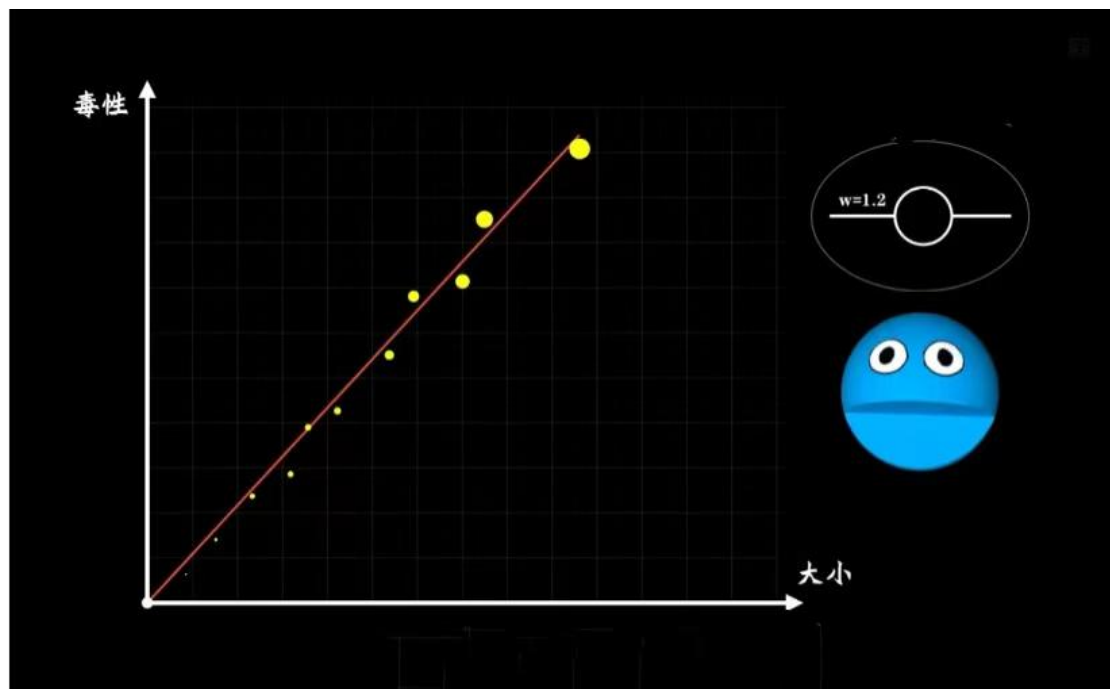
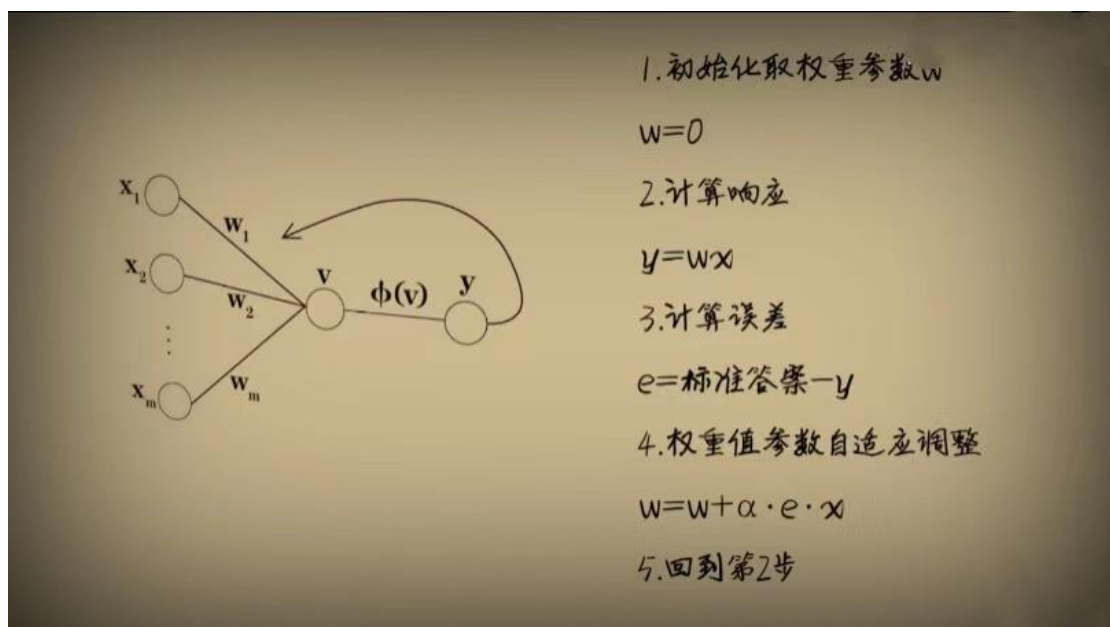


