电影评论分类：二分类问题

数据集：IMDB

包含 50000 条严重两级分化的评论，测试集和数据集各一半，且包含 50%的正向评论和 50%的负向评论。

在 keras 中，imdb 已经内嵌，直接调用接口即可获取。

```
from keras.datasets import imdb
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words = 10000)
```

1.数据集处理

特征处理方法：

第一步，将单词转为数字：定义一个字典（数组），每一个索引对应一个单词，单词出现频率越高索引越低。对于每一条评论，可以将其拆分成许多单词的组合，然后适当舍弃低频词汇（比如只取索引前 10000 的值），这样就可以把单词数据转换为数字数组；

第二步，将数字转为合适的矩阵。每一条评论是一个数组，那么数据集就是一个行为评论数，列不确定（由评论单词数决定）的二位数组，将其转为确定的二维数组：因为先前舍弃低频词后只剩 10000 中单词可能，所以将二维矩阵扩展为 10000 列，每一列对应某个单词是否出现。

标签处理方法：

标签只有两种可能，即评论为正向还是负向。有两种方法，一种是整数张量编码，即正向为 1，负向为 0，只需要一个标量；一种是单热点编码，即一个二元数组，第一个为 1 表示负向，第 2 个为 1 表示正向。后面的代码使用第二种方法。

处理代码：

对于特征的处理：

```
def vectorixe_sequences(sequences, dimension = 10000):
    rst = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        rst[i,sequence] = 1

    return rst
```

对于标签的处理，keras 提供了单热点编码的接口，直接调用即可：

```
from keras.utils import to_categorical
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

2.定义中间层：

首先需要确定输入的维度，每条评论有 10000 个特征，所以有 10000 个输入；

中间层使用 relu 激活函数，节点数取 16，放两层；

输出层大小为 2，因为前面使用了单热点编码，使用 sigmoid 激活函数

```
model = models.Sequential()
model.add(layers.Input((10000,)))
model.add(layers.Dense(16, activation=activations.relu))
model.add(layers.Dense(16, activation=activations.relu))
model.add(layers.Dense(2,activation=activations.sigmoid))
```

3.优化算法和损失函数的选取：

优化函数使用均方根传播优化，损失函数二元交叉熵损失。此外追踪结果的准确度。

```
model.compile(
    optimizer=optimizers.RMSprop(),
    loss=losses.binary_crossentropy,
    metrics=[metrics.binary_accuracy]
)
```

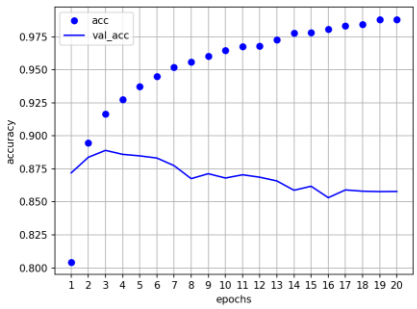
4.模型训练

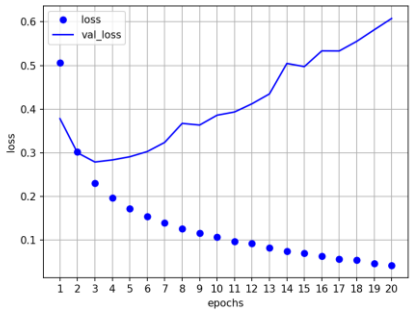
使用预处理好的数据训练模型，批量大小为 512，迭代次数为 20，同时使用测试集评估模型

