DOCKER------MNIST PROJECT

Tian Weiran
AUG 16, 2019

Contents

Project Background..... 1

Introduction..... 1

MNIST dataset.....1

Relating knowledge..... 2

Build a neural network..... 2

TensorFlow.....3

Flask.....3

Docker..... 5

Cassandra.....5

Conclusion..... 8

Project Background

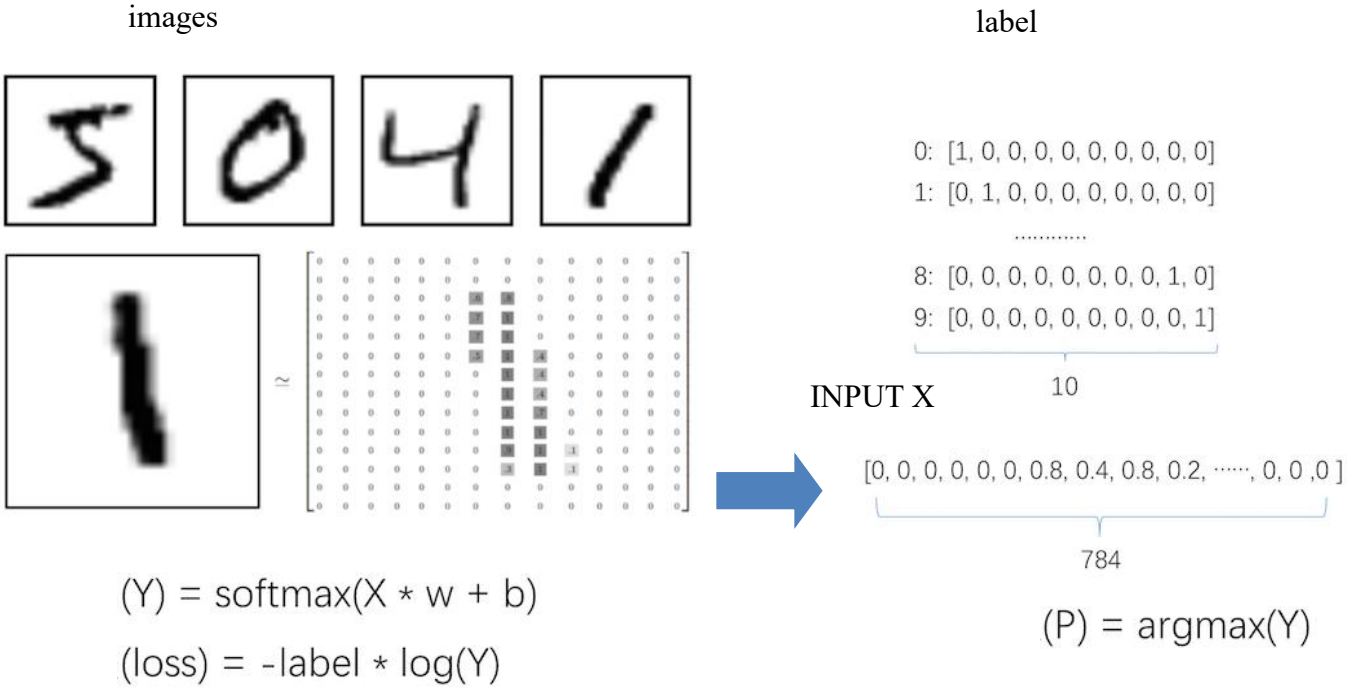
Introduction

This project contains two docker containers: one is for application and the other is for cassandra database.

The user have two different methods to submit a digit image to the certain flask route. The first choice is to use a curl command, and the second is to submit in the browser. The router will then saves the upload image and recognizes it by requesting the MNIST model. After the prediction, the result will be returned in the terminal or in the browser. Meanwhile, the router will saves the image's filename, image's file path, request time and the predict result into the cassandra database container through the docker network bridge.