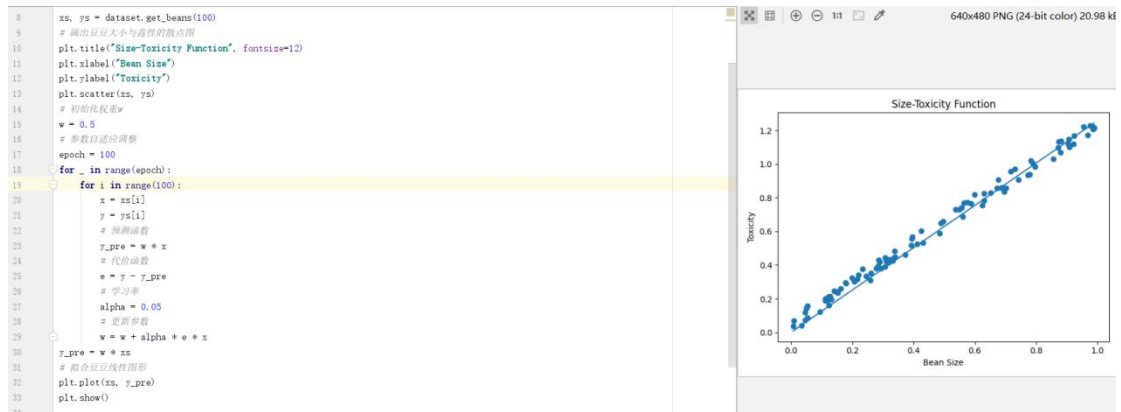
1. Rosenblatt 感知器模型预测豆豆毒性与豆子大小的关系



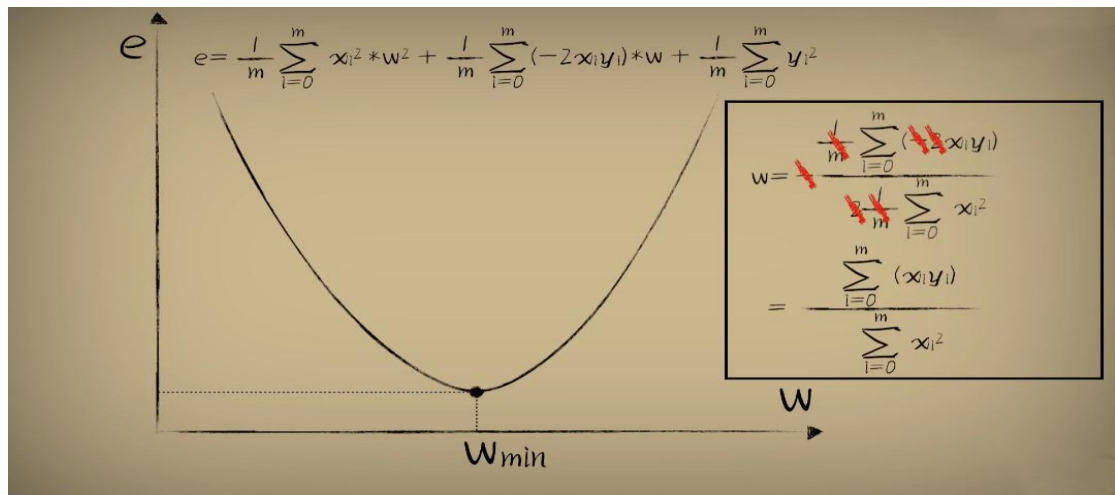
Rosenblatt 感知器计算流程



代码复现

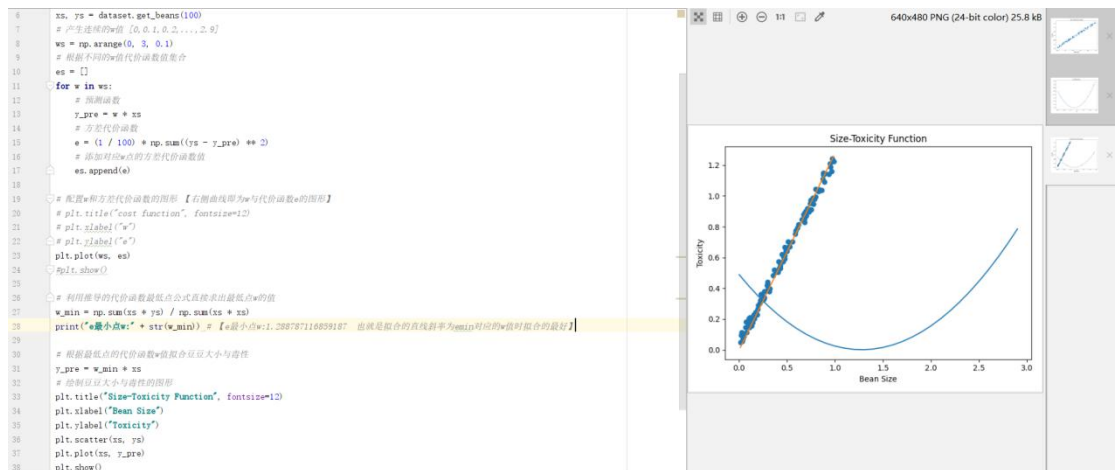


2. 根据方差代价函数 e 与 w 更新权重拟合豆豆大小与毒性的关系



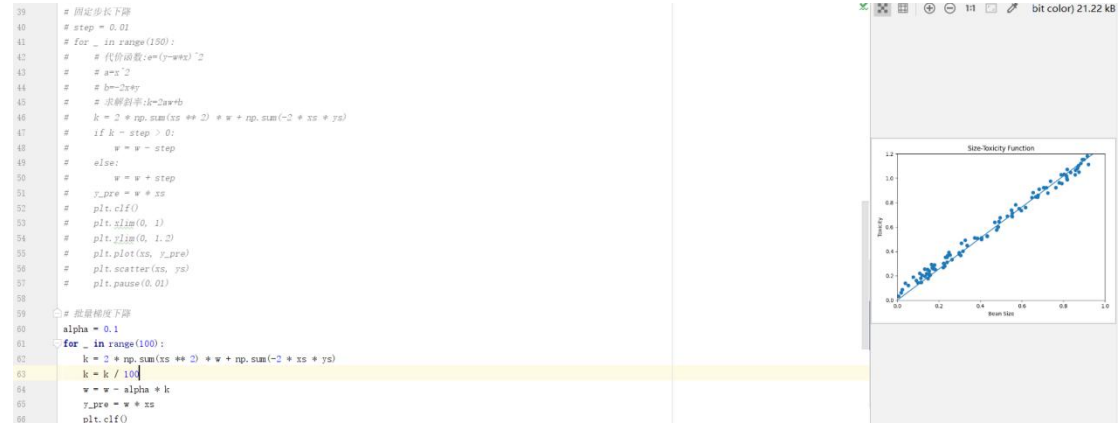
一步到位计算最佳参数 w 意味着巨大的计算量和存储量。输入特征过多、样本数量过大的时候非常消耗计算资源，比如有 100 万数据集,每个数据集 1000 个维度,就得一次性计算 $100 \text{ 万} * 1000 = 10 \text{ 亿次}$,比较考验计算机性能。

代码复现



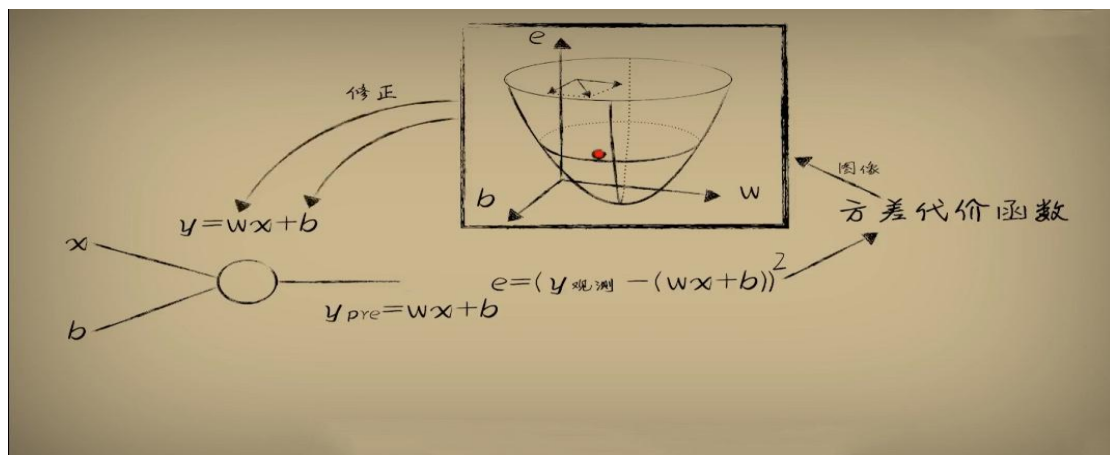
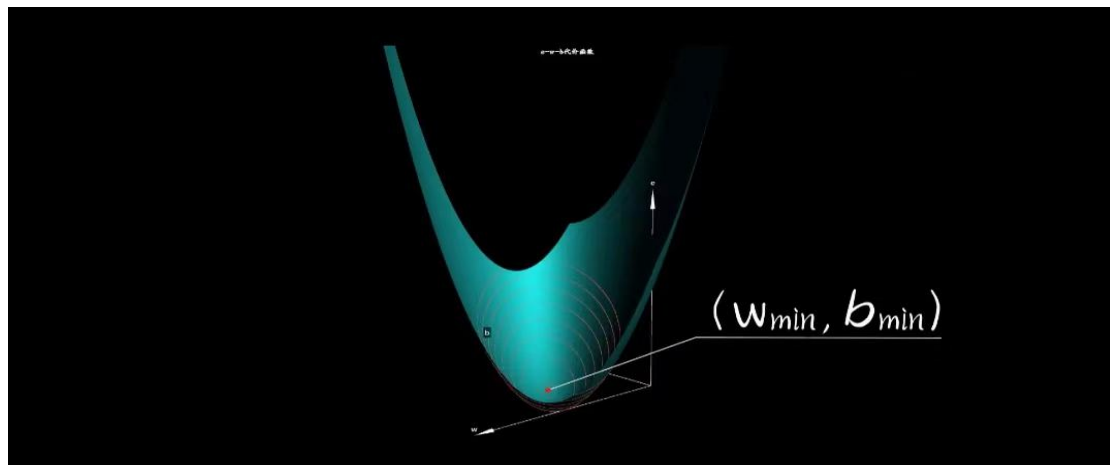
3. 用随机批量梯度下降去拟合豆豆毒性和大小的关系

