
-rw-r--r-- 1 root root 7510 Feb 27 14:23 Code-of-Conduct.md
-rw-r--r-- 1 root root 1665 Feb 27 14:23 Contributing.md
drwxr-xr-x 3 root root 4096 Feb 27 14:23 docs
drwxr-xr-x 7 root root 4096 Feb 28 13:40 examples
-rw-r--r-- 1 root root 1 Feb 28 17:00 executor_id
drwxr-xr-x 2 root root 4096 Feb 27 14:23 lib
-rw-r--r-- 1 root root 9210 Feb 27 14:23 LICENSE
-rwxr-xr-x 1 root root 6040 Feb 28 16:35 mnist_dist.py
-rw-r--r-- 1 root root 5431 Feb 28 16:59 mnist_dist.pyc
drwxr-xr-x 2 root root 4096 Feb 28 15:58 mnist_model
-rwxr-xr-x 1 root root 3245 Feb 27 17:10 mnist_spark.py
-rw-r--r-- 1 root root 7137 Feb 27 14:23 pom.xml
-rw-r--r-- 1 root root 4392 Feb 27 14:23 README.md
-rw-r--r-- 1 root root 44 Feb 27 14:23 requirements.txt
drwxr-xr-x 3 root root 4096 Feb 27 14:23 scripts
-rw-r--r-- 1 root root 69 Feb 27 14:23 setup.cfg
-rw-r--r-- 1 root root 1029 Feb 27 14:23 setup.py
drwxr-xr-x 4 root root 4096 Feb 27 14:23 src
drwxr-xr-x 2 root root 4096 Feb 28 17:00 tensorboard_1
drwxr-xr-x 2 root root 4096 Feb 28 16:45 tensorflowonspark
drwxr-xr-x 2 root root 4096 Feb 27 14:23 test
-rw-r--r-- 1 root root 16409 Feb 27 14:30 tfspark.zip
[root@ht1.r1.n11 TensorFlowOnSpark]$
```

mnist_dist.py

99 行 : logdir = ctx.absolute_path(args.model) 改成 logdir =
'/tmp/'+args.model

136 行: done_dir = "{}/{}/done".format(ctx.absolute_path(args.model),
args.mode) 改成 done_dir = "{}/{}/done".format(logdir, args.mode)

服务器上要有/tmp 目录，会在该目录下生成中间临时文件。

否则会报 UnimplementedError: File system scheme 'parafs' not
implemented (file: 'parafs://localhost:4500/mnist_model')的错误。

注意点：修改 py 文件后要把 pyc 文件删除，下次执行才会重新编译

使用新文件。如果不删除有时候会报'AutoProxy[get_queue]' object has no attribute 'put' 错误。

最好把 `logdir` 设置成 `None`，否则跑 `yarn` 容易出超时错误。如果使用 `tmp` 做临时文件目录，需要把 `/tmp` 目录设置成 `777`。

会报错 Environment variable HADOOP_HDFS_HOME not set

配置环境变量 HADOOP_HDFS_HOME 和 HADOOP_HOME 一致

加环境变量：（用户）

```
# HADOOP
export HADOOP_HOME=$HADOOP_PARAFS_HOME/hadoop-2.7.3
export HADOOP_HDFS_HOME=$HADOOP_PARAFS_HOME/hadoop-2.7.3
export PATH=${HADOOP_HOME}/sbin:${HADOOP_HOME}/bin:$PATH
```

训练用户 CASE:

执行命令：

`Standalone` 的模式下，`spark` 的资源管理和调度是自己来管理和调度的，主要由 `master` 来管理。:

```

${SPARK_HOME}/bin/spark-submit \
--master ${MASTER} \
--py-files
${TFoS_HOME}/tfspark.zip,${TFoS_HOME}/examples/mnist/spark/mnist_dist.py \
--conf spark.cores.max=${TOTAL_CORES} \
--conf spark.task.cpus=${CORES_PER_WORKER} \
--conf spark.executorEnv.JAVA_HOME="$JAVA_HOME" \
${TFoS_HOME}/examples/mnist/spark/mnist_spark.py \
--cluster_size ${SPARK_WORKER_INSTANCES} \
--images examples/mnist/csv/train/images \
--labels examples/mnist/csv/train/labels \
--format csv \
--mode train \
--model mnist_model

