WAV音频文件录制

在`stop_record.py`文件中，定义了Recorder类，在`__init__`部分可修改录音的初始设置：

```
def __init__(self, chunk=1024, channels=2, rate=8000):
    self.CHUNK = chunk
    self.FORMAT = pyaudio.paInt16
    self.CHANNELS = channels
    self.RATE = rate
    self._running = True
    self._frames = []
    self.time = 2
```

默认通道数=2，默认频率=8000，默认录音时间=2s

WAV音频文件预处理

路径读取操作

获取训练集和测试集的绝对路径

```
path_film = os.path.abspath('.')    #获取当前绝对路径

path = path_film + "/data/xunlian/"
test_path = path_film + "/data/test_data/"
isnot_test_path = path_film + "/data/isnot_test_path/"
```

生成wav路径的str list

```
def read_wav_path(path):

    map_path, map_relative = [str(path) + str(x) for x in os.listdir(path) if os.path.isfile(str(path) + str(x))], [y for y in os.listdir(path)]
    return map_path, map_relative
```

map_path是wav文件的路径，路径中包含文件名

map_relative是wav文件所在路径，路径不包含文件名

生成MFCC矩阵

读取wav音频文件

```
from scipy.io import wavfile
fs, audio = wav.read(file_name)
```

fs是采样率，audio是声音数据

生成mfcc矩阵

```
from python_speech_features import mfcc,delta
processed_audio = mfcc(audio, samplerate=fs, nfft=2000)
```

samplerate是采样率，

nfft – the FFT size. Default is 512.

Fnc

```
def def_wav_read_mfcc(file_name):
    fs, audio = wav.read(file_name)
    processed_audio = mfcc(audio, samplerate=fs, nfft=2000)
    return processed_audio
```

one-hot编码

Define

one hot编码是将类别变量转换为机器学习算法易于利用的一种形式的过程。

One-Hot编码，又称为一位有效编码，主要是采用N位状态寄存器来对N个状态进行编码，并且在任意时候只有一位有效。

使用one-hot编码，将**离散特征的取值扩展到了欧式空间**，离散特征的某个取值就对应欧式空间的某个点。

Example

男 => 10

女 => 01

祖国特征：["中国"，"美国，"法国"]（这里N=3）：

中国 => 100

美国 => 010

法国 => 001

运动特征：["足球"，"篮球"，"羽毛球"，"乒乓球"]（这里N=4）：

足球 => 1000

篮球 => 0100

羽毛球 => 0010

乒乓球 => 0001

所以，当一个样本为["男","中国","乒乓球"]的时候，完整的特征数字化的结果为：

[1，0，1，0，0，0，0，0，1]

标签二值化

`sklearn.preprocessing.LabelBinarizer(neg_label=0, pos_label=1,sparse_output=False)`

将多类标签转化为二值标签，最终返回的是一个二值数组或稀疏矩阵

参数说明：

`neg_label`：输出消极标签值

`pos_label`：输出积极标签值

`sparse_output`：设置True时，以行压缩格式稀疏矩阵返回，否则返回数组

`classes_`属性：类标签的取值组成数组

```
def def_one_hot(x):
    binarizer = sklearn.preprocessing.LabelBinarizer()
    binarizer.fit(range(max(x)+1))
    y= binarizer.transform(x)
    return y
```

生成原始数据&标签

```
def read_wav_matrix(path):
    map_path, map_relative = read_wav_path(path)
    audio=[]
    labels=[]
    for idx, folder in enumerate(map_path):
        processed_audio_delta = def_wav_read_mfcc(folder)
        audio.append(processed_audio_delta)
        labels.append(int(map_relative[idx].split(".")[0].split("_")[0]))
    x_data,h,l = matrix_make_up(audio)
    x_data = np.array(x_data)
    x_label = np.array(def_one_hot(labels))
    return x_data, x_label, h, l
```

enumerate对象的例子：

```
>>>seasons = ['Spring', 'Summer', 'Fall', 'Winter']
>>> list(enumerate(seasons))
[(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]
```

`idx`为从0开始的index，

`folder`为map_path中的元素

`def_wav_read_mfcc`函数，首先通过自定义的`find_matrix_max_shape`函数获得所有sample中最大的mfcc矩阵，再将所有mfcc矩阵统一为最大矩阵的大小，空余部分填0