代码复现



4. 加入截距 b 后根据代价函数梯度下降反向传播 w 与 b 的值

加入截距 b 后,需要在两个维度进行梯度下降

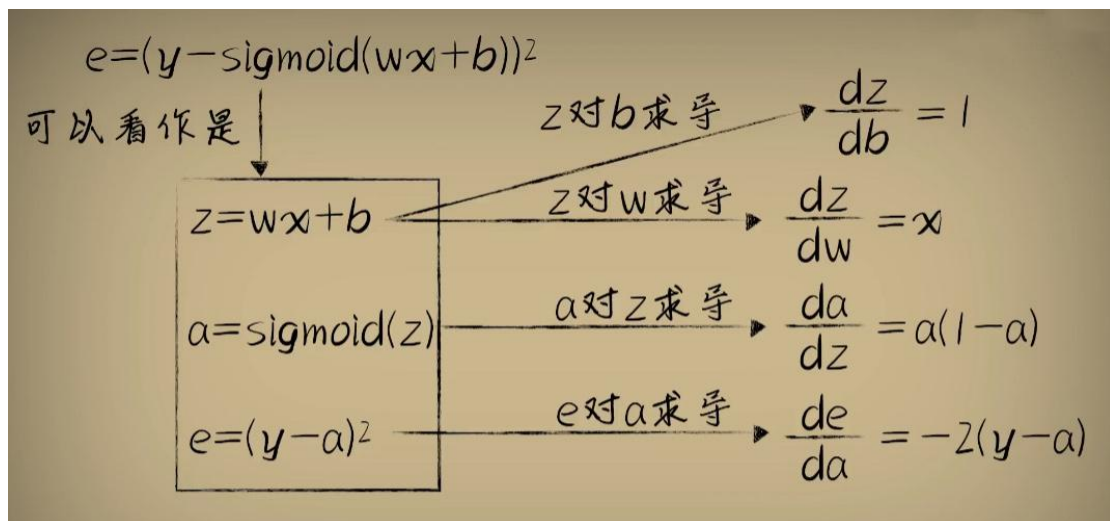


代码复现



5. 当豆豆毒性根据大小分成有毒无毒之后,引入 sigmoid 函数拟合

加入激活函数处理非线性关系模型



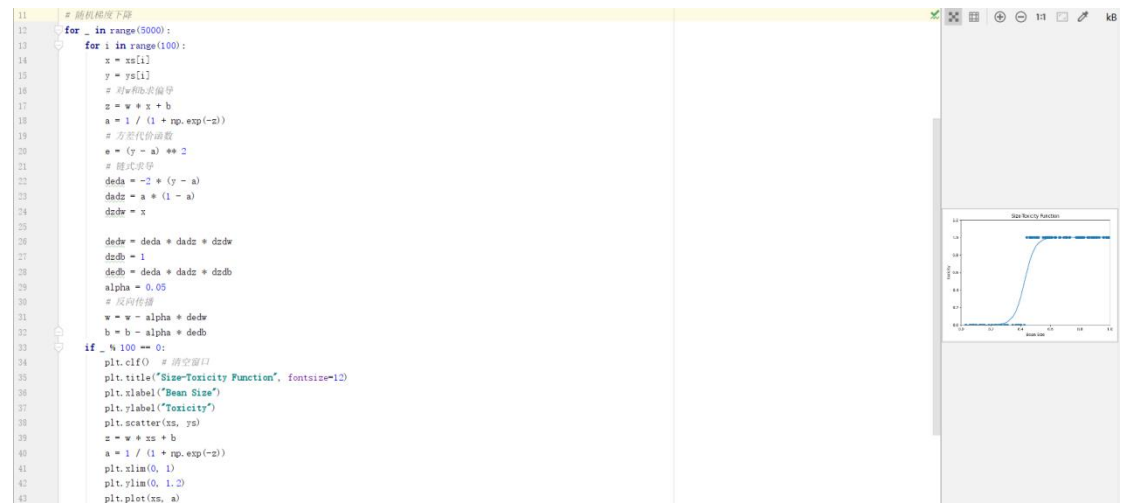
链式求导从而进行梯度下降

$$\frac{de}{dw} = \frac{de}{da} \cdot \frac{da}{dz} \cdot \frac{dz}{dw} = -2(y - a)a(1 - a)x$$

$$\frac{de}{db} = \frac{de}{da} \cdot \frac{da}{dz} \cdot \frac{dz}{db} = -2(y - a)a(1 - a)1$$

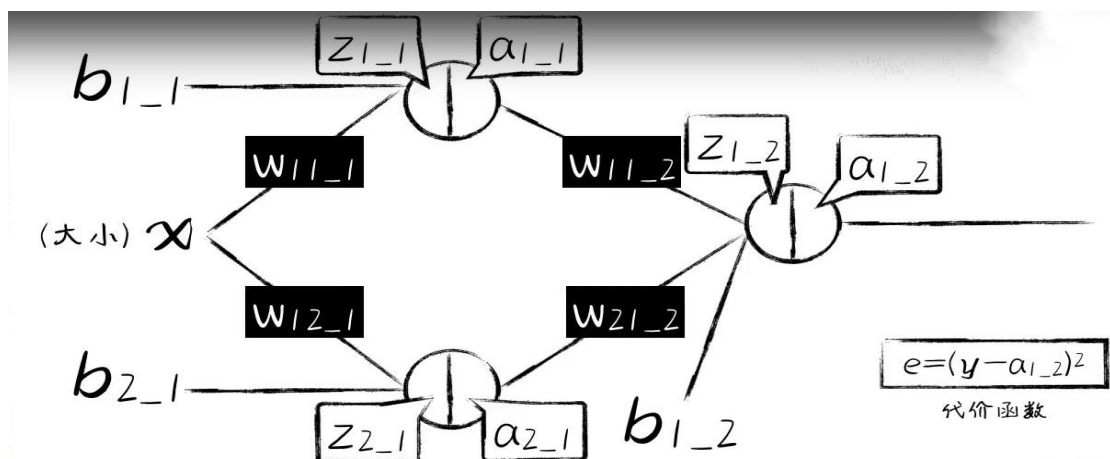
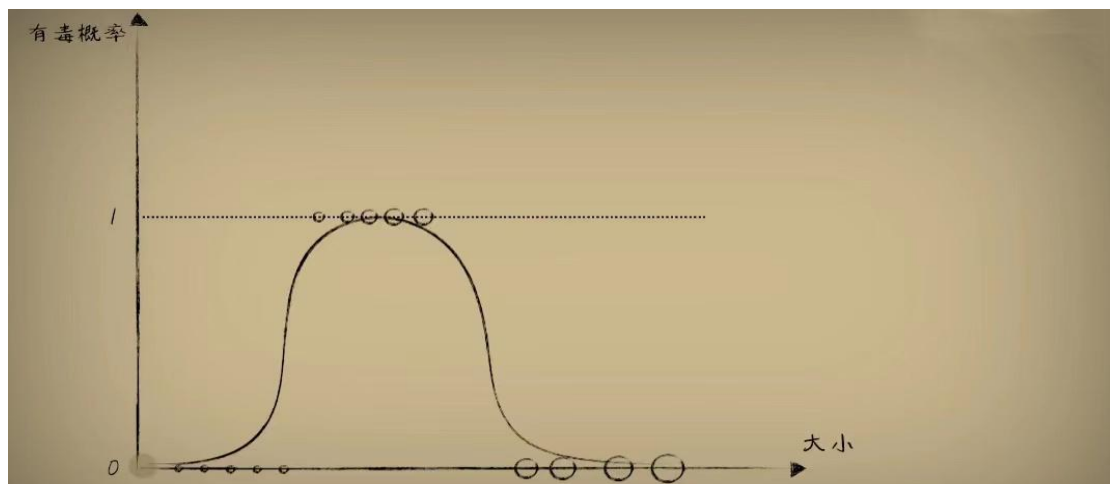
$a = \text{sigmoid}(wx + b)$