/opt/diyu/hadoop-system/spark-2.0.1/bin/spark-submit \
--master=yarn \
--py-files \
/opt/diyu/hadoop-system/TensorFlowOnSpark/tfspark.zip,mnist_dist.py \
--conf spark.cores.max=4 \
--conf spark.task.cpus=2 \
--conf spark.executorEnv.JAVA_HOME="$JAVA_HOME" mnist_spark.py \
--cluster_size 2 \
--images mnist/csv/train/images \
--labels mnist/csv/train/labels \
--format csv \
--mode train \
--model mnist_model

```

Spark-yarn

Client 提交任务给 resourceManager, resourceManager 会选择一台机器开启一个 container, 在 container 里面开启一个 applicationmaster 服务进程, applicationMaster 进行任务的管理和调度, applicationMaster 会向 resourceManager 申请资源, resourcemanager 会在其他的机器上开启 container 进行资源分配。applicationMaster 在 resourcemanager 分配的资源进行任务调度, 在 container 里面运行 task (map 和 reduce)。

Spark 集群基于 yarn 的时候任务的执行流程:

(1) client 模式

Client 提交任务给 resourceManager，在提交任务的时候，在提交任务的那台机器上面开启一个 driver 服务进程，resourceManager 在接收到 client 提交的任务以后，在集群中随机选择一台机器分配一个 container，在该 container 里面开启一个 applicationmaster 服务进程，driver 去找 applicationmaster，applicationmaster 去找 resourceManager 申请资源，resourceManager 会分配 container，在其中开启 excuter，excuter 会反向向 driver 注册，driver 把 task 放入到 excuter 里面执行。

(2) Cluster 模式

Spark 集群会在集群中开启一个 driver，此时开启就是 applicationmaster 和 driver 合二为一了。其他的都相同。

注：Standalone 和 yarn 上运行的业务的执行流程都是相同的，只是资源的分配和管理的方式不一样了。这里不讨论 SPARK-Mesos

集群：

```
/opt/diyu/hadoop-system/spark-2.0.1/bin/spark-submit \  
--master=yarn \  
--py-files \  
/opt/diyu/hadoop-system/TensorFlowOnSpark/tfspark.zip,mnist_dist.py \  
--conf spark.cores.max=3 \  
--conf spark.task.cpus=1 \  
--conf spark.executorEnv.JAVA_HOME="$JAVA_HOME" mnist_spark.py \  
--cluster_size 3 \  
--images mnist/csv/train/images \  
--labels mnist/csv/train/labels \  
--format csv \  
--mode train \  
--model mnist_model
```

spark.cores.max, spark.cores.cpus, - cluster_size 这些参数可根据集群实际情况进行配置。和 spark-defaults.conf 文件里的配置要一致。 $cluster_size = spark.cores.max / spark.cores.cpus$ 。

执行后可以去/tmp 目录查看

```

[root@htl.r1.n11 mnist_model]$cd /tmp
[root@htl.r1.n11 tmp]$ll mnist_model
total 2112
-rw-r--r-- 1 root root 179 Mar 1 10:56 checkpoint
-rw-r--r-- 1 root root 235511 Mar 1 10:56 events.out.tfevents.1551408754.htl.r1.n11
-rw-r--r-- 1 root root 153472 Mar 1 10:52 graph.pbtxt
-rw-r--r-- 1 root root 814168 Mar 1 10:52 model.ckpt-0.data-00000-of-00001
-rw-r--r-- 1 root root 375 Mar 1 10:52 model.ckpt-0.index
-rw-r--r-- 1 root root 60979 Mar 1 10:52 model.ckpt-0.meta
-rw-r--r-- 1 root root 814168 Mar 1 10:56 model.ckpt-594.data-00000-of-00001
-rw-r--r-- 1 root root 375 Mar 1 10:56 model.ckpt-594.index
-rw-r--r-- 1 root root 60979 Mar 1 10:56 model.ckpt-594.meta
drwxr-xr-x 3 root root 4096 Mar 1 10:56 train
[root@htl.r1.n11 tmp]$ll mnist_model/train
total 4
drwxr-xr-x 2 root root 4096 Mar 1 10:56 done
[root@htl.r1.n11 tmp]$

```

这里的 `mnist_dist.py` 主要是构建 TensorFlow 分布式任务，其中定义了分布式任务的主函数，也就是启动 TensorFlow 的主函数 `map_fun`，采用的数据获取方式是 `Feeding`。这里用到的 `TensorFlowOnSpark` 代码主要是获取 TensorFlow 集群和服务器实例，如下：

```
cluster, server = TFNode.start_cluster_server(ctx, 1, args.rdma)
```

其中 `TFNode` 调用我们刚才打包好的 `tfspark.zip` 中的 `TFNode.py` 文件。

`mnist_spark.py` 文件是我们训练的主程序，体现了 `TensorFlowOnSpark` 的部署步骤，如下：

3.4 预测的用户例子

用户执行命令：