MNIST dataset

MNIST is an entry-level computer version dataset, which contains of a variety of single handwriting digital pictures. The dataset contains the following four parts:
Train-images-idx3-ubyte.gz: training set-pictures, 60,000pictures
Train-labels-idx1-ubyte.gz: training set-label, 60,000pictures
T10k-images-idx3-ubyte.gz: Test Set-Pictures, 10,000pictures
T10k-labels-idx1-ubyte.gz: Test Set-Label, 10,000pictures
Pictures and labels
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.



Relating knowledge

Build a neural network

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).

LOSS FUNCTION

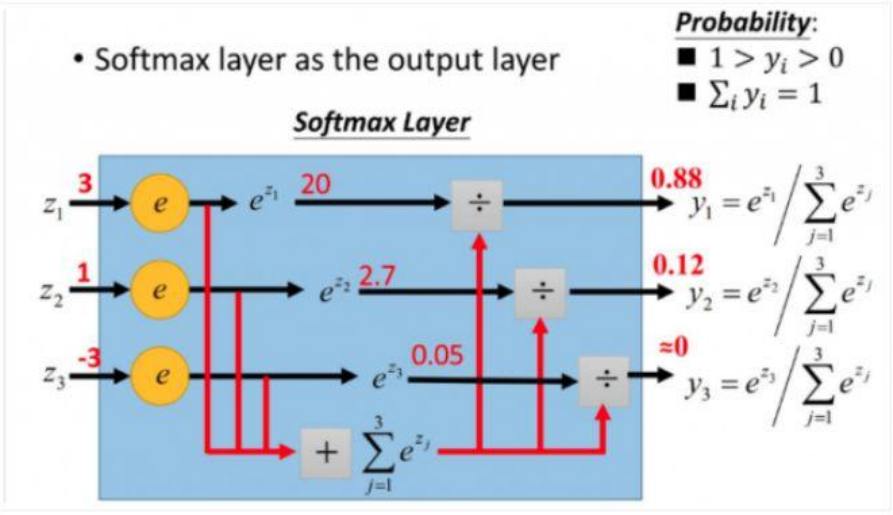
Loss function evaluates the quality of network model. The bigger the value, the worse the model and the smaller the value, the better the model. Because the goal of training a large number of training sets is to minimize the value of the loss function.

For MNIST model, we use the cross entropy to build the loss function. That's because cross-entropy only focuses on the loss of valid bits in unithermal coding. This avoids the change of the invalid bit value (the change of the invalid bit value will not affect the final result), and enlarges the loss of the effective bit by taking logarithm. When the value of the effective bit approaches zero, the cross-entropy approaches positive infinity.

REGRESSION MODEL

If we describe the network as a function, thus, the regression model is the way we hope to fit the function. For MNIST model, we defines the model as Y = X*w + b, By continuously passing in the values of X and label, we can modify w and b to minimize the loss of Y and label. The gradient descent method can be used in this training process. Through gradient descent, find the fastest direction, adjust w and b values, make w* X + b values closer and closer to label.

SOFTMAX ACTIVATION FUNCTION



For example, in this case, Softmax function switches the original output of 3,1, -3 to (0,1) values , and the sum of these values is 1 (satisfied the nature of probability). Then we can understand it as probability. When we finally select the output node, we can choose the most probabilistic (that is, the pair of values). The largest node should be our prediction target.

TensorFlow

TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

In this part, I download the MNIST datasets form the internet and used tensorflow to set up a

