

Tensorflow.js运行python训练好的模型

一、引言

本文主要描述如何利用js端来调用python训练好的pb模型，其中涉及到npm、webpack等工具的安装、配置和使用以及将pb模型转换为js需要的模型格式。

二、简单的神经网络训练

这个小例子是让神经网络来学习异或运算，代码如下（ceshi.py）：

```
#coding=utf-8#
import tensorflow as tf
import numpy as np
x_data=[[0.0,0.0],[0.0,1.0],[1.0,0.0],[1.0,1.0]] #训练数据
y_data=[[0.0],[1.0],[1.0],[0.0]] #标签
x_test=[[0.0,1.0],[1.0,1.0]] #测试数据
xs=tf.placeholder(tf.float32,[None,2])
ys=tf.placeholder(tf.float32,[None,1]) #定义x和y的占位符作为将要输入神经网络的变量

#构建隐藏层，假设隐藏层有20个神经元
W1=tf.Variable(tf.random_normal([2,10]))
B1=tf.Variable(tf.zeros([1,10])+0.1)
out1=tf.nn.relu(tf.matmul(xs,W1)+B1)
#构建输出层，假设输出层有一个神经元
W2=tf.Variable(tf.random_normal([10,1]))
B2=tf.Variable(tf.zeros([1,1])+0.1)
prediction=tf.add(tf.matmul(out1,W2),B2,name="model")
#计算预测值和真实值之间的误差
loss=tf.reduce_mean(tf.reduce_sum(tf.square(ys-prediction),reduction_indices=[1]))
train_step=tf.train.GradientDescentOptimizer(0.1).minimize(loss)

init=tf.global_variables_initializer() #初始化所有变量
sess=tf.Session()
sess.run(init)

for i in range(40): #训练10次
    sess.run(train_step,feed_dict={xs:x_data,ys:y_data})
    print(sess.run(loss,feed_dict={xs:x_data,ys:y_data})) #打印损失值
re=sess.run(prediction,feed_dict={xs:x_test})
print(re)
for x in re:
    if x[0]>0.5:
        print(1)
    else:
        print(0)
# 保存模型为saved_model
tf.saved_model.simple_save(sess, "./saved_model",inputs={"x": xs }, outputs={"model": prediction, })
```

最后模型保存在saved_model文件夹下。

三、模型转换

saved_model文件夹下保存的是pb格式的模型文件，我们需要将它转换为js需要的格式。

首先安装tensorflowjs: pip install

tensorflowjs，安装成功后会在python目录环境下的scripts文件夹下生成一个tensorflowjs_converter.exe文件

直接在控制台输入：

```
tensorflowjs_converter --input_format=tf_saved_model --output_node_names="model" --saved_model_tags=serve ./saved_model
./web_model
```

转换成功后会生成web_model文件夹，其中包含group1-shard1of1，tensorflowjs_model.pb和weights_manifest.json三个文件，这三个文件就是后面需要的文件

四、在Web中调用

1、html文件形式调用：

本来以为直接以html文件的形式就可以直接调用模型，然后在浏览器中打开html文件就可以得到结果的，html测试代码如下（index.html）：

```
<!doctype html>
<html lang="en">
<head>
  <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs"> </script>
  <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs-converter"></script>
</head>
<body>
  <script>
const MODEL_URL = './tensorflowjs_model.pb'
const WEIGHTS_URL = './weights_manifest.json'
async function fun(){
  const model = await tf.loadFrozenModel(MODEL_URL, WEIGHTS_URL)
  const cs = tf.tensor([[1.0,1.0],[0.0,0.0]])
  cs.print()
  model.predict(cs).print()
}
fun()
  </script>
</body>
</html>
```

但是在浏览器中打开这个html文件，F12之后，在控制台中并没有得到预想的结果，而是会报CORS问题，同时找不到模型文件，只能放弃这个方法。

2、使用webpack

在使用webpack之前，需要利用npm安装tensorflow.js（也可以使用yarn），即需要执行：npm install@tensorflow/tfjs，因此需要先安装npm

（1）安装npm

由于npm是Node.js下的包管理器，因此我们可以直接到Node.js官网下载安装程序，下载完成后，点击.msi安装程序，一路Next（根据自己的需要安装到自定义路径），安装完成后，将nodejs\node_modules\npm\bin添加到系统环境变量，