```

${SPARK_HOME}/bin/spark-submit \
--master ${MASTER} \
--py-files
${TFoS_HOME}/tfspark.zip,${TFoS_HOME}/examples/mnist/spark/mnist_dist.py \
--conf spark.cores.max=${TOTAL_CORES} \
--conf spark.task.cpus=${CORES_PER_WORKER} \
--conf spark.executorEnv.JAVA_HOME="$JAVA_HOME" \
${TFoS_HOME}/examples/mnist/spark/mnist_spark.py \
--cluster_size ${SPARK_WORKER_INSTANCES} \
--images examples/mnist/csv/test/images \
--labels examples/mnist/csv/test/labels \
--mode inference \
--format csv \
--model mnist_model \
--output predictions

/opt/diyu/hadoop-system/spark-2.0.1/bin/spark-submit \
--master=local[*] \
--py-files \
/opt/diyu/hadoop-system/TensorFlowOnSpark/tfspark.zip,mnist_dist.py \
--conf spark.cores.max=4 \
--conf spark.task.cpus=2 \

```

```

--conf spark.executorEnv.JAVA_HOME="$JAVA_HOME" mnist_spark.py \
--cluster_size 2 \
--images mnist/csv/test/images \
--labels mnist/csv/test/labels \
--mode inference \
--format csv \
--model mnist_model \
--output predictions

```

到/tmp 下可以看生成的临时文件

```

[root@ht1.r1.n11 tmp]$ll mnist_model
total 1468
-rw-r--r-- 1 root root 115 Mar 1 11:10 checkpoint
-rw-r--r-- 1 root root 186190 Mar 1 11:07 events.out.tfevents.1551409675.ht1.r1.n11
-rw-r--r-- 1 root root 71502 Mar 1 11:09 events.out.tfevents.1551409756.ht1.r1.n11
-rw-r--r-- 1 root root 196432 Mar 1 11:11 events.out.tfevents.1551409833.ht1.r1.n11
-rw-r--r-- 1 root root 153472 Mar 1 11:10 graph.pbtxt
drwxr-xr-x 3 root root 4096 Mar 1 11:11 inference
-rw-r--r-- 1 root root 814168 Mar 1 11:10 model.ckpt-0.data-00000-of-00001
-rw-r--r-- 1 root root 375 Mar 1 11:10 model.ckpt-0.index
-rw-r--r-- 1 root root 60979 Mar 1 11:10 model.ckpt-0.meta
[root@ht1.r1.n11 tmp]$ll mnist_model/inference
total 4
drwxr-xr-x 2 root root 4096 Mar 1 11:11 done
[root@ht1.r1.n11 tmp]$

```

预测生成的文件:

```

[root@ht1.r1.n11 hadoop-system]$hadoop fs -ls /predictions
Found 11 items
-rw-r--r-- 3 root root 0 2019-03-01 11:11 /predictions/_SUCCESS
-rw-r--r-- 3 root root 51000 2019-03-01 11:10 /predictions/part-00000
-rw-r--r-- 3 root root 51000 2019-03-01 11:10 /predictions/part-00001
-rw-r--r-- 3 root root 51000 2019-03-01 11:10 /predictions/part-00002
-rw-r--r-- 3 root root 51000 2019-03-01 11:11 /predictions/part-00003
-rw-r--r-- 3 root root 51000 2019-03-01 11:11 /predictions/part-00004
-rw-r--r-- 3 root root 51000 2019-03-01 11:11 /predictions/part-00005
-rw-r--r-- 3 root root 51000 2019-03-01 11:11 /predictions/part-00006
-rw-r--r-- 3 root root 51000 2019-03-01 11:11 /predictions/part-00007
-rw-r--r-- 3 root root 51000 2019-03-01 11:11 /predictions/part-00008
-rw-r--r-- 3 root root 51000 2019-03-01 11:11 /predictions/part-00009
[root@ht1.r1.n11 hadoop-system]$