代码复现

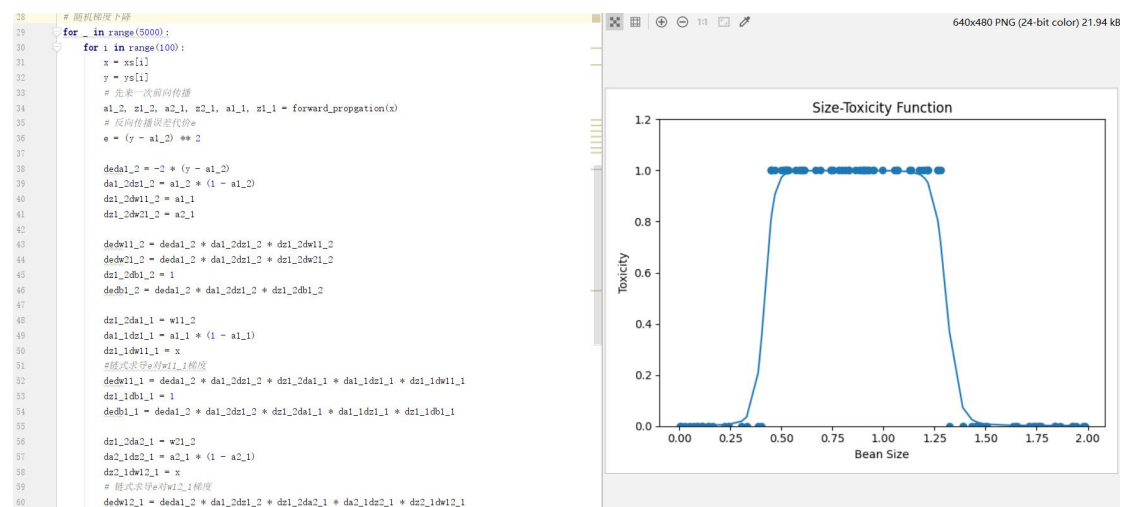


6. 加入隐藏层拟合豆豆大小和有毒概率的关系

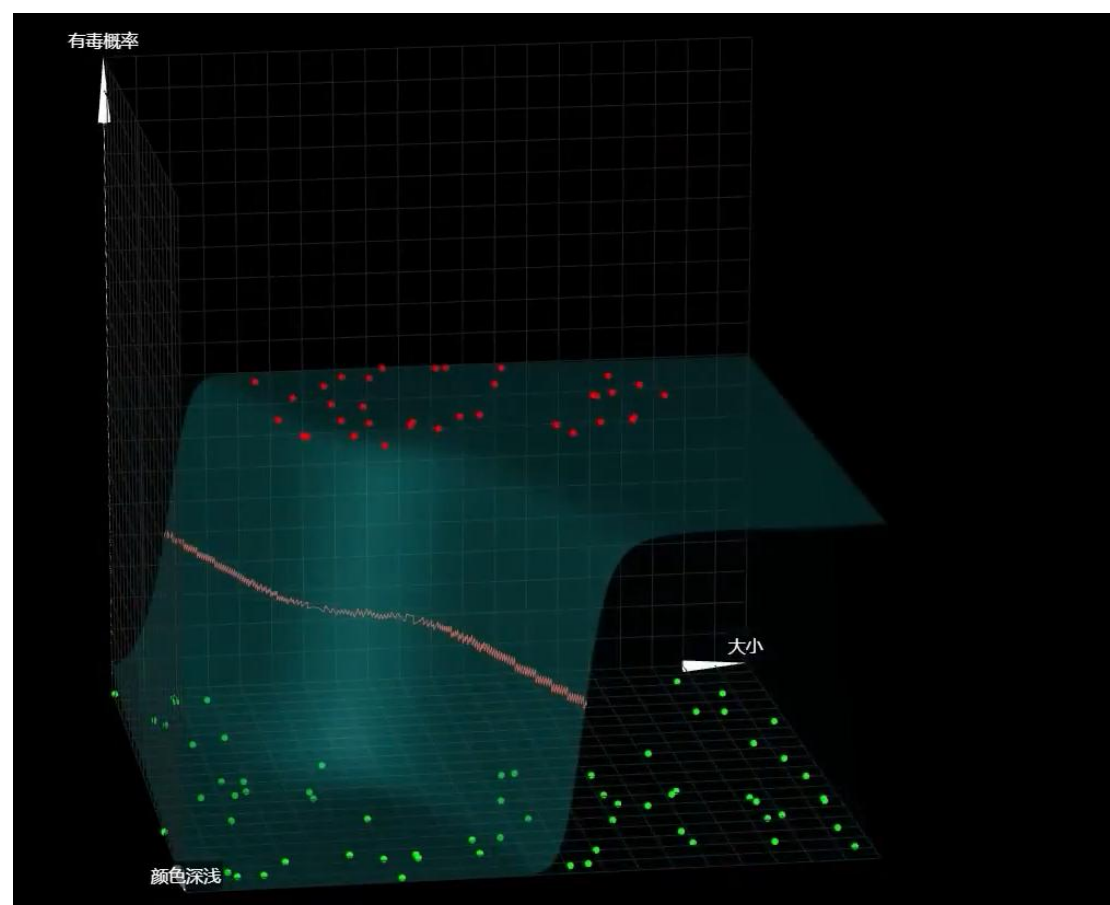
加入隐藏层处理更加复杂的模型关系



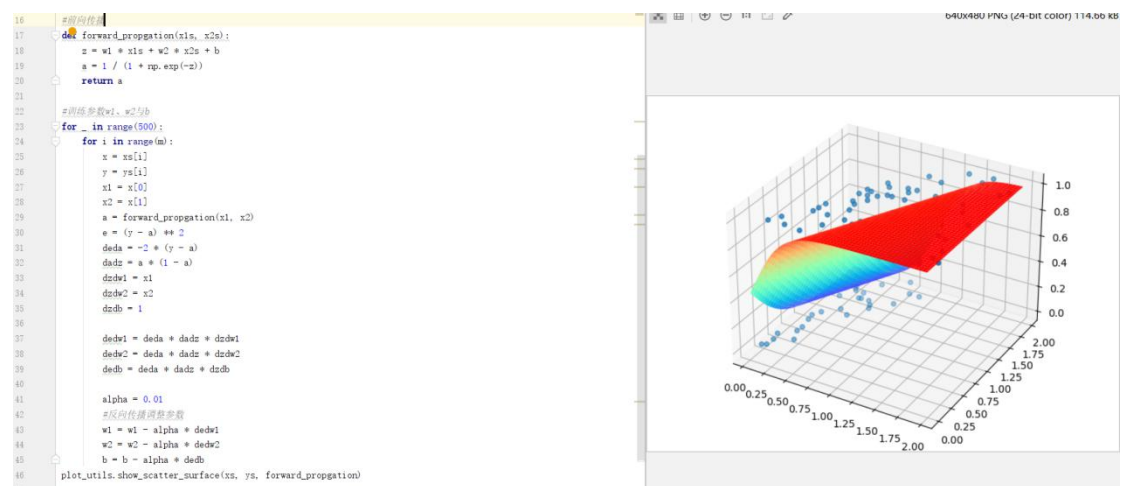
代码复现



7. 当毒性有毒概率不仅与大小有关还与颜色深浅有关时



代码复现



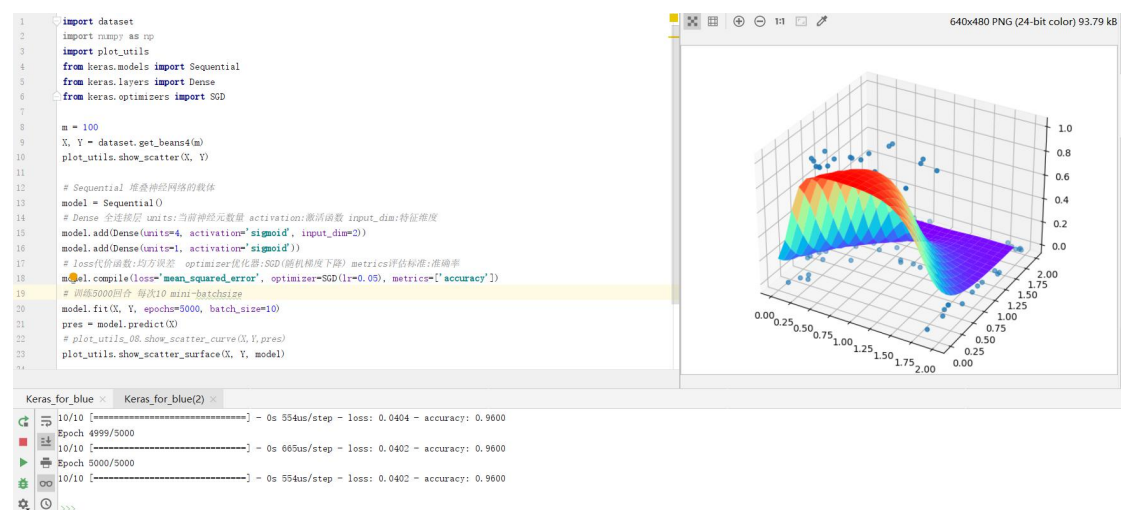
8. 初识 keras 框架

用矩阵模拟 $y=wx+b$ 的线性关系

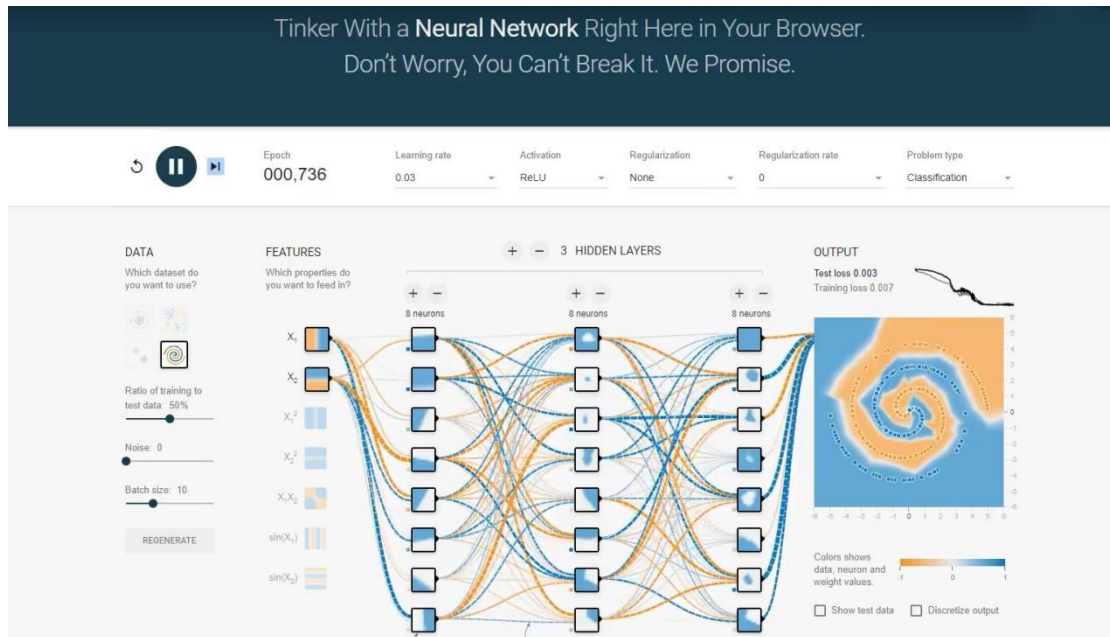


Keras 是一种高度模块化, 使用简单上手快, 合适深度学习初学者使用的深度学习框架。

代码复现

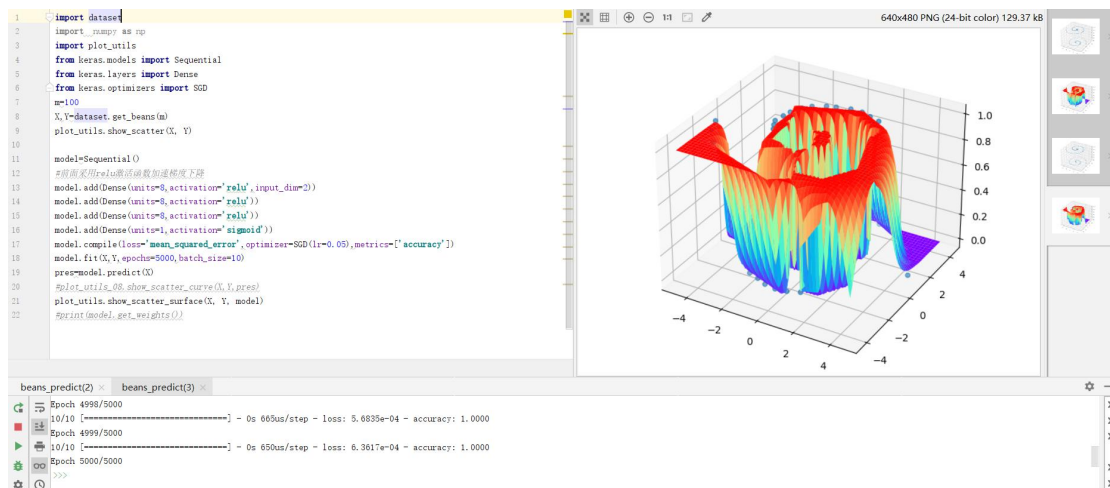


9. 利用 keras 框架拟合如下图图形



前面 3 层采用 relu 激活函数加速训练,输出层用 sigmoid 拟合

代码复现

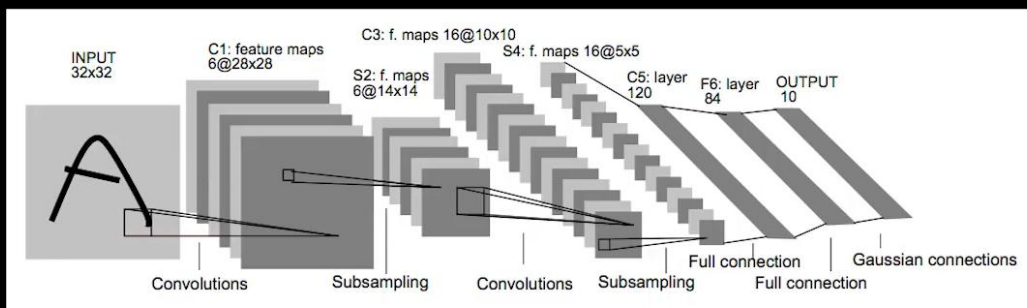


10. 初识 CNN

CNN 一般用于处理空间序列结构的数据。

图像中不同数据窗口的数据和卷积核（一个滤波矩阵）作内积的操作叫做卷积。

LeNet-5网络



代码复现

使用传统的全连接层 $28 \times 28 = 784$ 维度进行训练预测 mnist 识别,为加速训练,简单采取训练 50 回合,每次训练批次 10240 次,测试集准确率为 91.67%。

```
16 # plt.imshow(X_train[0], cmap='gray')
17 # plt.show()
18
19 X_train=X_train.reshape(60000,784)/255 #归一化
20 X_test=X_test.reshape(10000,784)/255
21
22 Y_train=to_categorical(Y_train,10)
23 Y_test=to_categorical(Y_test,10)
24 model=Sequential()
25 model.add(Dense(units=256,activation='relu',input_dim=784))
26 model.add(Dense(units=256,activation='relu'))
27 model.add(Dense(units=256,activation='relu'))
