回归问题

数据集：波士顿房价（boston_housing）

该数据集包含的内容很少，训练集包含 404 个样本，测试集包含 102 个样本，每个样本有 13 个特征值，1 个标签值。特征值是各种用于评估房屋价值的参数，标签值为房价（单位：K dollars）

1.数据集处理：

特征：

不同的特征取值范围也不同，所以需要进行数据标准化（减去数据平均值，除以标准差），将其转化为标准正态分布值，注意 13 个特征是独立的，每个特征分别计算，可以使用广播机制做到。

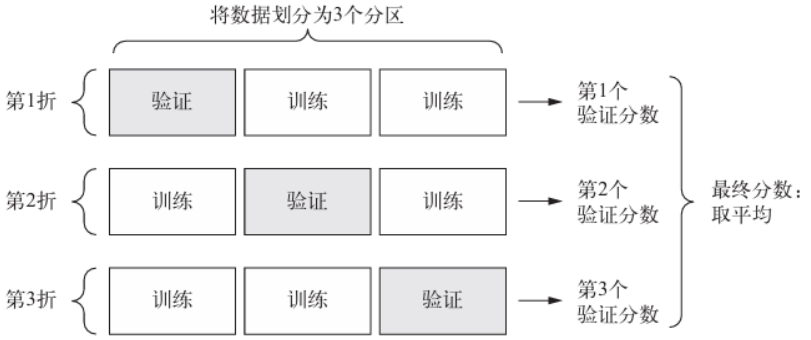
标签：

标签只有房价，不需要特殊处理

由于数据集很小，所以需要引入一个新的验证方法：K 折交叉验证

实例化 K 个相同的模型，将每个模型在 K-1 个分区上训练，并在剩 下的一个分区上进行评估。模型的验证分数等于 K 个验证分数的平均值。

3 折交叉验证为例：



具体可以参照实际实现的代码。

2.模型定义

和先前一样，隐藏层两个，每层 64 个单元。输入层 13，输出层 1。

注意这里输出层不要用激活函数，先前使用的激活函数，比如 sigmoid 会将单个输出限定在 0-1 的区间中，适合二分类，softmax 会将多个输出的和限制为 1，适合多分类，而回归问题，比如预测房价需要的输出没有准确范围，所以不需要。

损失函数使用均方误差 mse，监听平均绝对误差 mae。

```
# 由于后面需要多次定义模型，将模型封装到函数中
def build_model():
    model = models.Sequential()
    model.add(layers.Input((13,)))
    model.add(layers.Dense(64, "relu"))
    model.add(layers.Dense(64, "relu"))
    model.add(layers.Dense(1))
    model.compile(
        optimizer="rmsprop",
        loss=losses.mean_squared_error,
        metrics=[metrics.mean_absolute_error]
    )
    return model
```

3.K 折验证

数据集较少，使用 K 折验证。具体解释如图。

getFoldData 会根据传入的参数 i 将原先的 x_train 和 y_train 拆分。

```
"""
K折验证：
    将数据集拆分为几个分区，实例化K个相同模型，每个模型在K-1个分区上训练，在剩下的一个分区上评估，
    模型的验证分数等于K个验证分数的平均值
    下面以四个分区为例做K折验证
"""
k = 4 # 分区数
num_val_samples = len(x_train) // k # 每个分区的样本数
num_epochs = 500 # 循环次数
mae_history = [] # 保存每一个模型的平均绝对误差值mae

# 这个函数用来拆分验证集和测试集
def getFoldData(i:int):
    val_data = np.zeros((num_val_samples, 13))
    val_targets = np.zeros((num_val_samples,))

    val_data = x_train[i*num_val_samples:(i+1)*num_val_samples]
    val_targets = y_train[i*num_val_samples:(i+1)*num_val_samples]

    partical_train_data = np.concatenate([x_train[:i*(num_val_samples)], x_train[(i+1)*num_val_samples:]],axis=0)
    partical_train_targets = np.concatenate([y_train[:i*num_val_samples],y_train[(i+1)*num_val_samples:]],axis = 0)
    return (partical_train_data, partical_train_targets), (val_data, val_targets)
```