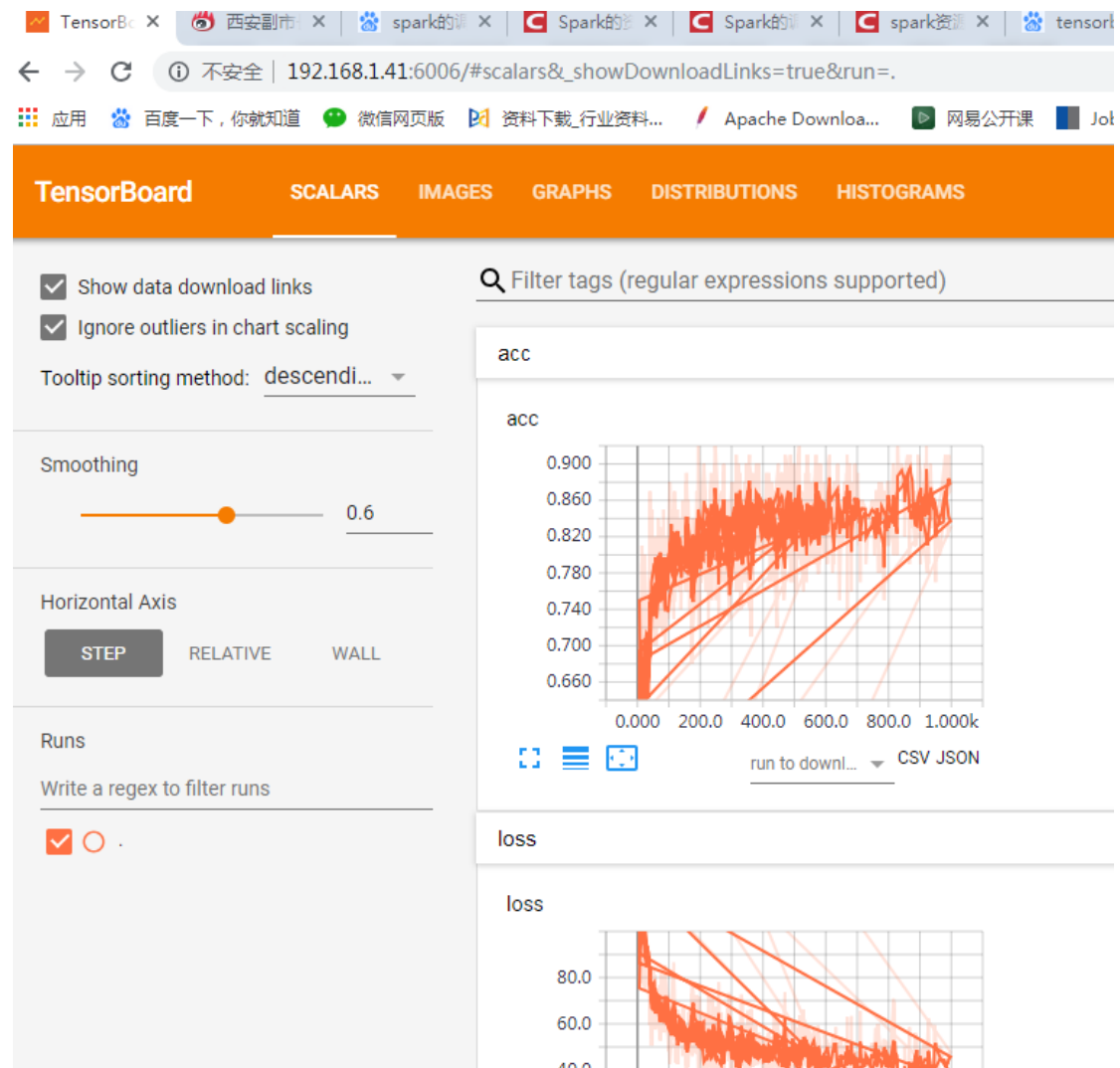
```

可以支持: CIFAR, CRITEO, IMAGENET, WIDE_DEEP 等其它 CASE, 或是利用 TF. LEARN 测更多的应用场景。

Tensorboard-可视化界面

<https://github.com/tensorflow/tensorboard/blob/master/README.md>

```
TensorFlowOnSpark]$tensorboard --logdir ./tensorboard_1/
```



● TF.learn

安装 TFLearn

我使用的 anaconda 安装，不了解 anaconda 的可以看一下我的另一篇文章 [Anaconda 简易使用教程](#)。

首先，我们用 anaconda 创建一个叫 tflearn 的新环境

```
conda create -n tflearn python=3.5
```

然后进入环境

```
source activate tflearn
```

然后安装库和 TFLearn 的依赖项

```
conda install numpy pandas scipy h5py
pip install tensorflow
pip install TFLearn
```

编码

安装好环境之后，我们就可以开始编码了。先引入相关库。

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-
import numpy as np
import tensorflow as tf
import tflearn
import tflearn.datasets.mnist as mnist
```

获取数据

```
# 获取数据，这里直接一个函数就把训练数据、测试数据全部分配好了，就是这么简单
trainX, trainY, testX, testY = mnist.load_data(one_hot=True)
```