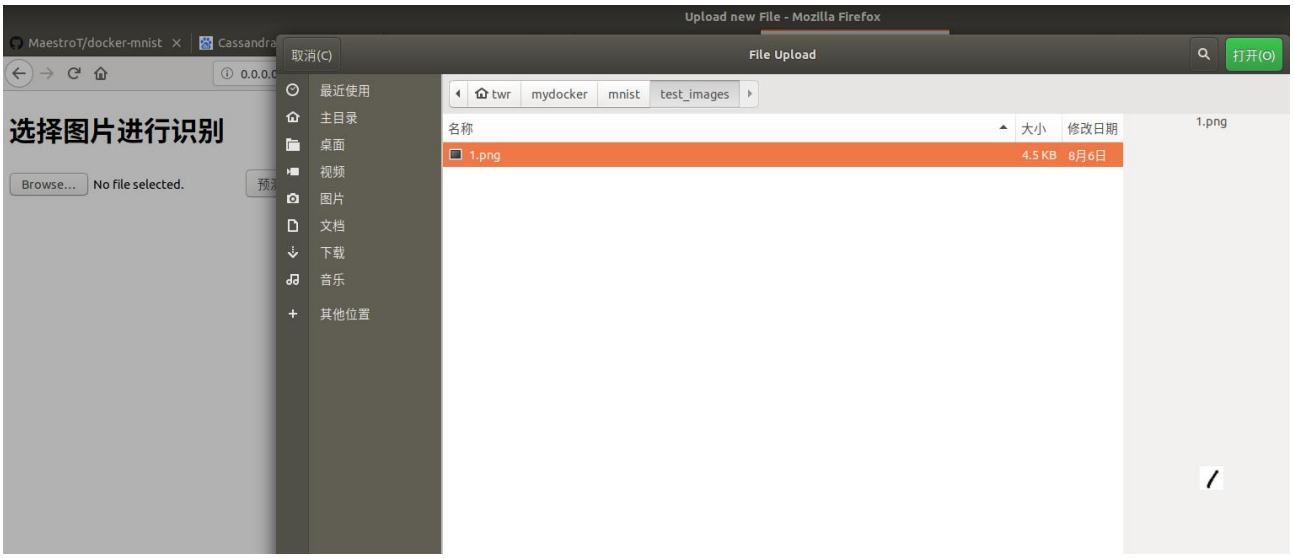
```
step 18400, training accuracy 1
step 18500, training accuracy 1
step 18600, training accuracy 1
step 18700, training accuracy 1
step 18800, training accuracy 1
step 18900, training accuracy 1
step 19000, training accuracy 1
step 19100, training accuracy 1
step 19200, training accuracy 0.98
step 19300, training accuracy 1
step 19400, training accuracy 1
step 19500, training accuracy 1
step 19600, training accuracy 1
step 19700, training accuracy 1
step 19800, training accuracy 1
step 19900, training accuracy 1
2019-07-30 19:13:39.489283: W tensorflow/core/framework/allocator.cc:107] Alloca
tion of 1003520000 exceeds 10% of system memory.
2019-07-30 19:13:39.922848: W tensorflow/core/framework/allocator.cc:107] Alloca
tion of 250880000 exceeds 10% of system memory.
2019-07-30 19:13:40.107397: W tensorflow/core/framework/allocator.cc:107] Alloca
tion of 501760000 exceeds 10% of system memory.
test accuracy 0.9929
```

training, and the training result is 99.29%

Flask

Flask is a web application framework written in python, which is based on Werkzeug WGSI tool kit and the Jinja2 template engine. In this project I use Flask to work with the back-end web application. In order to submit the image in the browser, I wrote a html code:

```python
# Flask Router
@app.route("/")
def index():
    return  '''
    <!doctype html>
    <title>Upload new File</title>
    <body>
    <h1>选择图片进行识别</h1>
    <form action='http://0.0.0.0:8000/mnist' method='post' enctype='multipart/form-data'>
        <input type='file' name='file'>
    <input type='submit' value='预测'>
    </form>'''
```



The router will saves the digit image and requests after the submit. And then, by calling the MNIST model, the result will be returned in the browser.

```python
@app.route("/mnist", methods=["POST"])
def mnist():
    req_time = datetime.datetime.now()#记录请求时间
    if flask.request.method == "POST":
        #if flask.request.files.get("image"):
        upload_file = flask.request.files["file"]
        upload_filename = secure_filename(upload_file.filename)#上传文件文件名的安全获取，文件名不要用中文

        save_filename = parseName(upload_filename,req_time)
        save_filepath = os.path.join(app.root_path, app.config['UPLOAD_FOLDER'], save_filename)#对文件路径进行拼接
        upload_file.save(save_filepath)#将upload_file保存在服务器的文件系统中
        mnist_result = str(predict(save_filepath))
        db.insertData(request.remote_addr, req_time, save_filepath, mnist_result)
        req_time = str(req_time)
        return upload_filename + ' ' + req_time + ' ' + mnist_result
```

Docker

Docker is a a software development platform, from which can put the application and the program in a file. Running the dockerfile generates a virtual container. The program runs in this virtual container as if it were running on a real physical machine. 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.