4.训练模型

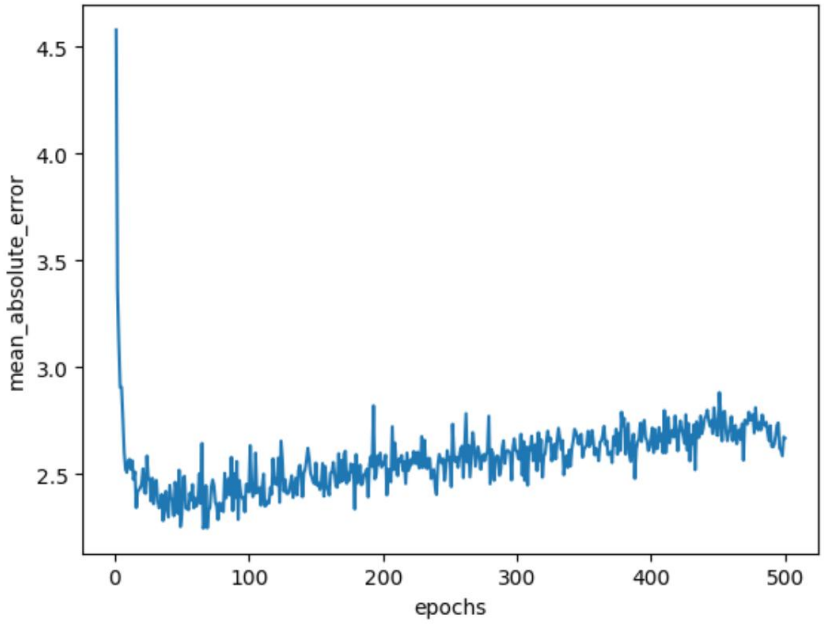
训练 4 个模型，记录 mae 并求平均值。

```
# 训练模型
for i in range(k):
    print("processing fold #", i)
    (partical_train_data, partical_train_targets), (val_data, val_targets) = getFoldData(i)
    print(partical_train_data.shape, partical_train_targets.shape)
    print(val_data.shape,val_targets.shape)
    model = build_model()
    history = model.fit(
        partical_train_data,
        partical_train_targets,
        batch_size=1,
        epochs=num_epochs,
        verbose="0",
        validation_data=(val_data, val_targets)
    )
    mae_history.append(history.history["val_mean_absolute_error"])

average_mae_history = [(np.mean([x[i] for x in mae_history])) for i in range(num_epochs)]
```

5.模型评估

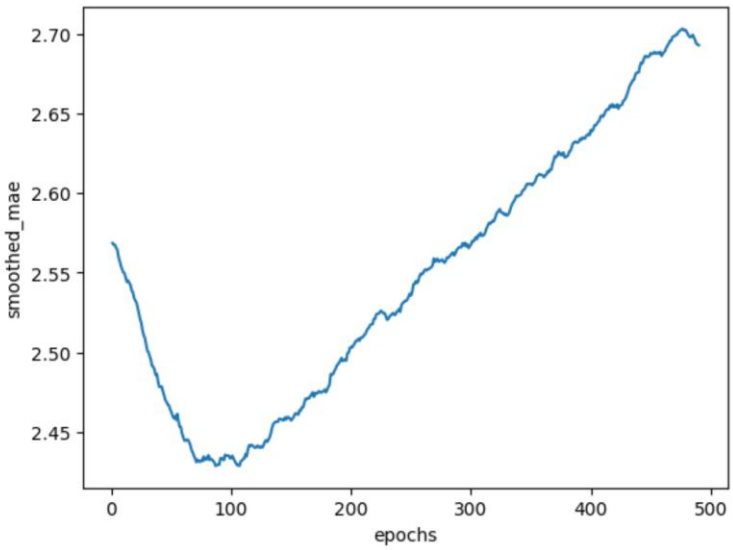
将记录的 mae 可视化，能够看到前面几轮后模型的误差大幅下降，之后在一个范围内抖动。

```python
# 将数据可视化
import matplotlib.pyplot as plt

plt.plot(range(1, len(average_mae_history)+1), average_mae_history)
plt.xlabel("epochs")
plt.ylabel("mean_absolute_error")
plt.show()
```



