作业一：手写MLP，实战Kaggle比赛——预测房价

姓名	学号
张三	0000000000
李四	1111111111

要求：

完成以下notebook，Kaggle数据集的下载和处理代码已经给出，请同学们自行完成训练过程并上传Kaggle。作业提交 jupyter notebook 文件。

Kaggle的房价预测比赛数据集由Bart de Cock于2011年收集， 涵盖了2006-2010年期间亚利桑那州埃姆斯市的房价。 这个数据集是相当通用的，不会需要使用复杂模型架构。 它比哈里森和鲁宾菲尔德的波士顿房价 https://archive,ics,uci,edu/ml/machine-learning-databases/housing/housing,names (https://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.names) 数据集要大得多，也有更多的特征。

本节将详细介绍数据预处理、模型设计和超参数选择。

下载和缓存数据集

在整本书中，我们将下载不同的数据集，并训练和测试模型。 这里(**实现几个函数来方便下载数据**)。 首先，建立字典 DATA_HUB ， 它可以将数据集名称的字符串映射到数据集相关的二元组上， 这个二元组包含数据集的url和验证文件完整性的sha-1密钥。 所有类似的数据集都托管在地址为 DATA_URL 的站点上。

In [1]:

```
import hashlib
import os
import tarfile
import zipfile
import requests

#@save
DATA_HUB = dict()
DATA_URL = 'http://d2l-data.s3-accelerate.amazonaws.com/'
```

下面的 download 函数用来下载数据集， 将数据集缓存在本地目录（默认情况下为 ../data ）中， 并返回下载文件的名称。 如果缓存目录中已经存在此数据集文件，并且其sha-1与存储在 DATA_HUB 中的相匹配， 我们将使用缓存的文件，以避免重复的下载。

In [2]:

```python
def download(name, cache_dir=os.path.join('..', 'data')):  #@save
    """下载一个DATA_HUB中的文件，返回本地文件名"""
    assert name in DATA_HUB, f"{name} 不存在于 {DATA_HUB}"
    url, sha1_hash = DATA_HUB[name]
    os.makedirs(cache_dir, exist_ok=True)
    fname = os.path.join(cache_dir, url.split('/')[-1])
    if os.path.exists(fname):
        sha1 = hashlib.sha1()
        with open(fname, 'rb') as f:
            while True:
                data = f.read(1048576)
                if not data:
                    break
                sha1.update(data)
        if sha1.hexdigest() == sha1_hash:
            return fname  # 命中缓存
    print(f'正在从{url}下载{fname}...')
    r = requests.get(url, stream=True, verify=True)
    with open(fname, 'wb') as f:
        f.write(r.content)
    return fname
```

还需实现两个实用函数： 一个将下载并解压缩一个zip或tar文件， 另一个是将本书中使用的所有数据集从 DATA_HUB 下载到缓存目录中。

In [3]:

```python
def download_extract(name, folder=None):  #@save
    """下载并解压zip/tar文件"""
    fname = download(name)
    base_dir = os.path.dirname(fname)
    data_dir, ext = os.path.splitext(fname)
    if ext == '.zip':
        fp = zipfile.ZipFile(fname, 'r')
    elif ext in ('.tar', '.gz'):
        fp = tarfile.open(fname, 'r')
    else:
        assert False, '只有zip/tar文件可以被解压缩'
    fp.extractall(base_dir)
    return os.path.join(base_dir, folder) if folder else data_dir

def download_all():  #@save
    """下载DATA_HUB中的所有文件"""
    for name in DATA_HUB:
        download(name)
```

Kaggle

Kaggle (https://www.kaggle.com)是一个当今流行举办机器学习比赛的平台， 每场比赛都以至少一个数据集为中心。 许多比赛有赞助方，他们为获胜的解决方案提供奖金。 该平台帮助用户通过论坛和共享代码进行互动，促进协作和竞争。 虽然排行榜的追逐往往令人失去理智： 有些研究人员短视地专注于预处理步骤，而不是考虑基础性问题。 但一个客观的平台有巨大的价值：该平台促进了竞争方法之间的直接定量比较，以及代码共享。 这便于每个人都可以学习哪些方法起作用，哪些没有起作用。 如果你想参加Kaggle比赛，你首先需要注册一个账户。

