MIT REMOTE REARCH ON BIG DATA

RAN TONG

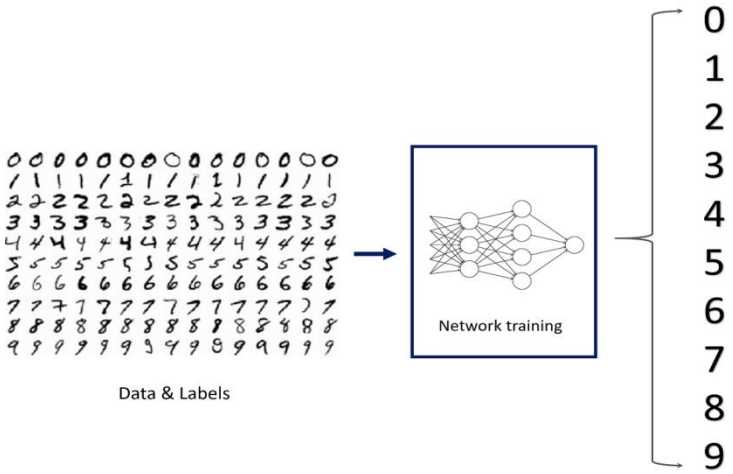
2019.10.17

Overall description

Big data represents the information assets characterized by such a high volume, velocity and variety to require specific technology and analytical methods for its transformation into value.This project aims to get the participants famliar with the cutting edge big data technology such as container,Docker,Cassandra,Spark etc.
This project is consisted of several aims:
To deploy a digit handwriting program based on MNIST via Flask.We can get access to this program using docker. At the exact same time,the result and the time could be recorded into Cassandra.

Mnist

The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized .
Each picture in the MNIST dataset is 28 * 28 pixels in size, and an array of 28 * 28 can be used to represent a picture.



In this program,we could consider using the following code:

The first is a simple but well defined mnist program:

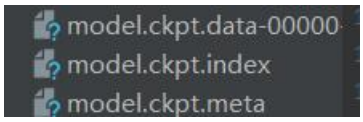
```python
def main(_):
  # Import data
  mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)

  # Create the model
  x = tf.placeholder(tf.float32, [None, 784])
  W = tf.Variable(tf.zeros([784, 10]))
  b = tf.Variable(tf.zeros([10]))
  y = tf.matmul(x, W) + b

  # Define loss and optimizer
  y_ = tf.placeholder(tf.float32, [None, 10])

  # The raw formulation of cross-entropy,
  #
  #   tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(tf.nn.softmax(y)),
  #                                 reduction_indices=[1]))
  #
  # can be numerically unstable.
  #
  # So here we use tf.nn.softmax_cross_entropy_with_logits on the raw
  # outputs of 'y', and then average across the batch.
  cross_entropy = tf.reduce_mean(
      tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y))
  train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

  sess = tf.InteractiveSession()
  tf.global_variables_initializer().run()
```

We could also build a neural network ,

```python
def main(_):
  # Import data
  mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)

  # Create the model
  x = tf.placeholder(tf.float32, [None, 784])

  # Define loss and optimizer
  y_ = tf.placeholder(tf.float32, [None, 10])

  # Build the graph for the deep net
  y_conv, keep_prob = deepnn(x)

  with tf.name_scope('loss'):
    cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=y_,
                                                            logits=y_conv)
  cross_entropy = tf.reduce_mean(cross_entropy)

  with tf.name_scope('adam_optimizer'):
    train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)

  with tf.name_scope('accuracy'):
    correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
    correct_prediction = tf.cast(correct_prediction, tf.float32)
  accuracy = tf.reduce_mean(correct_prediction)

  graph_location = tempfile.mkdtemp()
  print('Saving graph to: %s' % graph_location)
  train_writer = tf.summary.FileWriter(graph_location)
  train_writer.add_graph(tf.get_default_graph())

  with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(20000):
```

In which :
Input x: Input refers to the vector passed into the network, which is equivalent to the variable in the mathematical function.
Output Y: Output refers to the result returned after network processing, which is equivalent to the function value in the data function.
Label: Labels refer to the results we expect the network to return.
For MNIST image recognition, the input is a vector of 784 (28 * 28) size, and the output is a probability vector of 10 size (the position with the highest probability, i.e. the predicted number).
We can choose one of them .(the latter may consume more time)

We need to train our the model we applied with the mnist dataset,and save results :

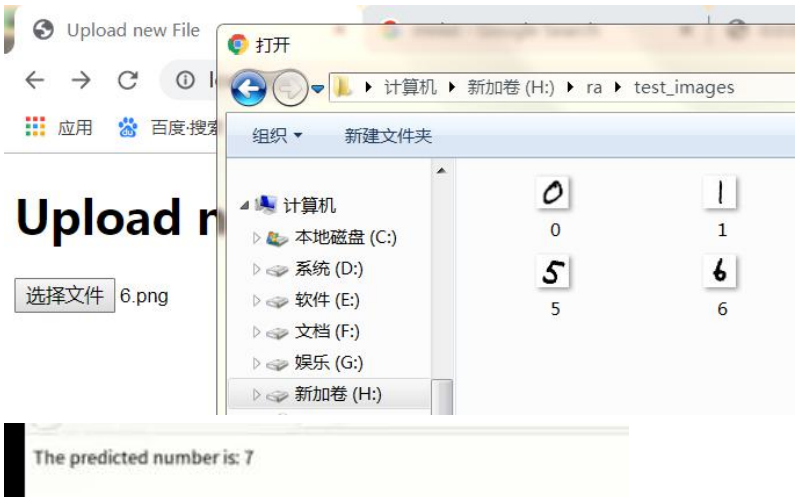


WE save teh model as ckpt.And what we are going to do next is to integrate it with flask.

