周报

这周老师布置了使用LSTM拟合音色的任务，找出使用python办成的相关代码和方法，但是找出的大多内容都是如何拟合函数，在github上寻找timbre有寻找到相关的js文件，可能是查找的资源和方法不对，也希望其他同学可以分享一下自己的查找方法，所以这周了解了python的一些编程功能，并尝试了自己训练一个数据，使用python搭建一个输入层，十个隐藏层，一个输出层的神经网络，用来训练一条y=x^2-0.5的曲线，下周开始训练magenta的模型

一、了解Python的一些基础功能和numpy ,Matplotlib

1.for语句

for i in range(1,10):

输出1到9,range(1,10,2)步长是2

Print(i)

1. if语句

若x<y，什么都不输出（<=,>=,==,!=）

x=4，y=2，z=3

if x>y:

print('x is less than y,y is less than z')

1. if : else:
2. If :

elif (else if) ：

else:(表示以上情况都不包含的时候成立)

x=1

if x>1:

print('x>1')

elif x<1:

print('x<1')

else:

print('x =1')

1. 定义函数，格式：def function(a,b):function

Def function():

print('This is a function')

a=1+2

print(a)

function():

1. 定义函数def变量

def function(a,b):

print('This is a function')

c=a\*b

print('c is',c)

function(a=1,b=2)

1. def默认函数和参数

Def function(price,length,heigth,color='red',brand='carmy',is\_second\_hand='true'):

print('price:',price,

'color:',color,

'brand:',brand,

'is\_second\_hand:',is\_second\_hand,

'length:',length,

'heigth:',heigth)

function(1000,50,50)

def function():

a=10

print(a)

return a+100

print(function())

1. 全局变量（通常以大写方式）和局部变量

输出全局变量后不能输出局部变量

APPLE**=**100  
**def function**()**:** a**=**10  
print(APPLE)  
print(a)

不能在方程中定义全局变量。要定义的话加global

APPLE**=**100  
a**=None  
def function**()**:  
 global** a  
 a**=**20  
 **return** a**+**20  
print(APPLE)  
print(**'a past='**,a)  
print(function())运行该函数  
print(**'z now='**,a)

1. 读写文件

（1）写文件

text**='apple\npeach\n'**print(text)

text**='apple\npeach\nbanana'**my\_file**=**open(**'my file.txt'**,**'w'**)  
my\_file.write(text)  
my\_file.close()

1. 再已写的后边追加

append\_text**='\nwatermelon'**my\_file**=**open(**'my file.txt'**,**'a'**)  
my\_file.write(append\_text)  
my\_file.close()

（3）读文件

file**=**open(**'my file.txt'**,**'r'**)

Content=file.read()  
content**=**file.readline()逐行读取  
second\_read\_time**=**file.readline()读取第二行

Content=file.readlines()读取全部内容  
print(content,second\_read\_time)

1. class类

**class Calculator:** name**='Good Calculator'** price**=**18（固有属性）

**def init**(self,*name*,*price*,*hight*,*width*,*weight*)**:** self.name**=***name* self.price**=***price* self.h**=***hight* self.wi**=***width* self.we**=***weight（自定义属性）*  
 **def add**(self,*x*,*y*)**:** print(self.name)  
 reslut**=***x***+***y* print(reslut)  
 **def minus**(self,*x*,*y*)**:** reslut**=***x***-***y* print(reslut)  
 **def times**(self,*x*,*y*)**:** print(*x***\****y*)  
 **def divide**(self,*x*,*y*)**:** print(*x***/***y*)  
calcul**=**Calculator()  
calcul.name  
calcul.price  
calcul.add(10,11)

1. Input

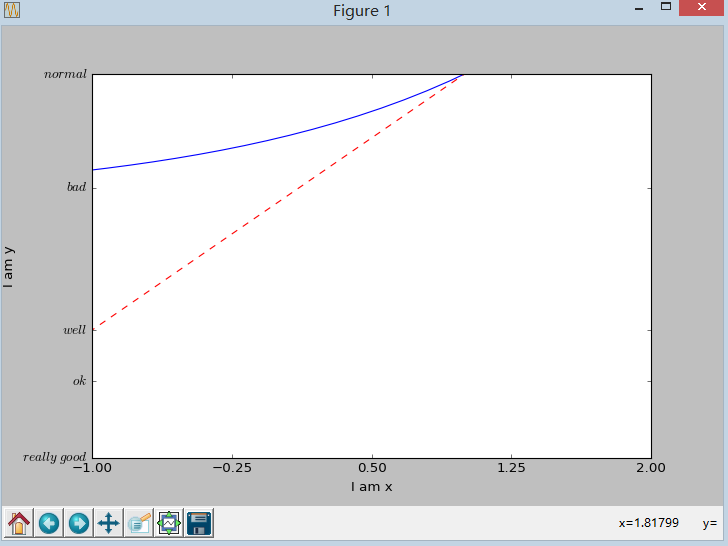
a\_input**=**input(**'Please input a number:'**)*#返回的是一个字符串int(input('Please input a number'))*print(**'The input number is:'**,a\_input)  
**if** a\_input**=='1':** print(**'your are good'**)  
**elif** a\_input**=='2':***#str(2)* print(**'your are bad'**)  
**else:** print(**'good luck'**)