在房价预测比赛页面， 你在"Data"选项卡下可以找到数据集。 你可以通过下面的网址提交预测，并查看排名：

> https://www.kaggle.com/c/house-prices-advanced-regression-techniques (https://www.kaggle.com/c/house-prices-advanced-regression-techniques)

访问和读取数据集

注意，竞赛数据分为训练集和测试集。 每条记录都包括房屋的属性值和属性，如街道类型、施工年份、屋顶类型、地下室状况等。 这些特征由各种数据类型组成。 例如，建筑年份由整数表示，屋顶类型由离散类别表示，其他特征由浮点数表示。 这就是现实让事情变得复杂的地方：例如，一些数据完全丢失了，缺失值被简单地标记为“NA”。 每套房子的价格只出现在训练集中（毕竟这是一场比赛）。 将希望划分训练集以创建验证集，但是在将预测结果上传到Kaggle之后， 只能在官方测试集中评估我们的模型。

开始之前，将**使用** `pandas` **读入并处理数据。** 因此，在继续操作之前，你需要确保已安装 `pandas` 。 幸运的是，如果你正在用Jupyter阅读该书，你可以在不离开笔记本的情况下安装 `pandas` 。

In [4]:

```
# 如果你没有安装pandas，请取消下一行的注释
# !pip install pandas

%matplotlib inline
import numpy as np
import pandas as pd
import torch
from torch import nn
from d2l import torch as d2l
```

为方便起见，我们可以使用上面定义的脚本下载并缓存Kaggle房屋数据集。

In [5]:

```
DATA_HUB['kaggle_house_train'] = (  #@save
    DATA_URL + 'kaggle_house_pred_train.csv',
    '585e9cc93e70b39160e7921475f9bcd7d31219ce')

DATA_HUB['kaggle_house_test'] = (  #@save
    DATA_URL + 'kaggle_house_pred_test.csv',
    'fa19780a7b011d9b009e8bff8e99922a8ee2eb90')
```

我们使用 `pandas` 分别加载包含训练数据和测试数据的两个CSV文件。

In [6]:

```
train_data = pd.read_csv(download('kaggle_house_train'))
test_data = pd.read_csv(download('kaggle_house_test'))
```

训练数据集包括1460个样本，每个样本80个特征和1个标签， 而测试数据集包含1459个样本，每个样本80个特征。

In [7]:

```
print(train_data.shape)
print(test_data.shape)
```

```
(1460, 81)
(1459, 80)
```

让我们看看[**前四个和最后两个特征，以及相应标签**]（房价）。

In [8]:

```
print(train_data.iloc[0:4, [0, 1, 2, 3, -3, -2, -1]])
print(train_data.tail())
```

```
   Id  MSSubClass MSZoning  LotFrontage SaleType SaleCondition  SalePrice
0   1          60       RL         65.0       WD        Normal     208500
1   2          20       RL         80.0       WD        Normal     181500
2   3          60       RL         68.0       WD        Normal     223500
3   4          70       RL         60.0       WD       Abnorml     140000
        Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
1455  1456          60       RL         62.0     7917   Pave   NaN      Reg
1456  1457          20       RL         85.0    13175   Pave   NaN      Reg
1457  1458          70       RL         66.0     9042   Pave   NaN      Reg
1458  1459          20       RL         68.0     9717   Pave   NaN      Reg
1459  1460          20       RL         75.0     9937   Pave   NaN      Reg

     LandContour Utilities  ... PoolArea PoolQC  Fence MiscFeature MiscVal  \
1455         Lvl    AllPub  ...        0    NaN    NaN         NaN       0
1456         Lvl    AllPub  ...        0    NaN  MnPrv         NaN       0
1457         Lvl    AllPub  ...        0    NaN  GdPrv        Shed    2500
1458         Lvl    AllPub  ...        0    NaN    NaN         NaN       0
1459         Lvl    AllPub  ...        0    NaN    NaN         NaN       0

     MoSold YrSold  SaleType  SaleCondition  SalePrice
1455      8   2007        WD         Normal     175000
1456      2   2010        WD         Normal     210000
1457      5   2010        WD         Normal     266500
1458      4   2010        WD         Normal     142125
1459      6   2008        WD         Normal     147500

[5 rows x 81 columns]
```