定义神经网络

这里，我们定义一个有 784 个输入，有两个隐藏层，输出层节点为 10 的神经网络。

```

def build_model():
    # 重置所有参数和变量
    tf.reset_default_graph()

    # 定义输入层
    net = tflearn.input_data([None, 784])

    # 定义隐藏层
    net = tflearn.fully_connected(net, 200, activation='ReLU')
    net = tflearn.fully_connected(net, 30, activation='ReLU')

    # 输出层
    net = tflearn.fully_connected(net, 10, activation='softmax')
    net = tflearn.regression(net, optimizer='sgd', learning_rate=0.1,
loss='categorical_crossentropy')

    model = tflearn.DNN(net)
    return model

```

构建模型

```

# 构建模型
model = build_model()

```

训练模型

```

# 训练模型
model.fit(trainX, trainY, validation_set=0.1, show_metric=True,
batch_size=100, n_epoch=30)

```

测试模型

```

predictions = np.array(model.predict(testX)).argmax(axis=1) # 预测
值
actual = testY.argmax(axis=1) # 真实值
test_accuracy = np.mean(predictions == actual, axis=0) # 准确度
print("Test accuracy: ", test_accuracy)

```

最后，我们打印出来的精确度：

<https://github.com/freeman93/Demos-of-Deep-Learning->

[Frameworks/blob/master/tflearn/tflearn_mnist_demo.py](#)

● Tf.contrib

TensorFlow 的 contrib 模块已经超越了单个存储库中可以维护和支持的模块。较大的项目最好分开维护，我们将在 TensorFlow 的主代码里添加一些规模较小的扩展。因此，作为发布 TensorFlow 2.0 的一部分，我们将停止分发 tf.contrib。我们将在未来几个月与 contrib 模块的所有者合作制定详细的迁移计划，包括如何在我们的社区页面和文档中宣传您的 TensorFlow 扩展。

对于每个 contrib 模块，我们要么 a) 将项目集成到 TensorFlow 中；b) 将其移至单独的存储库；c) 完全将其移除。这意味着所有的 tf.contrib 都会被弃用，我们将从今天开始停止添加新的 tf.contrib 项目。我们正在寻找目前在 tf.contrib 的许多项目的所有者/维护者，如果您有兴趣，请联系我们。

● TFprof 是 Tensorflow 的性能分析器

● 附一

MNIST 是训练手写数字

CIFAR 是训练图像分类

验证 python 是否有 numpy 包

```
>>> from numpy import *
```

```
>>> eye(4)
```

```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

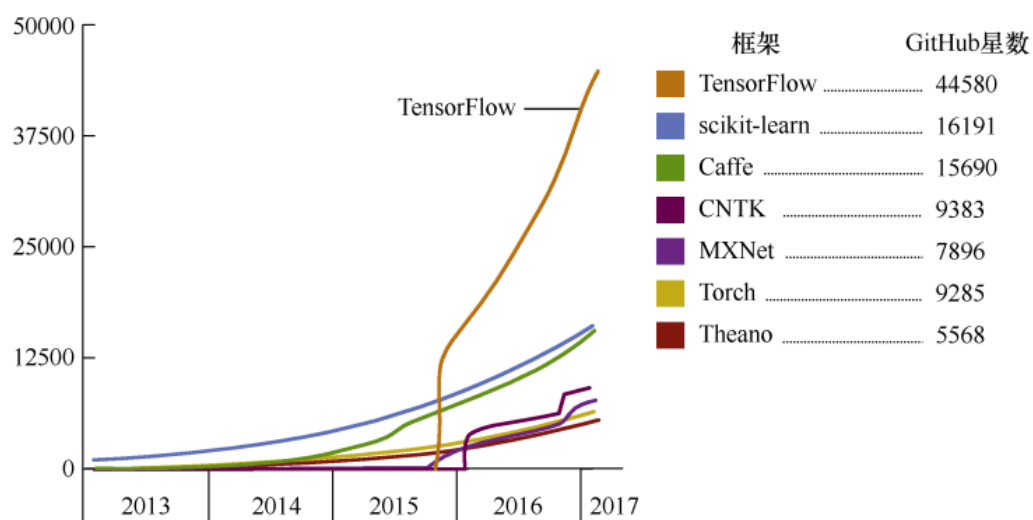
```
Pip install numpy -upgrade
```

Matplotlib 绘图

```
Pip install matplotlib -upgrade
```

jupyter notebook 是 Ipython 的升级版

```
pip install jupyter --upgrade
```



如果支持 GPU，需 CUDA SDK

基于 JAVA 的 TF 安装