28 model.add(Dense(units=10,activation='softmax'))
29 #交叉熵代价函数多分类categorical_crossentropy
30 model.compile(loss='categorical_crossentropy',optimizer=SGD(lr=0.05),metrics=['accuracy'])
31 model.fit(X_train,Y_train,epochs=50,batch_size=10240)
32 loss,accuracy=model.evaluate(X_test,Y_test)
33 print("loss"+str(loss))
34 print("accuracy"+str(accuracy))
35
```

mnist_recoginze	
Epoch 48/50	6/6 [=====] - 1s 97ms/step - loss: 0.3097 - accuracy: 0.9116
Epoch 49/50	6/6 [=====] - 1s 96ms/step - loss: 0.3074 - accuracy: 0.9118
Epoch 50/50	6/6 [=====] - 1s 98ms/step - loss: 0.3048 - accuracy: 0.9127
313/313 [=====] - 1s 2ms/step - loss: 0.2909 - accuracy: 0.9167	
loss: 0.29085054993629456	
accuracy: 0.916700005531311	

采用 LeNet-5 结构进行 CNN 卷积在同批次情况进行训练得到的测试集准确率为 92.23%, 比直接平铺展开 784 全连接层维度的训练效果好。

```

13 X_train = X_train.reshape(60000, 28, 28, 1) / 255 # 归一化
14 X_test = X_test.reshape(10000, 28, 28, 1) / 255
15
16 Y_train = to_categorical(Y_train, 10)
17 Y_test = to_categorical(Y_test, 10)
18 model = Sequential()
19 # filters卷积核数量 kernel_size卷积核尺寸, strides卷积核步长 input_shape输入形状 padding:填充方式valid不加填充 same加填充