打开cmd输入：npm -v，打印出版本信息，表示npm安装成功

（2）为npm设置代理

到web_model下面执行：npm install@tensorflow/tfjs，此时可能会出现rollbackFailedOptional verb npm-session的问题，导致安装失败，因此需要为npm进行代理的修改或删除。

如果是公司网络需要设置代理，则：

```
npm config set proxyhttp://192.168.16.232:8080
```

```
npm config set https-proxyhttp://192.168.16.232:8080
```

如果网络不需要代理，则要把npm代理去掉：

```
npm config delete proxy
```

```
npm config delete https-proxy
```

重新执行：npm install@tensorflow/tfjs，应该会安装成功，此时会在web_model文件夹下生成node_modules文件夹，下面是一些依赖包

（3）安装webpack

在web_model目录下，先全局安装webpack: `npm install -g webpack`，然后安装到当前项目目录中: `npm install --save-dev webpack`，安装完成后，node_modules文件夹下会多出一些依赖文件

(4) 新建index.js，内容如下：

```
import * as tf from '@tensorflow/tfjs';
import {loadFrozenModel} from '@tensorflow/tfjs-converter';

const MODEL_URL = 'tensorflowjs_model.pb';
const WEIGHTS_URL = 'weights_manifest.json';

async function predict() {
  try {
    const model = await loadFrozenModel(MODEL_URL, WEIGHTS_URL)
    var cs = tf.tensor([[1.0,1.0],[0.0,0.0]])
    var output = model.execute({x: cs})
    console.log(output.dataSync())
    return output
  } catch (e) {
    console.log(e)
  }
}
```

(5) 新建webpack.config.js文件，内容如下：

```
const path = require('path');

module.exports={
  //入口文件的配置项
  entry:{
    entry: './index.js'
  },
  //出口文件的配置项
  output:{
    path: path.resolve(__dirname, 'dist'),
    filename: 'bundle.js'
  }
}
```

(6) 执行node_modules/.bin/webpack

当上述文件新建完成后，利用webpack打包，当直接在当前目录的cmd中输入webpack时，会提示安装webpack-cli，按照指令输入yes，安装完成。重新执行webpack时，依然出现这个问题，说明webpack命令还是没有成功执行