我们可以看到，(**在每个样本中，第一个特征是ID，**) 这有助于模型识别每个训练样本。 虽然这很方便，但它不携带任何用于预测的信息。 因此，在将数据提供给模型之前，(**我们将其从数据集中删除**)。

In [9]:

```
all_features = pd.concat((train_data.iloc[:, 1:-1], test_data.iloc[:, 1:]))
```

数据预处理

如上所述，我们有各种各样的数据类型。 在开始建模之前，我们需要对数据进行预处理。 首先，我们[**将所有缺失的值替换为相应特征的平均值。**]然后，为了将所有特征放在一个共同的尺度上， 我们(**通过将特征重新缩放到零均值和单位方差来标准化数据**)：

$$x \leftarrow \frac{x - \mu}{\sigma},$$

其中μ和σ分别表示均值和标准差。 现在，这些特征具有零均值和单位方差，即 $E[\frac{x-\mu}{\sigma}] = \frac{\mu-\mu}{\sigma} = 0$和 $E[(x-\mu)^2] = (\sigma^2 + \mu^2) - 2\mu^2 + \mu^2 = \sigma^2$。 直观地说，我们标准化数据有两个原因： 首先，它方便优化。 其次，因为我们不知道哪些特征是相关的， 所以我们不想让惩罚分配给一个特征的系数比分配给其他任何特征的系数更大。

In [10]:

```
# 若无法获得测试数据，则可根据训练数据计算均值和标准差
numeric_features = all_features.dtypes[all_features.dtypes != 'object'].index
all_features[numeric_features] = all_features[numeric_features].apply(
    lambda x: (x - x.mean()) / (x.std()))
# 在标准化数据之后，所有均值消失，因此我们可以将缺失值设置为0
all_features[numeric_features] = all_features[numeric_features].fillna(0)
```

接下来，我们[**处理离散值。**] 这包括诸如“MSZoning”之类的特征。 (**我们用独热编码替换它们**)， 方法与前面将多类别标签转换为向量的方式相同 （请参见 :numref: `subsec_classification-problem` ）。 例如，“MSZoning”包含值“RL”和“Rm”。 我们将创建两个新的指示器特征“MSZoning_RL”和“MSZoning_RM”，其值为0或1。 根据独热编码，如果“MSZoning”的原始值为“RL”， 则：“MSZoning_RL”为1，“MSZoning_RM”为0。 `pandas` 软件包会自动为我们实现这一点。

In [11]:

```
# “Dummy_na=True”将“na”（缺失值）视为有效的特征值，并为其创建指示符特征
all_features = pd.get_dummies(all_features, dummy_na=True)
all_features.shape
```

Out[11]:

```
(2919, 331)
```

你可以看到，此转换会将特征的总数量从79个增加到331个。 最后，通过 `values` 属性，我们可以 [**从 `pandas` 格式中提取NumPy格式，并将其转换为张量表示**]用于训练。

In [12]:

```
n_train = train_data.shape[0]
train_features = torch.tensor(all_features[:n_train].values, dtype=torch.float32)
test_features = torch.tensor(all_features[n_train:].values, dtype=torch.float32)
train_labels = torch.tensor(
    train_data.SalePrice.values.reshape(-1, 1), dtype=torch.float32)
```

训练（自行完成）

看到这里，请同学自己完成从**定义网络、定义损失函数、定义优化器到进行训练等一系列深度学习流水线**，尽量使用手写实现而不用PyTorch自带库（有加分）

提示(有助于效果提升，自行决定是否采用)

损失函数

房价就像股票价格一样，关心的是相对数量，而不是绝对数量。 因此，**更关心相对误差** $\frac{y-\hat{y}}{y}$， 而不是绝对误差 $y-\hat{y}$。 例如，如果在俄亥俄州农村地区估计一栋房子的价格时， 假设预测偏差了10万美元， 然而那里一栋典型的房子的价值是12.5万美元， 那么模型可能做得很糟糕。 另一方面，如果在加州豪宅区的预测出现同样的10万美元的偏差， （在那里，房价中位数超过400万美元） 这可能是一个不错的预测。

(**解决这个问题的一种方法是用价格预测的对数来衡量差异**)。 事实上，这也是比赛中官方用来评价提交质量的误差指标。 即将δ for $|\log y - \log \hat{y}| \leq \delta$ 转换为$e^{-\delta} \leq \frac{\hat{y}}{y} \leq e^{\delta}$。 这使得预测价格的对数与真实标签价格的对数之间出现以下均方根误差：