In this project, I firstly create a docker network for the communication between the two containers:

```
docker network create [bridge-name]
```

And then use the dockerfile to construct the application container by the following command:

```
docker build -t [imagename]:latest .
```

Here is a snapshot of my dockerfile:

```
FROM python:3.7-slim

WORKDIR /flask-test

COPY requirements.txt requirements.txt

RUN python -m venv venv
RUN venv/bin/pip install -r requirements.txt

COPY app.py model.py db.py boot.sh ./
COPY ckpt ./ckpt
ADD ckpt ./ckpt
RUN chmod +x boot.sh

ENV FLASK_APP app.py

EXPOSE 5000
CMD ["venv/bin/python", "app.py"]
```

Cassandra

Cassandra is a NoSQL database, which is highly scalable and can be used to manage large amounts of structured data. It provides high availability without single point of failure.

Cassandra cluster requires strict time synchronization, a little synchronization will occur such and such problems, I have already explained in the strict time synchronization of Cassandra cluster, so time synchronization is the premise of Cassandra cluster.

Cassandra uses the final consistency model, that is to say, the data updated concurrently at the beginning may be inconsistent, but after this inconsistent period of time, the system will achieve the final consistency. Let each client see the same result.

The strength of this final consistency is determined by the consistency model you choose in cassandra. Usually using cassandra, we select the QUORUM level to indicate that half of the replicas receive the request and return the client response, so as to ensure that the inserted data can be queried. However, there is a problem here. With regard to concurrency, suppose the client updates the same record, on what basis does Cassandra determine the order of requests? Only in

time, Cassandra determines the order of requests based on the time when the requests arrive at the server. The datastax client has a request policy concept called LoadBalancingPolicy, which can be specified by Cluster. builder (). withLoadBalancingPolicy (policy). It also has three strategies, namely:

DCAware Round Robin Policy

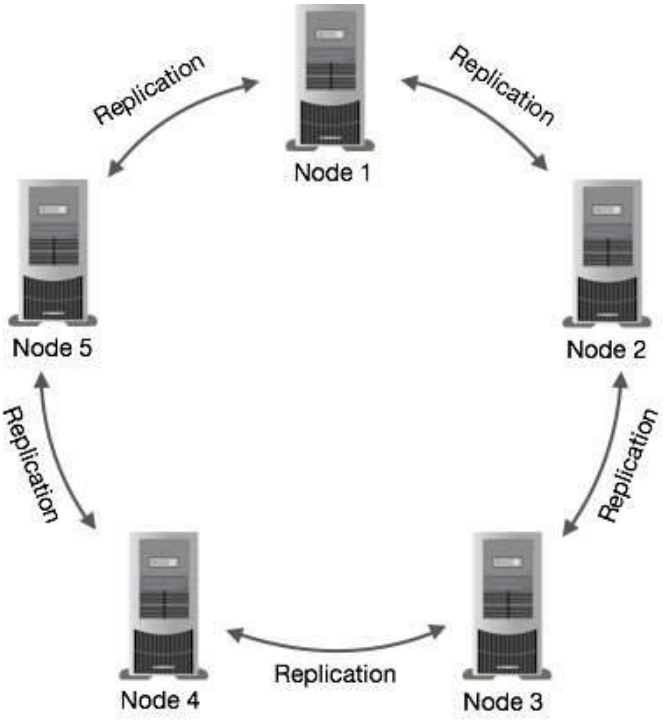
Round Robin Policy

Token Aware Policy

In this project, I use Round Robin Policy, which queries nodes in a circular manner. For a given query, if the host fails, the next query (in circular order) will be attempted until all hosts have attempted it.

```
# 默认本机数据库集群
cluster = Cluster(contact_points=['cassandra'], load_balancing_policy=RoundRobinPolicy(),port=9042,)
# 连接并创建一个会话
session = cluster.connect()
```

In Cassandra, one or more nodes in the cluster act as copies of a given piece of data. If some nodes are detected to respond with expired values, Cassandra will return the nearest value to the client. After returning the latest value, Cassandra performs read fixes in the background to update the invalid value.The following figure shows how Cassandra uses data replication between nodes in the cluster to ensure that there is no single point of failure.