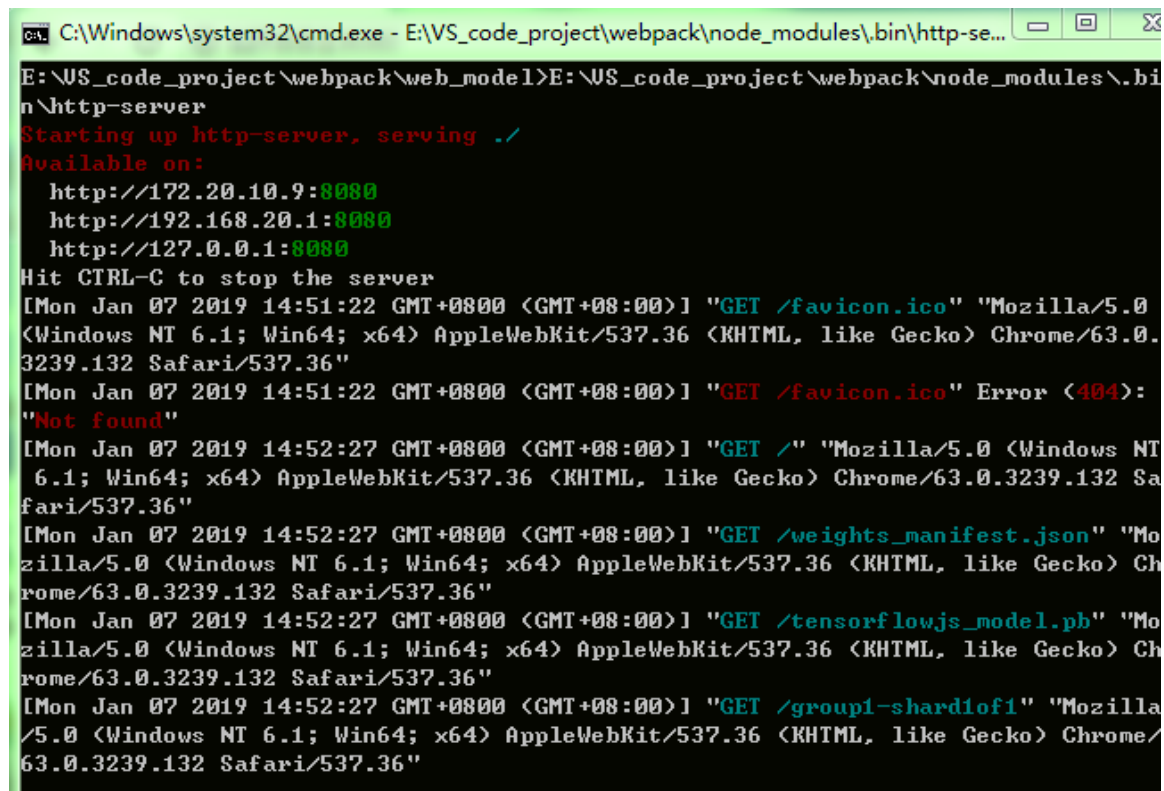
根据网上教程：<https://segmentfault.com/a/1190000013699050>，直接从node_modules/.bin/中执行webpack，成功。

```

E:\VS_code_project\webpack\web_model>E:\VS_code_project\webpack\node_modules\.bin\nwebpack
Hash: e8f6e63e27394b362835
Version: webpack 4.28.3
Time: 16559ms
Built at: 2019-01-07 14:47:53
    Asset      Size  Chunks             Chunk Names
main.js      15 KiB       0 [emitted]  entry
Entrypoint entry 0 = bundle.js
[80] util <ignored> 15 bytes < > [built]
[82] util <ignored> 15 bytes < > [built]
[115] buffer <ignored> 15 bytes < > [built]
[116] crypto <ignored> 15 bytes < > [built]
[163] ./index.js + 2 modules 229 KiB < > [built]
      | ./index.js 440 bytes [built]
      | + 2 hidden modules
      + 159 hidden modules

```

此时将在当前目录下生成dist文件夹，cd进入dist文件夹，cmd中首先npm install http-server，安装完成后依然从node_modules/.bin/中执行http-server。