20 model.add(
21     Conv2D(filters=6, kernel_size=(5, 5), strides=(1, 1), input_shape=(28, 28, 1), padding='valid', activation='relu'))
22 # 2*2平均池化 数据的下采样
23 model.add(AveragePooling2D(pool_size=(2, 2)))
24 model.add(Conv2D(filters=16, kernel_size=(5, 5), strides=(1, 1), padding='valid', activation='relu'))
25 model.add(AveragePooling2D(pool_size=(2, 2)))
26 # 全连接层 池化层输出平铺成数组
27 model.add(Flatten())
28 # 120个神经元隐藏层
29 model.add(Dense(units=120, activation='relu'))
30 model.add(Dense(units=84, activation='relu'))
31 # softmax多分类输出层
32 model.add(Dense(units=10, activation='softmax'))
33 # 交叉熵代价函数
34 model.compile(loss='categorical_crossentropy', optimizer=SGD(lr=0.05), metrics=['accuracy'])
35 model.fit(X_train, Y_train, epochs=50, batch_size=10240)
36 loss, accuracy = model.evaluate(X_test, Y_test)

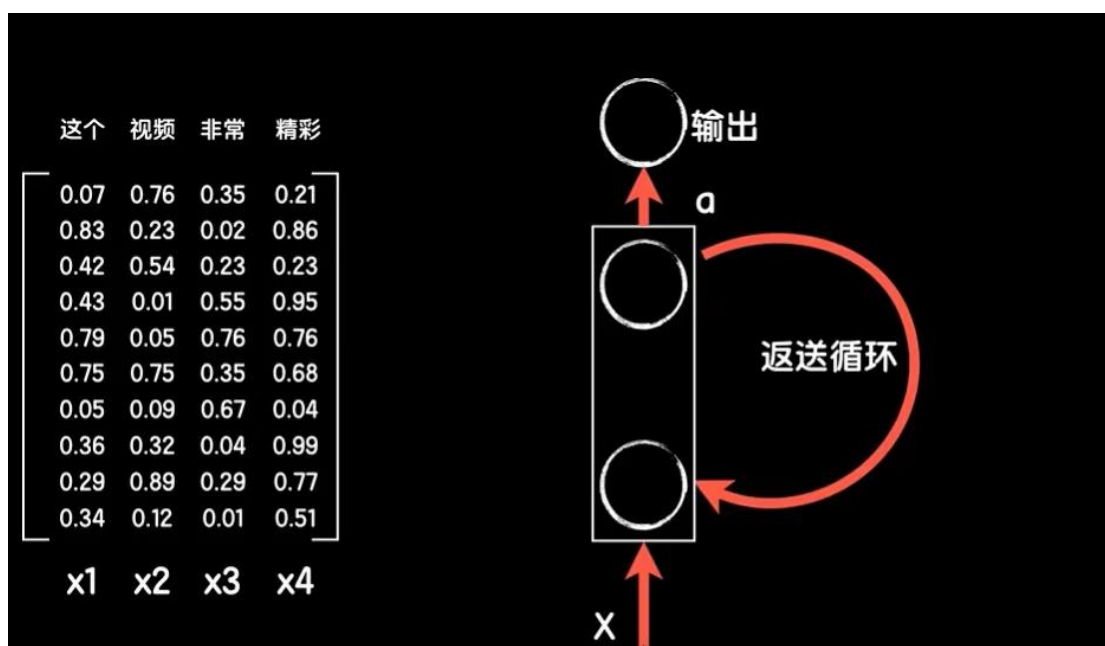
```

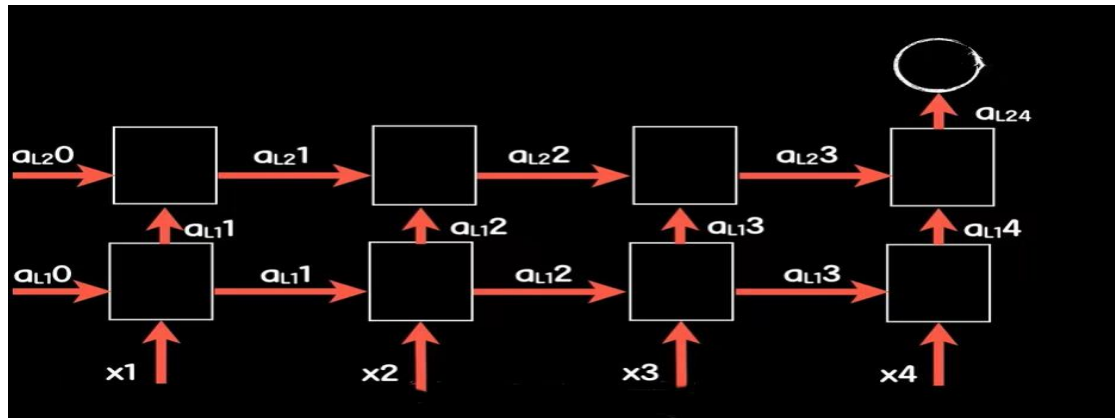
mnist_recoginze x LeNet_5 x

Epoch 50/50
 6/6 [=====] - 5s 771ms/step - loss: 0.2751 - accuracy: 0.9167
 313/313 [=====] - 1s 3ms/step - loss: 0.2541 - accuracy: 0.9223
 loss 0.2540920376777649
 accuracy 0.9222999811172485
 >>>