Cassandra can backs up data stored on multiple nodes to ensure reliability and fault tolerance. The replication strategy determines how replicas are placed on nodes.

The total number of replicas in the cluster is called the replication factor. The replication factor of 1 means that each row has only one copy at a node. 2 means that each row has two backups, each of

which is at a different node. All copies are equally important, without distinction between master and deputy. And two replication strategies are available:

SimpleStrategy: Using a single data center. If you want more than one data center, use network topology system strategy.

Network Topology Strategy: It's highly recommended for most deployments because it's easier to extend to multiple data centers when it needs to be extended in the future.

In this project, I choose the SimpleStrategy with 2 republication factors:

```
session.execute("""
    CREATE KEYSPACE %s
    WITH replication = { 'class': 'SimpleStrategy', 'replication_factor': '2' }
    """ % KEYSPACE)
```

And I use cassandra driver to create the keyspace through python.

```
14 KEYSPACE = "mnist"
```

Here is how I define the table:

```
session.execute("""
    CREATE TABLE History (
        IP_Address text,
        access_time timestamp,
        image_path text,
        mnist_result text,
        PRIMARY KEY (IP_Address, access_time)
    )
    """)
```

And then I need to insert data into the table:

```
session.execute("""
    INSERT INTO mnist.History (IP_Address, access_time, image_path, mnist_result)
    VALUES(%s, %s, %s, %s);
    """,
    (ip_addr, access_time, image_path, mnist_result)
)
```

Selecting data from the database:

```
rows = session.execute("SELECT * FROM mnist.History")
log.info("IP_Address text \t access_time \t image_path \t mnist_result")
log.info("---------\t-----\t-----\t-----")
count=0
for row in rows:
    if(count%100==0):
        log.info('\t'.join(row))
    count=count+1
log.info("Total")
log.info("-----")
log.info("rows %d" %(count))
```

To construct the database container, the first step is to pull the cassandra image from the Docker Hub：

```
docker pull Cassandra
```

And the second step is to run the container in the network we have built:

```
docker run --name cassandra --net=[bridge-name] -p 9042:9042 -d cassandra:latest
```

After the prediction, user can check the data in the database by this command:

```
docker exec -it cassandra cqlsh
```

```
twr@twr-911K:~/mydocker/mnist$ docker exec -it cassandra cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.4 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh> use mnist;
cqlsh:mnist> Select * from History;

 ip_address | access_time                      | image_path
                          | mnist_result
------------+----------------------------------+-------------------------------
---------------------------+--------------
 172.20.0.1 | 2019-08-06 10:40:04.876000+0000 | /flask-test/static/uploads/2019-
08-06 10:40:04.876549_1.png |            1
 172.20.0.1 | 2019-08-10 16:08:56.244000+0000 | /flask-test/static/uploads/2019-
08-10 16:08:56.244493_1.png |            1
 172.20.0.1 | 2019-08-13 13:34:00.364000+0000 | /flask-test/static/uploads/2019-
08-13 13:34:00.364792_1.png |            1
 172.20.0.1 | 2019-08-13 13:55:00.111000+0000 | /flask-test/static/uploads/2019-
08-13 13:55:00.111942_1.png |            1
 172.20.0.1 | 2019-08-17 13:46:24.082000+0000 | /flask-test/static/uploads/2019-
08-17 13:46:24.082970_1.png |            1

(5 rows)
cqlsh:mnist>
```

Conclusion

It’s great honored for me to get involved in this project. In this project, I used a lot of tool that I never learned before, like docker, flask, cassandra and tensorflow. In order to finish it, I have looked through lots of information on the Internet. And it’s very proud that I can finished it on my own.

Having finished this project, I think I have took a brief look of machine learning and big data. It makes me more confidence to my further study in this field.

Sincerely thanks Prof Fan Zhang to give me this opportunity to participate in this project and guided me to the big data world.