```
history = model.fit(
    x_train,
    y_train,
    batch_size=512,
    epochs=20,
    validation_data=(x_test, y_test)
)
```

5.模型评估

准确度：　　损失：



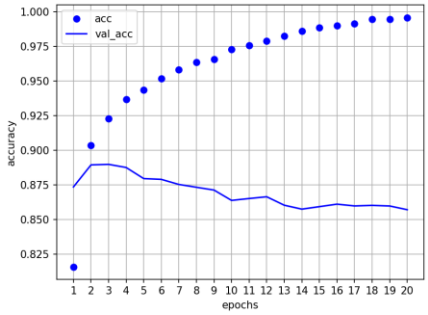


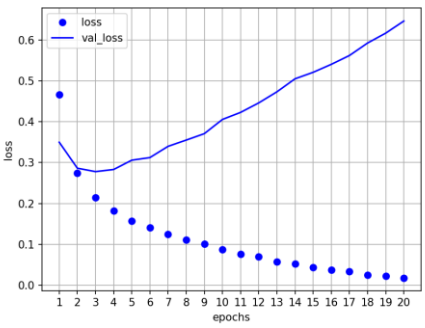
可以看到在第 3 轮左右模型在测试集上的效果达到最佳，准确率达到 88%左右，且损失最小，往后模型准确率开始逐步下降，且损失增大，而测试集准确率在上升，损失在下降，可以认为是过拟合了，所以迭代 3 次是最优解。

6.其他

补充另一种做法，即标签值不使用单热点，也很简单，取消原先的单热点编码，并且在最后的输出层，将输出个数设置为 1 即可。

最后的效果和上面的做法相差不大：





新闻分类：多分类问题

数据集：路透社数据集（reuters）

这个数据集和二元分类的数据集很像，也是对文字处理分类，区别在于这个数据集的输出有 46 种（46 种新闻类型）。训练集有 8982 条数据，测试集有 2246 条数据。

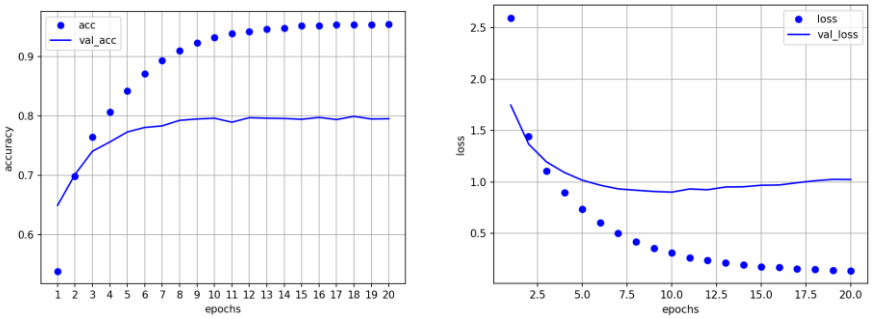
处理方法和代码结构与先前基本一致，区别在于：

（1）神经网络最后一层输出为 46 个，因为要分 46 类，激活函数使用 softmax；

（2）中间层的参数要多一些（比如 64），过少会导致信息保存太少而降低精度，过多训练太慢且可能过拟合；

（3）损失函数改为分类交叉熵损失

最后拟合结果：



根据结果分析，在第 10 轮左右效果比较好，后面准确率基本维持不变，损失有缓慢上升的趋势。

图像分类问题

数据集：MNIST/Fashion Mnist

数据集包含 10 中不同类别的图片，训练集 60000 个样本，测试集 10000 个样本，每张图片是 28*28 的灰度图，mnist 是图像分类最经典的数据集。mnist 是数字 0-9，fashion mnist 是 10 中不同的服装。

1.数据集处理

图像一般会有三个维度：（通道，高度，宽度），有一些库会把通道这个维度放在后面，有一些在前面。mnist 的灰度图只有一个通道，所以这个维度可以忽略，加上样本的维度，共有三个维度：（样本，高度，宽度），训练集 60000 个样本，所以形状为：(60000,28,28)

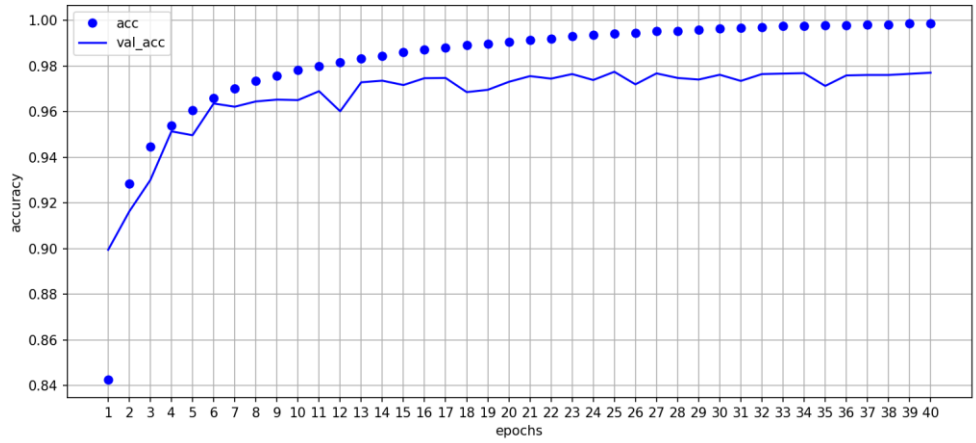
而高度和宽度可以合并为一个维度，即把 28*28 的矩阵转为长 784 的向量，这样就可以把数据转为 2 维处理。

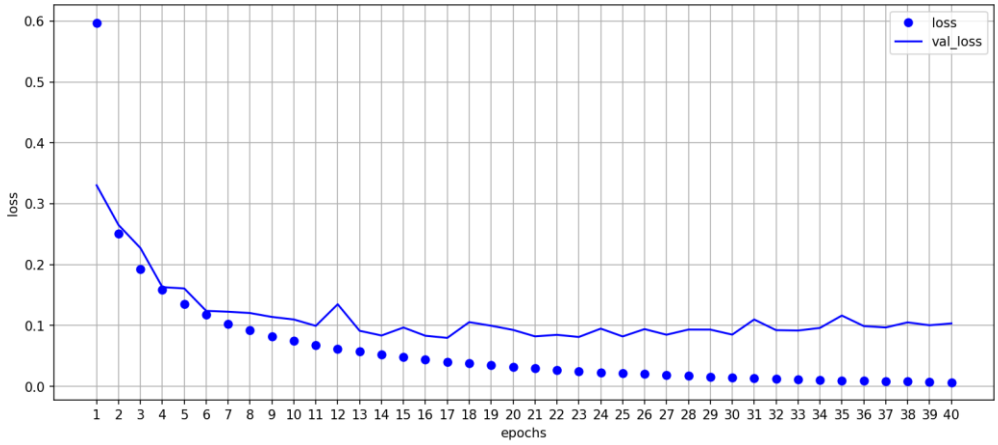
2.模型定义

模型和之前用的一致，使用两个全连接层，最后输出层的激活函数使用 softmax，因为图像分类本质还是多分类问题。