程序中已将`find_matrix_max_shape`函数的返回值固定，如更换数据集，需要将此函数返回值修改成变量，找出sample中的最大矩阵大小，再将其返回值重新固定，并参考注释改变`xunlianlo`函数中的每层大小。

`audio`中存储统一大小后的mfcc矩阵

`labels`中存储mfcc矩阵数据的标签

Tensorflow-CNN

数据准备

x是mfcc矩阵，x.shape = (10, 700, 13), 即共10个样本文件，每个样本文件中有700个时间单元，每个时间单元的信号频率倒谱离散为13个level

y是数据类别，y.shape = (10, 4), 即10个样本文件，4-1=3种类型

```
x_train, y_train, h, l = read_wav_matrix(path)
x_test, y_test, h, l = read_wav_matrix(test_path)
```

初始化权值

```
def weight_variable(shape,name):
    initial = tf.truncated_normal(shape,stddev=0.01)#生成一个截断的正态分布
    return tf.Variable(initial,name=name)
```

初始化偏置

```
def bias_variable(shape,name):
    initial = tf.constant(0.01,shape=shape)
    return tf.Variable(initial,name=name)
```

卷积层定义

`strides[0] = strides[3] = 1`. strides[1]代表x方向的步长，strides[2]代表y方向的步长

`padding`: A string from: "SAME", "VALID"

```
def conv2d(x,W):
    return tf.nn.conv2d(x,W,strides=[1,1,1,1],padding='SAME')
```

池化层定义

`ksize [1,x,y,1]`

```
def max_pool_2x2(x):
    return tf.nn.max_pool(x,ksize=[1,2,2,1],strides=
[1,2,2,1],padding='SAME')
```

数据传入placeholder

```
x = tf.placeholder(tf.float32, [None, h, l], name='x-input')
y = tf.placeholder(tf.float32, [None, n], name='y-input')
# 改变x_placeholder的格式转为4D的向量
# [batch, in_height, in_width, in_channels]`
x_image = tf.reshape(x, [-1, h, l, 1], name='x_image') # 700*13*1
```

定义网络层

1. convolutional layer1 + max pooling;
2. convolutional layer2 + max pooling;
3. fully connected layer1 + dropout;
4. fully connected layer2 to prediction.

卷积层部分

初始化第一个卷积层的权值和偏置：

```
W_conv1 = weight_variable([5, 5, 1, 32], name='W_conv1')
# 5*5的采样窗口，输入数据为1层，输出数据为32层
b_conv1 = bias_variable([32], name='b_conv1')
# 每一个卷积核一个偏置值，输出数据为32层
```

定义第一层卷积：

```
# 把x_image和权值向量进行卷积，再加上偏置值，然后应用于relu激活函数

# 定义卷积层
conv2d_1 = conv2d(x_image, W_conv1) + b_conv1
# 选择激活函数
h_conv1 = tf.nn.leaky_relu(conv2d_1) # 700*13*32
# Pooling
h_pool1 = max_pool_2x2(h_conv1)  # 进行max-pooling  350*7*32
```

初始化第二个卷积层的权值和偏置：

```
W_conv1 = weight_variable([5, 5, 32, 64], name='W_conv2')
# 5*5的采样窗口，输入数据为32层，输出数据为64层
b_conv2 = bias_variable([64], name='b_conv2')
# 每一个卷积核一个偏置值，输出数据为64层
```

定义第二层卷积：

```
# 把h_pool1和权值向量进行卷积，再加上偏置值，然后应用于relu激活函数
conv2d_2 = conv2d(h_pool1, W_conv2) + b_conv2
h_conv2 = tf.nn.leaky_relu(conv2d_2) # 350*7*64
h_pool2 = max_pool_2x2(h_conv2)  # 进行max-pooling 175*4*64
```

全连接层部分

通过`tf.reshape()`将`h_pool2`的输出值从一个三维的变为**一维**的数据

```
# [n_sample,175,4,64] ->> [n_sample,175*4*64]
h_pool2_flat = tf.reshape(h_pool2, [-1, 175 * 4 * 64], name='h_pool2_flat')
```