Flask

Flask is a web application framework written in python.. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.In this project, I use flask to deploy the application.

```
@app.route('/', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        if 'file' not in request.files:
            return ('No file part')
        file = request.files['file']

        if file.filename == '':
            flash('No selected file')
            return redirect(request.url)

        if file and allowed_file(file.filename):
            filename = secure_filename(file.filename)
            file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
            predict = mnist_read.do_predict(filename)
            bd_cassandra.write_cassandra(filename, predict)
            return ('The result is: ' + str(predict) + '\n')
```



We can upload the pictures that we have prepared.The router will save the image and requests ,and the resuilt will return to the browser after calling the model.All of this will be done in local host.

Docker

A quite significant tool I have got to known in this project is docker.**Docker** is mainly a software development platform and a kind of virtualization technology that makes it easy for us to develop and deploy apps inside of neatly packaged virtual containerized environments.To be specific, it is just like running on a virtue machine without those we do not need at all ,that will definitely save a lot of space and calculation time. What is more,Docker's interface is fairly simple. Users can easily create and use containers and put their applications in containers. Containers can also be versioned, copied, shared, and modified, just like ordinary code.

Adavantages:

Flexible: Even the most complex applications can be containerized.

Lightweight: Containers leverage and share the host kernel, making them much more efficient in terms of system resources than virtual machines.

Portable: You can build locally, deploy to the cloud, and run anywhere.

Loosely coupled: Containers are highly self sufficient and encapsulated, allowing you to replace or upgrade one without disrupting others.

Scalable: You can increase and automatically distribute container replicas across a datacenter.

Secure: Containers apply aggressive constraints and isolations to processes without any configuration required on the part of the user.

Images and containers

Fundamentally, a container is nothing but a running process, with some added encapsulation features applied to it in order to keep it isolated from the host and from other containers. One of the most important aspects of container isolation is that each container interacts with its own, private filesystem; this filesystem is provided by a Docker **image**. An image includes everything needed to run an application -- the code or binary, runtimes, dependencies, and any other filesystem objects required.

In this project,we can get access to our app using docker.Before,that we need to have a look at dockerfile,which we are going to need when we want to build a image for our app,It contains the information of our app.

```
FROM cassandra:latest
WORKDIR /app
COPY . /app
RUN mv /etc/apt/sources.list.d/cassandra.list /etc/apt/sources.list.d/cassandra.list.bak
RUN mv /etc/apt/sources.list /etc/apt/sources.list.bak
RUN set -e; \
        apt-get update;\
        apt-get install -y --no-install-recommends \
        python3-pip \
        python3-setuptools;

RUN pip3 install -r requirements.txt
EXPOSE 80
CMD ./services_script.sh
```

We build our image based on the dockerfile.

Cassandra

Cassandra is a free and open-source, distributed, wide column store, NoSQL database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. Cassandra offers robust support for clusters spanning multiple datacenters,with asynchronous masterless replication allowing low latency operations for all clients.

How to do:
In this project, we need to create a docker network for the communication between the two containers: by using
docker network create (name)

```
Administrator@MS-20170809TPNF MINGW64 /c/Program Files/Docker Toolbox
$ docker network create mnist2
bb797d14add014cbf52a9db615d953a0ce74e7d8a001cd7cb9d654f25aa12548
```

And then we need to create the container to hold the application we have made by:
docker build -t [image]:latest

Since we use cassandra driver ,thus we need to define keyspace first:

```
KEYSPACE="mykeyspace_mnist"
```

And then,we need to define teh table:

```
def createKeySpace():
    cluster = Cluster(contact_points=['172.20.0.2'],port=9042)
    session = cluster.connect()

    log.info("Creating keyspace...")
    try:
        session.execute("""
            CREATE KEYSPACE IF NOT EXISTS %s
            WITH replication = { 'class': 'SimpleStrategy', 'replication_factor':
            """ % KEYSPACE)

        log.info("setting keyspace...")
        session.set_keyspace(KEYSPACE)

        log.info("creating table...")
        session.execute("""
            CREATE TABLE mytable (
                time text,
                file_name text,
                number text,
                PRIMARY KEY (time)
            )
            """)
upload()
```

Insert the data:

```
def insertData(time, file, number):
    cluster = Cluster(contact_points=['172.20.0.2'],port=9042)
    session = cluster.connect(KEYSPACE)

    log.info("setting keyspace...")
    session.set_keyspace(KEYSPACE)

    prepared = session.prepare("""
    INSERT INTO mytable (time, file_name, number)
    VALUES (?, ?, ?)
    """)
```

The code work is done here.