由于无法直接看到规律，所以需要对数据处理，具体为舍弃前面 10 个数据点，剩余数据点使用指数移动平均值，平滑曲线后重新分析。

```python
"""
原始数据波动太大没有办法看到实际规律，对数据做如下处理：
    1. 删除前十个数据点，因为这10个数据点不在大多数点的范围内
    2. 将数据点替换为前面数据点的指数移动平均值，平滑曲线
从图中可以看出大约在80轮时模型的误差达不再显著降低，100轮以后误差开始增大，过拟合
"""
def smooth_curve(points, factor = 0.98):
    smoothed_points = []
    for point in points:
        if smoothed_points:
            smoothed_points.append(smoothed_points[-1]*factor + point*(1 - factor))
        else:
            smoothed_points.append(point)

    return smoothed_points

smoothed_mae_history = smooth_curve(average_mae_history[10:])

plt.plot(range(1, len(smoothed_mae_history)+1), smoothed_mae_history)
plt.xlabel("epochs")
plt.ylabel("smoothed_mae")
plt.show()
```



80-100 轮时误差较小，后面就开始过拟合。

6.最后模型结果

新建模型，在所有训练集上训练，并使用测试集评估，训练 80 轮，查看最终误差。

```
# 新建一个模型，在这个模型上使用全部数据集，并使用测试集做测试，迭代80轮
# 最终预测的误差依然有2.52
model = build_model()
model.fit(
    x_train,
    y_train,
    epochs = 90,
    batch_size=16,
    verbose = "0"
)
test_mse, test_mae = model.evaluate(x_test, y_test)
print(test_mae)
```

```
2.5206878185272217
```

```
"""
小结:
    1.回归问题和分类问题不同，使用的损失函数为均方误差（mse）
    2.回归问题使用的回归指标，常见的是平均绝对误差(mae)
    3.输入特征如果具有不同的取值范围，应该先进行预处理，对每个特征单独进行缩放（标准化）
    4.可用数据很少时，使用K折验证可以评估模型，并且隐藏层要少，使用小型网络避免严重过拟合
"""
```

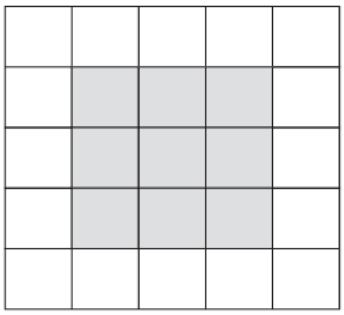
卷积神经网络

数据集：dogs-vs-cats

该数据集来源于 kaggle 的一次竞赛，公布的数据集中包含测试集图片 25000 张，其中猫狗各 12500 张，测试集 12500 张，猫狗图片均有。

1.卷积神经网络简析：

先前使用的都是密集连接层 Dense，它从特征空间中学习到的是涉及所有像素的模式（全局模式），而卷积层可以学到局部模式，假设窗口为 3*3，特征块为 5*5，如图：



卷积在输入特征图上滑动，在每一个 3*3 的窗口上提取信息，最后对提取到的信息进行空间重组并输出。

keras 中的卷积层使用 Conv 定义，对于当前的数据集使用 Conv2D。

最大池化运算：假设一个 28*28 的图像，经过输出深度为 32，窗口为 3*3 的卷积层，得到的形状为（32,26,26），当图片变大时，这个参数量是很大的，不仅会占用大量内存，还会造成严重的过拟合。最大池化通常是窗口大小为 2*2，步幅为 2 的卷积层，作用是将参数数量减少一半。

依然使用 mnist 数据集，将密集链接层和卷积神经网络做对比，原先的代码准确率为 97%，使用卷积神经网络构建：

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation="relu", input_shape = (28,28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation="relu"))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation="relu"))

model.add(layers.Flatten())
model.add(layers.Dense(64, activation="relu"))
model.add(layers.Dense(10, activation="softmax"))
```

最终的结果：