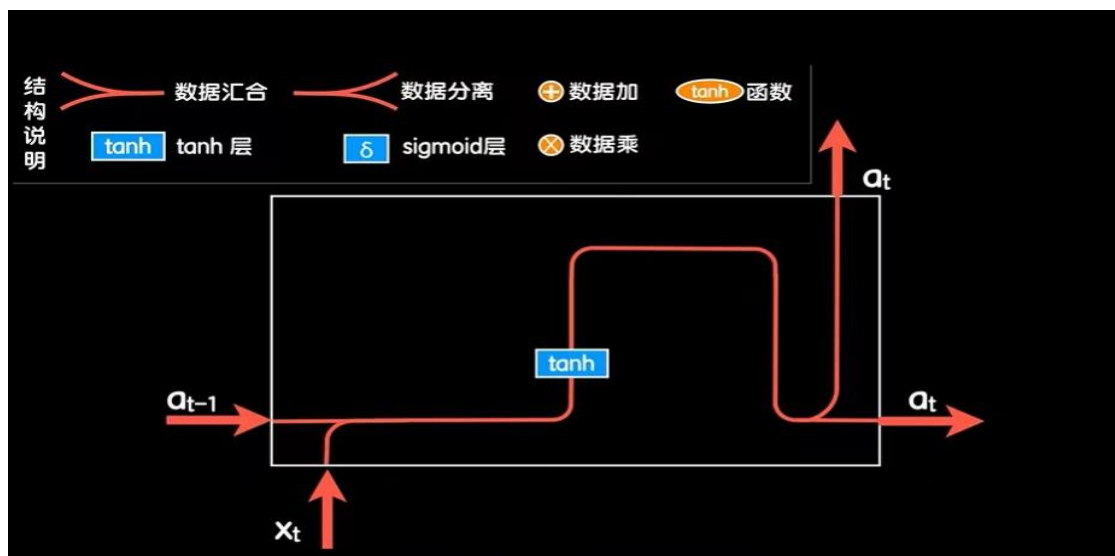
11. 初识 RNN

RNN 一般处理具有时间序列特征的数据。

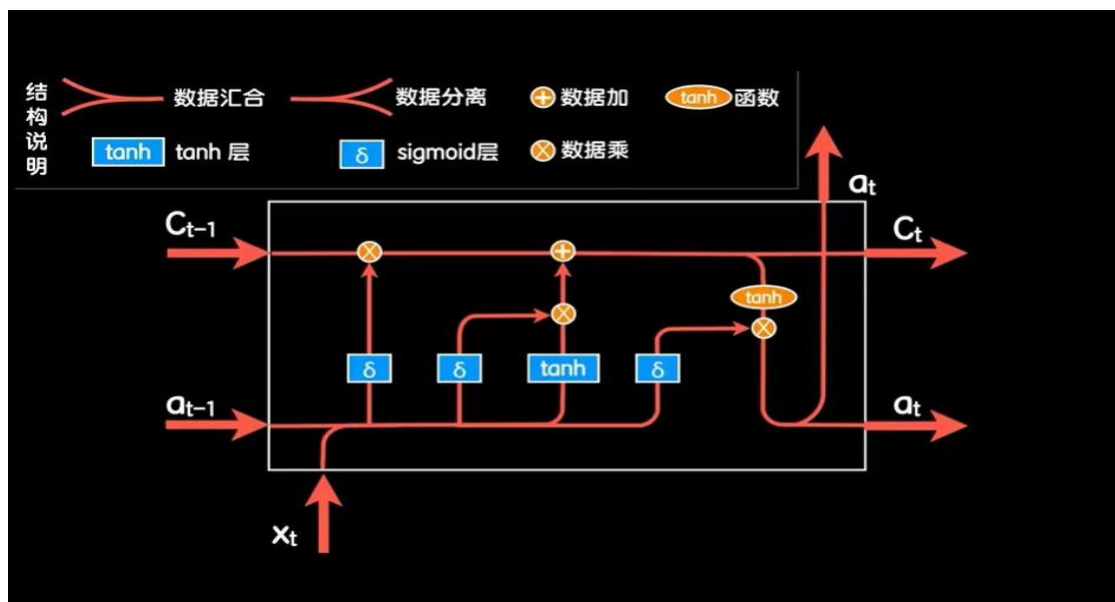




RNN 结构图



LSTM 结构图(相比于传统的 RNN 模型增加了遗忘门、更新门、输出门和细胞状态从而达到长短时记忆的功能,能较好的处理时间上具有长依赖序列的数据)。



代码复现

对购物的评论的正面评论还是负面评论进行情感分类,数据集中衣服代表商品种类,1 代表积极评论情感分类,0 代表负面评论情感分类,右侧表示商品评论详情。

衣服, 1, 面料舒适度: 衣服的质量很好, 穿上很舒服, 很修身, 特有范, 卖家的服务态度好, 物流快, 下次还会再来的。 衣服, 1, 裤子真的超赞 不论是材质还是样式 物超所值 推荐大家购买 现在穿季节正好 衣服, 1, 很满意的一次购物, 一个字, 好! 衣服, 1, 衣服质量很好, 无论是面料还是内里以及做工都是无可挑剔。感觉既有深度又不失青春活力, 是值得拥有的春、秋、冬三季的必备款。 衣服, 1, 老公穿起来真好看, 大小刚刚好, 这个价钱真划算, 质量也很好, 朋友让我再帮他们买, 喜欢就赶快买吧, 真的很好 衣服, 1, 牛仔裤收到, 质量不错, 裤型好, 肥瘦刚刚好, 物流好快。 衣服, 1, 裤子质量还不错, 货发到的也很及时 衣服, 1, 裤子质量很不错, 穿着很舒服, 以后还会再来~ 衣服, 1, 版型很不错, 大小也合适。难得能在网上买到喜欢又合适的衣服。 衣服, 1, 裤子挺不错的, 基本上我家人穿的裤子都让这家店铺承包了, 足见其裤子质量好的没话说! 衣服, 1, 弹性效果: 收到裤子, 一打个首先包装很规整, 拆包之后我等一个就是看口袋的布料的材质, 质量不错, 先赞一个。然后上身试穿, 版型很好, 修身, 合身, 长度也合适, 在赞一个 衣服, 1, 不错。快速很快, 衣服还是这么好, 又打折 衣服, 0, 穿上不舒服, 颜色和质感跟图片差异很大, 建议慎重购买! 后悔了! 衣服, 0, 真心垃圾, 以后再也不会买班尼路了 衣服, 0, 物流超慢, 整整6天多, 准备退订了才到, 无语。衣服质量嘛, 中下水准, 穿着不太舒服。衣服穿起来的外形跟农民工差不多, 设计太垃圾了。总之不值这个价, 三分之一的价格买来都觉得亏。谁买谁哭! 衣服, 0, 这个T恤不安逸得, 和农贸市场一二十的差不多样 衣服, 0, 你以为是Lee那就错了, 你以为是Lee一半的品质, 那就又错了, 你以为会送货很快, 那就又错了! 衣服, 0, “无语-是袖子吗?” 衣服, 0, 发货慢, 10天才收到, 质量差, 线头松动, 第二天就起毛, 建议大家不要在这买, 过来人给他家的忠告。 衣服, 0, 我只能说质量很差! 肯定不是正品, 衣服, 0, 跟图片不一样 衣服, 0, 差评, 半蓝色发的黑色, 32-发的31 衣服, 0, 裤子好差劲, 要沾毛, 真是无语了还那么贵。 衣服, 0, 穿的头一天就有个洞洞差评! 衣服, 0, 垃圾, 还说质量保证, 朋友说算了, 叫我不退货 衣服, 0, 裤子一般, 然而发货的时候本来买的两条才给我发了一条, 送的皮带太大根本穿不了, 直接扔了。 衣服, 0, 与图跟本不符, 穿都穿不上, 店家服务特别差, 而且退货运费还要自己出, 一星都不想给, 在京东购物最差的一次 衣服, 0, 烂, 退货后, 到现在没有返钱。这一星, 是给京东的。 衣服, 0, 质量不好, 穿两个月就变形了。 衣服, 0, 质量一般, 实物前面的印花商标没有, 衣服, 0, 很好很好很好很好很好 衣服, 0, 很好很好很好很好很好	
---	--