```
model = models.Sequential()
model.add(layers.Input((28*28,)))
model.add(layers.Dense(64, activation=activations.relu))
model.add(layers.Dense(64, activation=activations.relu))
model.add(layers.Dense(10, activation=activations.softmax))
```

3.模型评估





测试集 13 轮后准确率基本维持在 97%，损失也基本维持在 0.1，并且损失有缓慢上升的趋势，所以 13 轮应该是一个比较合适的轮数。

chineseMNIST

这个数据集也是 MNIST 的子类，包含 15 种手写汉字共 15000 个样本，每个样本大小为 64*64，同样也是灰度图。

1.数据集处理

这个数据集需要自己写导入函数，具体定义如下：

```
class ChineseMNIST:
    def __init__(self):
        # 导入数据
        self.json_data = json.loads(open("./chineseMNIST.json",mode="r",encoding="utf-8").read())
        np.random.shuffle(self.json_data)

    def load_data(self,num_train:int = 10000) -> tuple[tuple[np.ndarray, np.ndarray],tuple[np.ndarray, np.ndarray]]:
        """
        args:
            num_train: size of train dataset.
            range: (0, 15000)
            the num_test will be 15000 - num_train.
        return:
            (x_train, y_train), (x_test, y_test)
        """
```

ChineseMNIST 类实现了从 json 中获取信息，调用 load_data 时可以根据 json 中的信息来加载对应的图片，并获取图片的标签。

json 中的信息格式如下：