```
To enable the following instructions: AVX2 AVX_VNNI FMA, in other operations, rebuild TensorFlow wi
Epoch 1/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 14s 14ms/step - categorical_accuracy: 0.8667 - loss: 0.4119
Epoch 2/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 13s 14ms/step - categorical_accuracy: 0.9844 - loss: 0.0518
Epoch 3/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 13s 14ms/step - categorical_accuracy: 0.9894 - loss: 0.0335
Epoch 4/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 13s 14ms/step - categorical_accuracy: 0.9920 - loss: 0.0250
Epoch 5/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 13s 14ms/step - categorical_accuracy: 0.9943 - loss: 0.0184
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - categorical_accuracy: 0.9900 - loss: 0.0358
0.9922000169754028
```

最终预测的准确率可以达到 99.2%，可见是比先前好很多的。

2.数据集的处理

代码中只会用到 4000 张图片，具体切分代码参照 dataset_slice.py 代码，不做详细解释。

此外还会使用到一个自定义的图片迭代器，这个参照 myImageGenerator.py，这里只做简单介绍：这个迭代器接收图片路径，每个图片的标签（每个标签对应路径下某一图片，注意 windows 的文件系统排序方法，从左往右逐个匹配字符），以及一个批量大小参数 batch_size，每次迭代返回一个元组(file_list, label_list)，前一个元素是分装并压缩为(150, 150)大小的图片矩阵，后一个是该图片的标签，并且原先的顺序会被打乱。

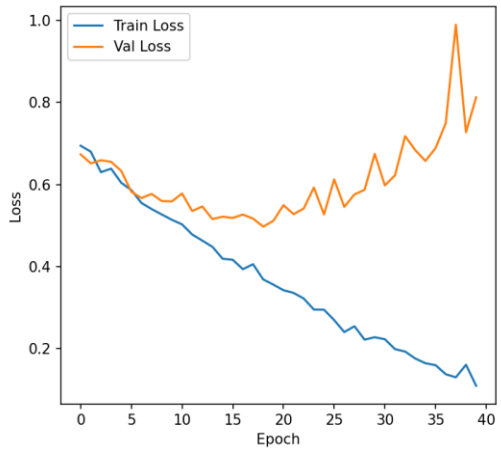
3.模型定义

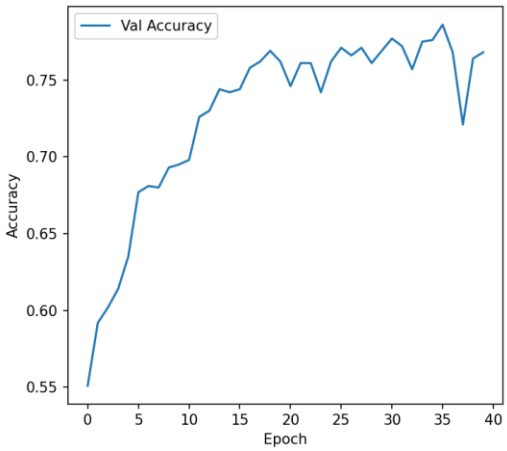
卷积层+输出层，卷积层将特征图缩小到一定程度后展平接输出。

```python
class CatDogCNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv_layers = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding = 1), # 输出32*150*150
            nn.ReLU(), # 记得加上激活函数
            nn.MaxPool2d(kernel_size=2, stride = 2), # 最大池化层，输出32*75*75

            nn.Conv2d(32, 64, kernel_size=3, padding=1), # 输出64*75*75
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2), # 输出64*37*37

            nn.Conv2d(64, 128, kernel_size=3, padding=1), # 输出128*37*37
            nn.ReLU(),
            nn.MaxPool2d(2, 2), # 输出128*18*18

            nn.Conv2d(128, 256, kernel_size=3, padding=1), # 输出256*18*18
            nn.ReLU(),
            nn.MaxPool2d(2, 2) # 输出256*9*9
        )

        self.fc_layers = nn.Sequential(
            nn.Flatten(),
            nn.Linear(256*9*9, 512),
            nn.ReLU(),
            nn.Dropout(0.5), # 正则化，防止过拟合
            nn.Linear(512, 1),
            nn.Sigmoid()
        )