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\log y_i - \log \hat{y}_i\right)^2}.$$

使用K折交叉验证调参

K折交叉验证有助于模型选择和超参数调整。 首先需要定义一个函数，在K折交叉验证过程中返回第i折的数据。 具体地说，它选择第i个切片作为验证数据，其余部分作为训练数据。 注意，这并不是处理数据的最有效方法，如果数据集大得多，会有其他解决办法。

当在K折交叉验证中训练K次后，**计算训练和验证误差的平均值。**

找到一组调优的超参数可能需要时间，这取决于一个人优化了多少变量。 有了足够大的数据集和合理设置的超参数，K折交叉验证往往对多次测试具有相当的稳定性。 然而，如果尝试了不合理的超参数，可能会发现验证效果不再代表真正的误差。

请注意，有时一组超参数的训练误差可能非常低，但K折交叉验证的误差要高得多， 这表明模型过拟合了。 在整个训练过程中，你将希望监控训练误差和验证误差这两个数字。 较少的过拟合可能表明现有数据可以支撑一个更强大的模型， 较大的过拟合可能意味着可以通过正则化技术来获益。

提交你的Kaggle预测

将模型应用于测试集。 将预测保存在CSV文件中可以简化将结果上传到Kaggle的过程。

In [ ]:

```
def train_and_pred(train_features, test_feature, train_labels, test_data,
                   num_epochs, lr, weight_decay, batch_size):
    """
    在这里部署你的训练代码
    """

    # 将其重新格式化以导出到Kaggle
    # preds 是你对所有训练数据的预测结果，形状应该是 (1459, 1) 或类似形状的。（训练数据个数是14
    test_data['SalePrice'] = pd.Series(preds.reshape(1, -1)[0])
    submission = pd.concat([test_data['Id'], test_data['SalePrice']], axis=1)
    submission.to_csv('submission.csv', index=False)
```

执行上面的代码将生成一个名为 `submission.csv` 的文件。将这个文件提交到 Kaggle 即可得到测试结果。

In [ ]:

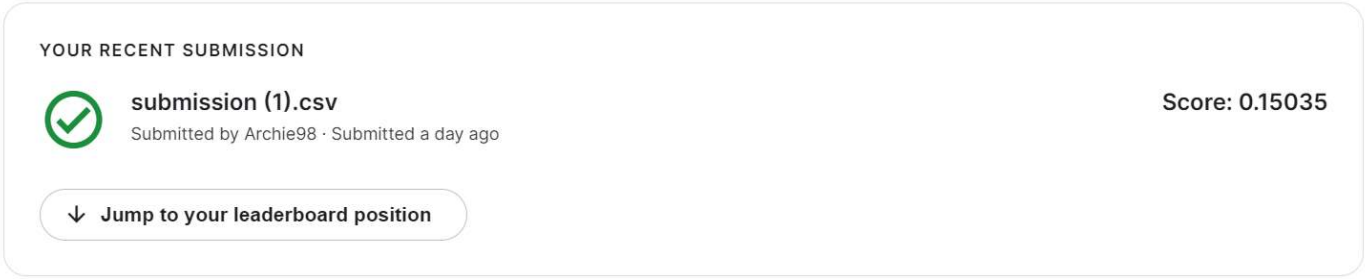
```
train_and_pred(train_features, test_features, train_labels, test_data,
               num_epochs, lr, weight_decay, batch_size)
```

提交预测到Kaggle上，并查看在测试集上的预测与实际房价（标签）的比较情况。 步骤非常简单：

- 登录Kaggle网站，访问房价预测竞赛页面。
- 点击“Submit Predictions”或“Late Submission”按钮（在撰写本文时，该按钮位于右侧）。
- 点击页面底部虚线框中的“Upload Submission File”按钮，选择你要上传的预测文件。
- 点击页面底部的“Make Submission”按钮，即可查看你的结果。

结果截图：

结果截图请放在这里：



截图以图片形式保存在同一文件夹内，与 jupyter notebook 一起压缩成zip文件，命名为 work1_<组长姓名>_<组长学号>.zip 发送到邮箱 archie98@qq.com (mailto:archie98@qq.com)