初始化第一个全连接层的权值和偏置：

```
W_fc1 = weight_variable([175 * 4  * 64, 1024], name='W_fc1')
# 上一层有175*4*64个神经元，全连接层有1024个神经元
b_fc1 = bias_variable([1024], name='b_fc1')  # 1024个节点
```

定义第一个全连接层：

```
# 定义全连接层
wx_plus_b1 = tf.matmul(h_pool2_flat, W_fc1) + b_fc1
# 选择激活函数
h_fc1 = tf.nn.leaky_relu(wx_plus_b1)
```

过拟合的dropout处理：

```
# keep_prob用来表示神经元的输出概率
keep_prob = tf.placeholder(tf.float32, name='keep_prob')
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob, name='h_fc1_drop')
```

初始化第二个全连接层的权值和偏置：

```
W_fc2 = weight_variable([1024, n], name='W_fc2')
# 第二个全连接层依然有1024个神经元
b_fc2 = bias_variable([n], name='b_fc2')
```

定义第二个全连接层：

```
# 定义全连接层
wx_plus_b2 = tf.matmul(h_fc1_drop, W_fc2) + b_fc2
# 选择激活函数输出
prediction = tf.nn.leaky_relu(wx_plus_b2)
```

softmax分类器输出分类：

```
p = tf.nn.softmax(wx_plus_b2)
```

优化

利用交叉熵损失函数来定义我们的cost function:

```
cross_entropy =
tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y,
logits=prediction),name='cross_entropy')
```

使用AdamOptimizer进行优化:

```
train_step = tf.train.AdamOptimizer(1e-5).minimize(cross_entropy)
```

求准确率

```
# 结果存放在一个布尔列表中
correct_prediction = tf.equal(tf.argmax(prediction, 1), tf.argmax(y, 1))
# argmax返回一维张量中最大的值所在的位置
# 求准确率
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

训练模型

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer()) # 初始化变量

    for i in range(100001):
        # 训练模型
        sess.run(train_step, feed_dict={x: x_train, y: y_train, keep_prob:
1.0})

        # 计算准确率
        test_acc = sess.run(accuracy, feed_dict={x: x_test, y: y_test,
keep_prob: 1.0})
        train_acc = sess.run(accuracy, feed_dict={x: x_train, y: y_train,
keep_prob: 1.0})

        print("训练第 " + str(i) + " 次，训练集准确率= " + str(train_acc) + " ,
测试集准确率= " + str(test_acc))
```

```
        if test_acc == 1 and train_acc >= 0.95:
            print("准确率完爆了")
            # 保存模型
            saver.save(sess, 'nn/my_net.ckpt')
            break
```

模型应用

数据导入

```
x_test, y_test, h, l = read_wav_matrix(isnot_test_path)
```

迭代网络

```
with tf.Session() as sess:
    # 保存模型使用环境
    saver = tf.train.import_meta_graph("nn/my_net.ckpt.meta")
    saver.restore(sess, 'nn/my_net.ckpt')

    predictions = tf.get_collection('predictions')[0]
    p = tf.get_collection('p')[0]

    graph = tf.get_default_graph()

    input_x = graph.get_operation_by_name('x-input').outputs[0]
    keep_prob = graph.get_operation_by_name('keep_prob').outputs[0]

    for i in range(m):
        result = sess.run(predictions, feed_dict={input_x:
np.array([x_test[i]]),keep_prob:1.0})
        haha = sess.run(p, feed_dict={input_x: np.array([x_test[i]]),
keep_prob: 1.0})
        print("取值置信度"+str(haha))

        print("实际 :"+str(np.argmax(y_test[i]))+" ,预测:
"+str(np.argmax(result))+" ,预测可靠度: "+str(np.max(haha)))

```

绘图

在`test.py`文件中，可以对音频文件可视化，并画出音频文件的MFCC矩阵。

画音频图：

```
def plot_wav(fs, audio):
    frames = audio.shape
    time = np.arange(0, frames[0]) * (1.0/fs)
    plt.plot(time, audio)
```

画MFCC矩阵图：

```
plt.matshow(processed_audio.T)
```