    def forward(self, x):
        x = self.conv_layers(x)
        x = self.fc_layers(x)
        return x
```

4.训练与评估

训练代码较长，具体参照 main_torch.py，直接看最后的效果，基本确定在第 15 轮之后开始过拟合，准确率基本在 75%，其实是比较低的。





一篇论文：

Hybrid_CNN-LSTM_With_Attention_Mechanism_for_Robust_Credit_Card_Fraud_Detection

下面的内容和这个论文下的 analysis.pdf 内容相同。

使用数据集：

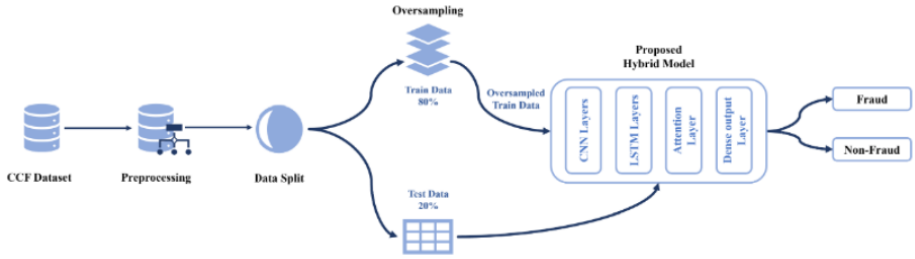
Credit_Card_Fraud_Detection

该数据集共有 284807 条交易记录，其中有 492 条为诈骗交易，数据分布高度不平衡，

诈骗记录占所有交易的 0.172%。

每条记录包含时间，金额，功能（V1-V28）和类，前三者是特征，其中时间是每个事务与数据集中第一个事务之间经过的秒数。最后的类是标签，在欺诈交易的情况下取 1，正常交易取 0。

论文神经网络训练流程：



可见就是一个二分类问题。数据预处理后，切分为训练集和测试集（应该还会从训练集切分出验证集），神经网络包含四个主要部分：卷积层，LSTM 层，注意力层和输出层。最后的输出是二分类。

数据集的处理：

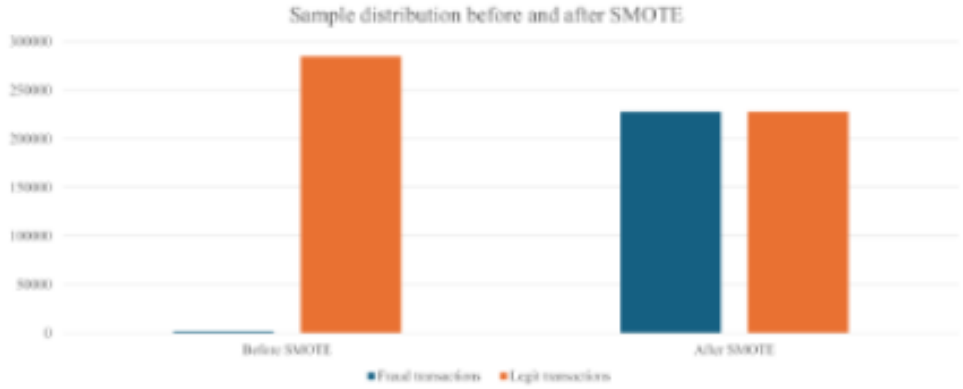
首先是数据的归一化。通常而言会使用减去平均值，除以标准差，将数据压缩到标准正态分布上，本篇论文使用了：

$$X_{\text{norm}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

这是另外一种缩放方式，可以将数据压缩到[0,1]，但是对于稀疏数据可能会破坏稀疏性。

处理类不平衡：SMOTE

在原先的数据集基础上使用 SMOTE 生成数据集，实现数据集的平衡，它通过在样本中划一条线，在沿线的某一点绘制新样本来合成数据。处理后样本分布：



具体参照：SMOTE: Synthetic Minority Over-sampling Technique

神经网络层：

由卷积神经网络层，LSTM 层和注意力层组合。卷积层使用 3*3 的窗口和 relu 激活函数，并使用 2*2，步幅为 2 的最大池化层缩小特征图。

LSTM 层：

处理长期依赖关系（时间特征），包含输入门，忘记门和输出门，在长序列中保留相关信息。

注意力层：

增强模型专注于输入序列重要部分的能力。