1. 元组和列表
2. a\_tuple**=**(1,2,3,4,5,6)  
   a\_list**=**[7,8,9,10]*#第0个位置是7***for** content **in** a\_tuple**:** print(content)  
   **for** index **in** range(len(a\_list))**:***#计算出a\_List的长度，输出range(5)数字* print(**'index='**,index,**'number in index'**,a\_list[index])
3. 列表

a**=**[1,2,3,4,5,6]  
a.insert(1,0)*#前面是位置，后边是插入的数字*a.remove(2)*#移出去第一次出现的2*print(a)  
print(a[1])*#打印出第一位*print(a[**-**1])*#索引出最后一位*print(a[0**:**3])*#打印出第0位到第三位：表示什么到什么*print(a[5**:**])*#打印出5到以后*print(a[**-**5**:-**3])  
print(a.index(4))*#打印出4在第几个位置*print(a.count(4))*#初现的次数*a.sort()  
print(a)*#从小到大排列，覆盖原来的列表*a.sort(reverse**=True**)  
print(a)*#从大到小*

16.使用Matplotlib（类似于matlab）画一条直线

**import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
x**=**np.linspace(**-**3,3,50)  
y1**=**2**\***x**+**1  
y2**=**2**\*\***x**+**1  
plt.figure()*#第一张图的序号  
#plt.figure(num=2,figsize=(8,5))#第二张图的序号，长高是8和5*plt.plot(x,y2)  
plt.plot(x,y1,color**='red'**,linewidth**=**1.0,linestyle**='--'**)  
plt.xlim((**-**1,2))*#x取值范围*plt.ylim((**-**2,3))*#y取值范围*plt.xlabel(**'I am x'**)  
plt.ylabel(**'I am y'**)*#坐标轴取值名称*new\_ticks**=**np.linspace(**-**1,2,5)  
print(new\_ticks)  
plt.xticks(new\_ticks)*#重新更改坐标轴取值范围*plt.yticks([**-**3,**-**1.8,**-**1,1.22,3],  
 [**r'$really\ good$'**,**r'$ok$'**,**r'$well$'**,**r'$bad$'**,**r'$normal$'**] )  
*#对应坐标点标不同的名字并改变字体大小，机器可读的数字字体*plt.show()



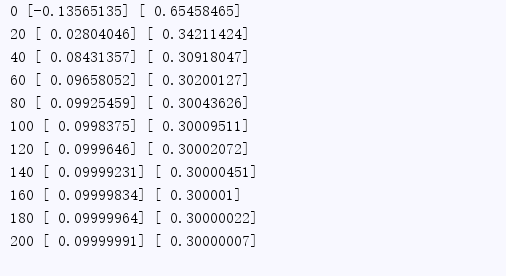
1. Numpy关于矩阵基础

**import** numpy **as** np  
  
a **=** np.array([2, 23, 4], dtype **=** np.int) *# 创建矩阵,定义它的属性，小数还是整数*print(a.dtype)  
b**=**np.array([[2,23,4],  
 [2,23,4]])*#二维或多维矩阵*print(b)  
c**=**np.zeros((3,4))*#定义3行4裂权威0的矩阵*print(c)  
d**=**np.ones((3,4))*#定义3行4裂权威0的矩阵*print(d)