通过词嵌入直接全连接层平铺展开构建模型在商品留言描述情感分类中测试集的准确率为 85.35%。

```
20 x_train_index = shopping_data.word2Index(x_train, word_index)
21 x_test_index = shopping_data.word2Index(x_test, word_index)
22
23 maxlen = 25
24 # 把序列按照maxlen进行对齐 长度不足25的句子就会被0补齐为25
25 x_train_index = sequence.pad_sequences(x_train_index, maxlen=maxlen)
26 x_test_index = sequence.pad_sequences(x_test_index, maxlen=maxlen)
27 |
28 model = Sequential()
29 # 嵌入层 trainable=True 是否让这一层在训练时更新参数 input_dim:输入维度 output_dim:输出维度 input_length:句子序列长度
30 model.add(Embedding(trainable=True, input_dim=vocalen, output_dim=300, input_length=maxlen))
31 model.add(Flatten())
32 model.add(Dense(256, activation='relu'))
33 model.add(Dense(256, activation='relu'))
34 model.add(Dense(256, activation='relu'))
35
36 model.add(Dense(1, activation='sigmoid'))
37 # loss:交叉熵代价函数 adam:是一种使用动量的自适应优化器
38 model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
39 model.fit(x_train_index, y_train, batch_size=512, epochs=100)
40 score, acc = model.evaluate(x_test_index, y_test)
41 print("Test score:", score)
42 print("Test accuracy:", acc)
```

comments_recognizer ✕

26/26 [=====] - 2s 61ms/step - loss: 0.0040 - accuracy: 0.9975

Epoch 100/100

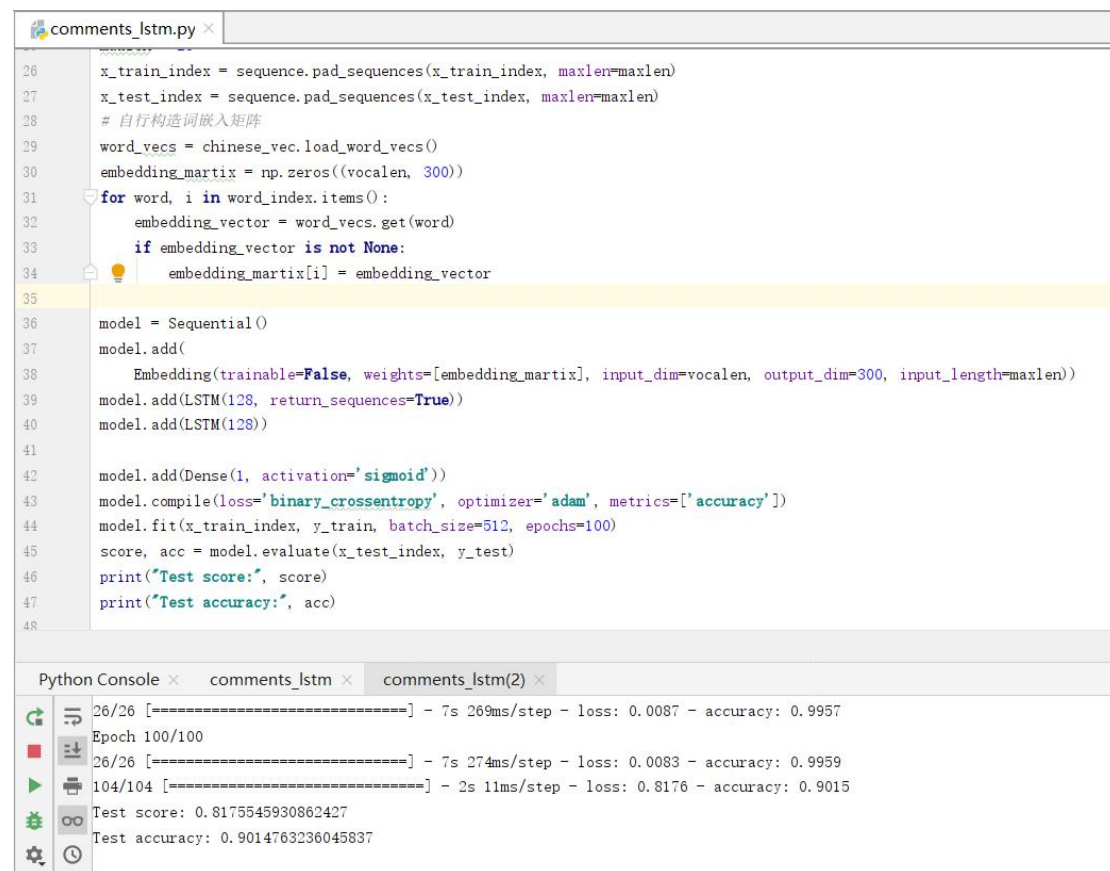
26/26 [=====] - 2s 63ms/step - loss: 0.0040 - accuracy: 0.9977

104/104 [=====] - 0s 2ms/step - loss: 1.6335 - accuracy: 0.8536

Test score: 1.6334710121154785

Test accuracy: 0.8535703420639038

加入 LSTM 层训练后商品情感分类在测试集得到准确率为 90.15%。比传统的全连接层平铺展开构建模型的训练得到的 85.35%测试集准确率有显著提升。



```
26 x_train_index = sequence.pad_sequences(x_train_index, maxlen=maxlen)
27 x_test_index = sequence.pad_sequences(x_test_index, maxlen=maxlen)
28 # 自行构造词嵌入矩阵
29 word_vecs = chinese_vec.load_word_vecs()
30 embedding_matrix = np.zeros((vocalen, 300))
31 for word, i in word_index.items():
32     embedding_vector = word_vecs.get(word)
33     if embedding_vector is not None:
34         embedding_matrix[i] = embedding_vector
35
36 model = Sequential()
37 model.add(
38     Embedding(trainable=False, weights=[embedding_matrix], input_dim=vocalen, output_dim=300, input_length=maxlen))
39 model.add(LSTM(128, return_sequences=True))
40 model.add(LSTM(128))
41
42 model.add(Dense(1, activation='sigmoid'))
43 model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
44 model.fit(x_train_index, y_train, batch_size=512, epochs=100)
45 score, acc = model.evaluate(x_test_index, y_test)
46 print("Test score:", score)
47 print("Test accuracy:", acc)
48
```

Python Console × comments_lstm × comments_lstm(2) ×

```
26/26 [=====] - 7s 269ms/step - loss: 0.0087 - accuracy: 0.9957
Epoch 100/100
26/26 [=====] - 7s 274ms/step - loss: 0.0083 - accuracy: 0.9959
104/104 [=====] - 2s 11ms/step - loss: 0.8176 - accuracy: 0.9015
Test score: 0.8175545930862427
Test accuracy: 0.9014763236045837
```

12. 代码地址

https://github.com/PeakWan/neural_network_learn_start.git