```
[
    {
        "path": "G:/\u591a\u6a21\u6001\u77e5\u8bc6\u95ee\u7b54\u7cfb\u7edf/\u6570\u636e\u96c6/chineseMNIST/data/input_100_10_1.jpg",
        "label": "1"
    },
    {
        "path": "G:/\u591a\u6a21\u6001\u77e5\u8bc6\u95ee\u7b54\u7cfb\u7edf/\u6570\u636e\u96c6/chineseMNIST/data/input_100_10_10.jpg",
        "label": "10"
    },
    {
        "path": "G:/\u591a\u6a21\u6001\u77e5\u8bc6\u95ee\u7b54\u7cfb\u7edf/\u6570\u636e\u96c6/chineseMNIST/data/input_100_10_11.jpg",
        "label": "11"
    },
```


每条信息包含 path 和 label，path 是图片保存的路径，label 是这个图片汉字的索引值+1，索引表：

```
注意当前json文件的标签值在此基础上+1
图像标签对应表如下：
零  --  0
一  --  1
二  --  2
三  --  3
四  --  4
五  --  5
六  --  6
七  --  7
八  --  8
九  --  9
十  --  10
百  --  11
千  --  12
万  --  13
亿  --  14
```

load_data 函数接受传入的参数 num_train，表示训练集的大小。然后这个函数会将数据集打乱并拆分成数据集和训练集返回，具体实现如下：

```
if not (num_train > 0 and num_train < 15000):
    raise ValueError(
        "num_train must in the range (0,15000)"
    )
self.x_train = np.zeros((num_train,64*64),dtype=np.int8)
self.y_train = np.zeros((num_train),dtype=np.int64)
self.x_test = np.zeros((15000-num_train,64*64),dtype=np.int8)
self.y_test = np.zeros((15000-num_train),dtype = np.int64)

for i, sample in zip(range(num_train), self.json_data[:num_train]):
    path = sample["path"]
    label = int(sample["label"]) - 1
    self.x_train[i, range(64*64)] = plt.imread(path).flatten() # 注意这里要把导入的图片展平为一维向量
    self.y_train[i] = label
for i, sample in zip(range(num_train,15000), self.json_data[num_train:]):
    path = sample["path"]
    label = int(sample["label"]) - 1
    self.x_test[i-num_train, range(64*64)] = plt.imread(path).flatten() # 这里也要展平
    self.y_test[i-num_train] = label
return (self.x_train, self.y_train),(self.x_test, self.y_test)
```

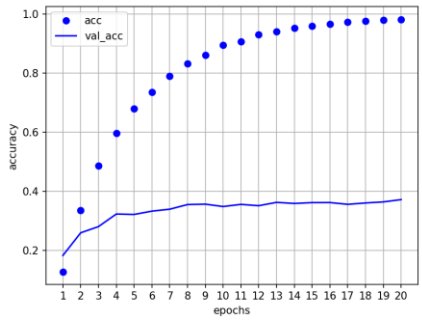
假设 num_train = 10000，最后返回时的形状为：

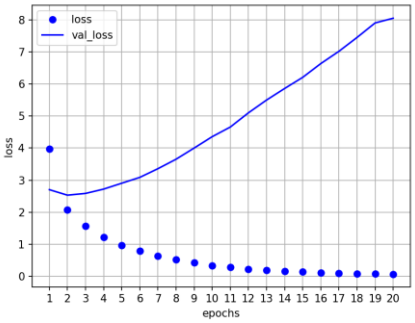
```
x_train.shape: (10000, 4096)
y_train.shape: (10000,)
x_test.shape: (5000, 4096)
y_test.shape: (5000,)
```

注意 4096 是将 64*64 的图片展平为 1 维。

之后还可以将 y_train 和 y_test 改为单热点编码，这样就和前面的处理一样了。模型定义和训练过程均与前面类似。

遇到问题：在测试集上，模型的准确率非常低，只有 38%，问题在于只有两个线性层的神经网络过于简单。





AI 给出的解决方法：使用卷积神经网络：

```
x_train = x_train.reshape((-1,64,64,1))
x_test = x_test.reshape((-1,64,64,1))
model = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(64,64,1)),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(15, activation='softmax')
])
```

由于还没学到卷积神经网络，这一部分还看不懂，之后的预测准确率就很高了：