二、使用python训练一些数据

1.训练模型，使weights越来越接近0.1，biases越来越接近0.3

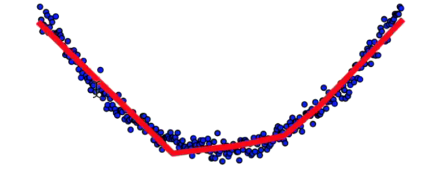
**import** tensorflow **as** tf  
**import** numpy **as** np  
*#创建一些数据，随机生成100个数，类型为32位小数*x\_data**=**np.random.rand(100).astype(np.float32)  
y\_data**=**x\_data**\***0.1**+**0.3  
*#开始创建tensorflow结构*Weights**=**tf.Variable(tf.random\_uniform([1],**-**1.0,1.0))*#随机生成-1到1的一维权重参数变量*biases**=**tf.Variable(tf.zeros([1]))*#定义初始值为0的偏置，之后不断学习提升使其接近于0.1或0.3*y**=**Weights**\***x\_data**+**biases*#要训练的直线*loss**=**tf.reduce\_mean(tf.square(y**-**y\_data))*#计算预测的Y与实际的y的差别*optimizer**=**tf.train.GradientDescentOptimizer(0.5)*#建立优化器减少神经网络的误差，提高准确性，0.5为学习效率，一般小于1*train**=**optimizer.minimize(loss)  
init**=**tf.initialize\_all\_variables()*#建立了Weights，biases但仍未初始化，因此要初始化*sess**=**tf.Session()  
sess.run(init)*#相当于指针，激活该神经网络，控制要运行的结构，激活init***for** step **in** range(201)**:***#开始训练神经网络200次* sess.run(train)  
 **if** step**%**20**==**0**:***#每隔20步打印出数据结果* print(step,sess.run(Weights),sess.run(biases))



可以看到结果，一个数越来越接近0.1，另一个越来越接近0.3

2.搭建一个神经网络并将其可视化

**import** tensorflow **as** tf  
**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
*#如何添加一个神经层***def add\_layer**(*inputs*,*in\_size*,*out\_size*,*activation\_function***=None**)**:***#添加神经层，activation\_function为激励函数，默认为线性函数* Weights**=**tf.Variable(tf.random\_normal([*in\_size*,*out\_size*]))*#定义权重，随机生成in\_size,out\_size的矩阵* biases**=**tf.Variable(tf.zeros([1,*out\_size*])**+**0.1)*#权重是一个列表，只有一行，初始值推荐不为0* Wx\_plus\_b**=**tf.matmul(*inputs*,Weights)**+**biases  
 *#Wx\_plus\_b为计算方式，矩阵相乘inputs\*Weights+biases得到预测的值存储在该变量中* **if** *activation\_function* **is None:***#激活神经网络* outputs**=**Wx\_plus\_b*#None为线性关系，只需保持输出的现状* **else:** outputs**=***activation\_function* (Wx\_plus\_b)  
 **return** outputs  
*#搭建神经网络并将其可视化*x\_data**=**np.linspace(**-**1,1,300)[**:**,np.newaxis]*#x有300行，每一行有一个维度*noise**=**np.random.normal(0,0.05,x\_data.shape)  
*#添加一个噪点，说明并未按照函数线画出，可能周围还有其他点,方差为0.5，x\_data.shape*y\_data**=**np.square(x\_data)**-**0.5**+**noise*#y的值  
#输入层（多少个data有多少个神经元，该函数有一个x\_data），隐藏层（假设有10个神经元），输出层(只有一个y\_data）# )  
#定义隐藏层和输出层*xs**=**tf.placeholder(tf.float32,[**None**,1])  
ys**=**tf.placeholder(tf.float32,[**None**,1])  
l1**=**add\_layer(xs,1,10,activation\_function**=**tf.nn.relu)*#定义隐藏层  
#add\_layer(inputs,in\_size,out\_size,activation\_function=None)，设置函数的参数*prediction**=**add\_layer(l1,10,1,activation\_function**=None**)*#定义输出层  
#定义隐藏层和输出层  
#开始预测*loss**=**tf.reduce\_mean(tf.reduce\_sum(tf.square(ys**-**prediction),  
 reduction\_indices**=**[1]))*#预测之前计算预测值与真实值的差别，tf.reduce\_sum对每个值求和再进行求平均值*train\_step**=**tf.train.GradientDescentOptimizer(0.1).minimize(loss)  
*#优化器，以0.1的进度使训练数据减少误差，提高准确性*init**=**tf.initialize\_all\_variables()*#初始化数据，即激活数据*sess**=**tf.Session()  
sess.run(init)*#激活该神经网络*fig**=**plt.figure()  
ax**=**fig.add\_subplot(1,1,1)  
ax.scatter(x\_data,y\_data)  
plt.ion()  
plt.show()  
**for** i **in** range(1000)**:***#开始训练神经网络* sess.run(train\_step,feed\_dict**=**{xs**:**x\_data,ys**:**y\_data})  
 **if** i**%**50**==**0**:** *#print(sess.run(loss,feed\_dict={xs:x\_data,ys:y\_data}))* **try:** ax.lines.remove(lines[0])  
 **except** Exception**:  
 pass** prediction\_value**=**sess.run(prediction,feed\_dict**=**{xs**:**x\_data})  
 lines**=**ax.plot(x\_data,prediction\_value,**'r-'**,lw**=**5)*#输出预测值* ax.lines.remove(lines[0])*#去除掉上一条曲线* plt.pause(0.1)



训练的过程中曲线不断发生变化，最后逐渐误差逐渐减小，接近真实的曲线