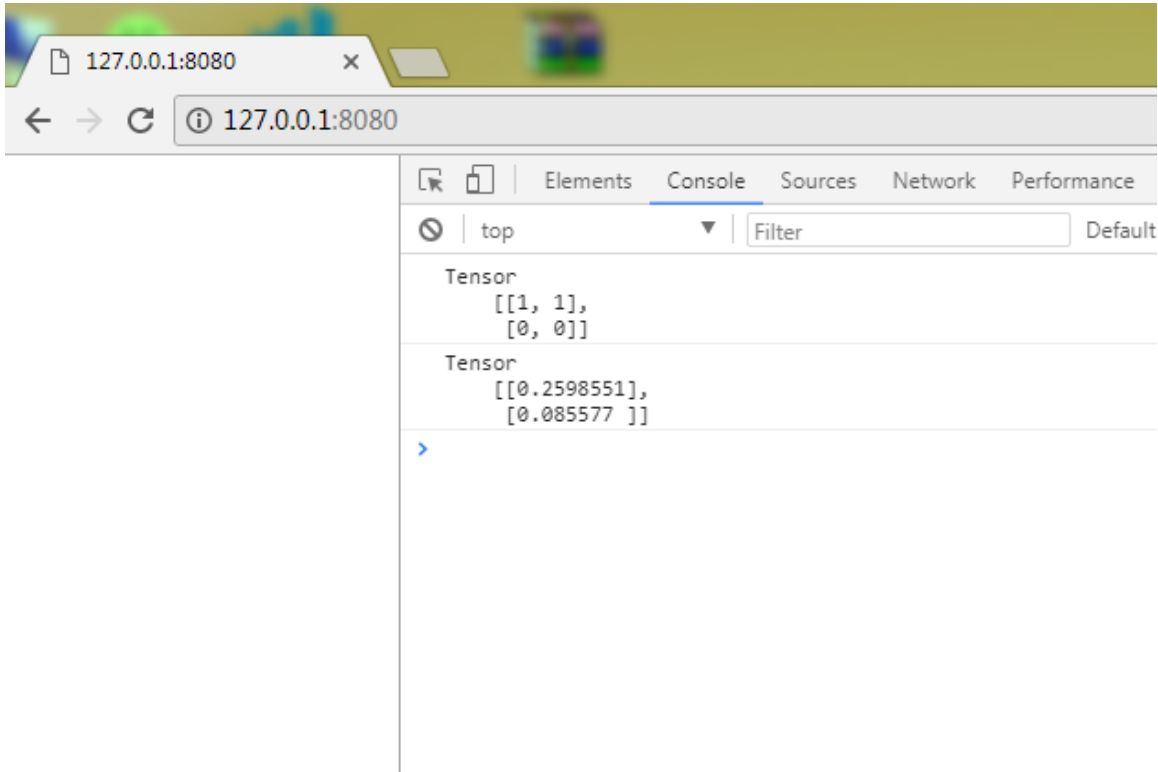


```

C:\Windows\system32\cmd.exe - E:\VS_code_project\webpack\node_modules\.bin\http-se...
E:\VS_code_project\webpack\web_model>E:\VS_code_project\webpack\node_modules\.bin\nhttp-server
Starting up http-server, serving ./
Available on:
  http://172.20.10.9:8080
  http://192.168.20.1:8080
  http://127.0.0.1:8080
Hit CTRL-C to stop the server
[Mon Jan 07 2019 14:51:22 GMT+0800 (GMT+08:00)] "GET /favicon.ico" "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.132 Safari/537.36"
[Mon Jan 07 2019 14:51:22 GMT+0800 (GMT+08:00)] "GET /favicon.ico" Error (404): "Not found"
[Mon Jan 07 2019 14:52:27 GMT+0800 (GMT+08:00)] "GET /" "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.132 Sa
fari/537.36"
[Mon Jan 07 2019 14:52:27 GMT+0800 (GMT+08:00)] "GET /weights_manifest.json" "Mo
zilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Ch
rome/63.0.3239.132 Safari/537.36"
[Mon Jan 07 2019 14:52:27 GMT+0800 (GMT+08:00)] "GET /tensorflowjs_model.pb" "Mo
zilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Ch
rome/63.0.3239.132 Safari/537.36"
[Mon Jan 07 2019 14:52:27 GMT+0800 (GMT+08:00)] "GET /group1-shard1of1" "Mozilla
/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/
63.0.3239.132 Safari/537.36"

```

(7) 浏览器打开输入http://127.0.0.1:8080/，摁下F12，在console中打印出结果：



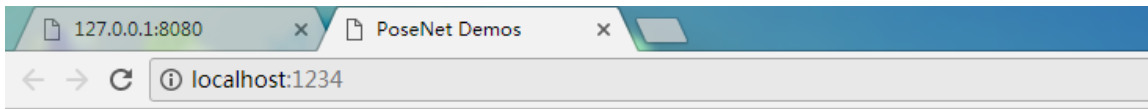
至此，利用Tensorflow.js运行python训练好的模型结束。

ps: 这只是一个简单的小例子，tensorflow.js刚接触，需要深入研究。

补充:

尝试复现了github上利用tensorflow.js构建的posenet demo (<https://github.com/tensorflow/tfjs-models/tree/master/posenet/demos>)

- (1) 将该项目克隆到本地后，需要先安装yarn, 利用yarn执行demo
- (2) 安装好node.js之后，下载yarn的安装程序，提供一个.msi文件，在运行时将引导您在Windows上安装Yarn
- (3) 安装好yarn后，将\Yarn\bin加入系统环境变量
- (4) `cd posenet/demos`
- (5) `yarn` (下载相应的依赖包，生成node_modules)
- (6) `yarn watch` (运行demo，跳转到浏览器)



PoseNet Demos

- [Pose Estimation - Camera feed demo](#)
- [Pose Estimation - Image demo](#)

```
C:\ yarn watch
E:\US_code_project\tfjs-models-master\posenet\demos>yarn watch
yarn run v1.12.3
$ cross-env NODE_ENV=development parcel index.html --no-hmr --open
Server running at http://localhost:1234
✓ Built in 3.80s.
```

(7) 选择任意一个demo，进入应用