How to run this part :
So in this project,we need to pull the cassandra image from the Docker Hub first,using the following code in docker toolbox:

```
Administrator@MS-20170809TPNF MINGW64 /c/Program Files/Docker Toolbox
$ docker pull cassandra
Using default tag: latest
latest: Pulling from library/cassandra
Digest: sha256:0c03f24ba76bbbc33548e2bd22da44ab9589960e8346f84ccd71adff13fc2ec5
Status: Image is up to date for cassandra:latest
docker.io/library/cassandra:latest
```

We need to construct the Docker network bridge in order to link the containers with each other by using:

```
Administrator@MS-20170809TPNF MINGW64 /c/Program Files/Docker Toolbox
$ docker network create mnist2
bb797d14add014cbf52a9db615d953a0ce74e7d8a001cd7cb9d654f25aa12548
```

We also need a container to hold the image of Cassandra we have pulled before.

```
$ docker run --name rdt-cassandra -p 9042:9042 -d cassandra:latest
```

We could check the status by using docker ps:

```
Administrator@MS-20170809TPNF MINGW64 /c/Program Files/Docker Toolbox
$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             ST
ATUS                PORTS                                                NAMES
4c506773f102        cassandra:latest    "docker-entrypoint.s"    4 hours ago         Up
4 hours             7000-7001/tcp, 7199/tcp, 9160/tcp, 0.0.0.0:9042->9042/tcp   rantong-ca
ssandra
```

Then we need to build the application image from the Dockerfile by using:

docker build -t mnist:latest .

```
Administrator@MS-20170809TPNF MINGW64 /c/Program Files/Docker Toolbox
$ docker build -t mnist:latest .
Sending build context to Docker daemon  398.2MB
Step 1/10 : FROM cassandra:latest
 ---> ce5334a7b86c
Step 2/10 : WORKDIR /app
 ---> Running in ae95186c4c00
Removing intermediate container ae95186c4c00
 ---> 21b62e0ca8eb
Step 3/10 : COPY . /app
 ---> 6bc4f0cf1f88
Step 4/10 : RUN mv /etc/apt/sources.list.d/cassandra.list /etc/apt/sources.list.d/cassa
ndra.list.bak
 ---> Running in 007b0672012d
Removing intermediate container 007b0672012d
 ---> 46b0d63d8b87
Step 5/10 : RUN mv /etc/apt/sources.list /etc/apt/sources.list.bak
 ---> Running in d3b9c9050045
Removing intermediate container d3b9c9050045
 ---> 516a27259ffe
Step 6/10 : RUN echo '    deb http://mirrors.tuna.tsinghua.edu.cn/debian/ stretch main
contrib non-free\n    deb http://mirrors.tuna.tsinghua.edu.cn/debian/ stretch-updates m
ain contrib non-free\n    deb http://mirrors.tuna.tsinghua.edu.cn/debian/ stretch-backp
orts main contrib non-free\n    deb http://mirrors.tuna.tsinghua.edu.cn/debian-security
 stretch/updates main contrib non-free' >> /etc/apt/sources.list
```

It goes well first,but i got a problem here:

Whenever it needs to download something large,the process takes so long that each time it returns time out.

```
 ---> 01d50876e7c8
Step 5/9 : RUN pip install Flask
 ---> Using cache
 ---> 2144ed288ed3
Step 6/9 : RUN pip install Pillow
 ---> Using cache
 ---> bbd73ed0da93
Step 7/9 : RUN pip install tensorflow==1.14
 ---> Running in 0c4962871705
Collecting tensorflow==1.14
  Downloading https://files.pythonhosted.org/packages/f4/28/96efba1a516cdacc2e2d6d081f6
9c001d414cc8ca3250e6d59ae657eb2b/tensorflow-1.14.0-cp37-cp37m-manylinux1_x86_64.whl (1
09.3MB)
```

But in the university lab it did work,thus it proved that there was nothing wrong with the code.And it will give you the result like this:

```
  np_resource = np.dtype([("resource", np.ubyte, 1)])
WARNING:tensorflow:From /usr/local/lib/python3.7/site-packages/tensorflow/python/compat/v2_compat.py:61: disable_resourc
e_variables (from tensorflow.python.ops.variable_scope) is deprecated and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
 * Serving Flask app "pictures" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:80/ (Press CTRL+C to quit)
```

Which suggests that we have achieved the goal of getting access to our app using docker.

We can proceed to running the following to run the container if there is nothing wrong.
docker run -p 5000:5000 mnist:latest

And call shell to check whether the information has been stored properly.
docker exec -it mnist_cassandra cqlsh .

Conclusion
So there is still something undone in the final step.I will keep uploading new versions.
In all,after learning for this month,I have learned a lot about the techniques of big data ,which I had barely heard before.It did widen my horizon.
In the following months,I will keep doing research on mnist ,tensorflow,and docker ,only practice could make perfect.And experience is invaluable.
In the end,I want to thank Dr.zhang, who had taught and instructed me in this project.I have learned a lot from you.Thank you.