实验八：softmax回归模型

本次实验旨在让同学们了解并掌握训练softmax回归做多类别分类的流程，学习pytorch和torchvision的基本使用。

内容

1. softmax回归

1.1 分类问题

回归可以用于预测。 线性回归模型适用于输出为 *连续值* 的情景，比如预测房屋被售出的价格，或者棒球队可能获得的胜利数，又或者患者住院的天数。

但是在输出为 *离散值* 的情景中，比如预测分类，我们感兴趣的不是“多少”，而是“哪一个”：

- 该电子邮件是否属于垃圾邮件文件夹？
- 该用户可能注册或不注册订阅服务？
- 该图像描绘的是驴、狗、猫、还是鸡？
- 韩梅梅接下来最有可能看哪部电影？

对于这样的离散值预测问题，我们可以使用诸如softmax回归在内的分类模型。和线性回归不同，softmax回归的输出单元从一个变成了多个，且引入了softmax运算使输出更适合离散值的预测和训练。在神经网络的分类模型中，经常使用到softmax运算。

通常，*分类* 可以有两种问题： （1）我们只对样本的硬性类别感兴趣，即属于哪个类别；（2）我们希望得到软性类别，即得到属于每个类别的概率。 这两者的界限往往很模糊，其中的一个原因是，即使我们只关心硬类别，我们仍然需要使用软类别的模型。

1.2 softmax

softmax回归是一个多分类算法，我们的数据有多少个特征，则有多少个输入，有多少个类别，它就有多少个输出。在这里采取的主要方法是将模型的输出视作为概率。

假设每个样本有4个特征，3个输出类别。需要计算每个样本的3个输出值（o_1、o_2和o_3），我们将需要12个权值（w），3个偏置（b）：

$$
\begin{aligned}
o_1 &= x_1 w_{11} + x_2 w_{21} + x_3 w_{31} + x_4 w_{41} + b_1,\\
o_2 &= x_1 w_{12} + x_2 w_{22} + x_3 w_{32} + x_4 w_{42} + b_2,\\
o_3 &= x_1 w_{13} + x_2 w_{23} + x_3 w_{33} + x_4 w_{43} + b_3.
\end{aligned}
$$

softmax可以让输出的离散值归一化，压缩为0 ~ 1。它通过下式将输出值变换成值为大于0且总和为1的概率分布：

$$\hat{y}_1, \hat{y}_2, \hat{y}_3 = \mathrm{softmax}(o_1, o_2, o_3)$$

其中

$$\hat{y}_1 = \frac{\exp(o_1)}{\sum_{i=1}^{3} \exp(o_i)}, \quad \hat{y}_2 = \frac{\exp(o_2)}{\sum_{i=1}^{3} \exp(o_i)}, \quad \hat{y}_3 = \frac{\exp(o_3)}{\sum_{i=1}^{3} \exp(o_i)}.$$

容易看出$\hat{y}_1 + \hat{y}_2 + \hat{y}_3 = 1$且$0 \leq \hat{y}_1, \hat{y}_2, \hat{y}_3 \leq 1$。此外，我们注意到$\arg\max_i o_i = \arg\max_i \hat{y}_i$，即softmax运算不改变预测类别输出。

模型的输出 $\hat{y}_i$ 可以视为属于类 i 的概率。然后我们可以选择具有最大输出值的类别$\arg\max_i \hat{y}_i$作为我们的预测。

1.3 算法流程

与训练线性回归相比，你会发现训练softmax回归的步骤和它非常相似：获取并读取数据、定义模型和损失函数并使用优化算法训练模型。事实上，绝大多数深度学习模型的训练都有着类似的步骤。

步骤 0：初始化权重矩阵W和偏置值b。

步骤 1：对于每个类别和输入特征，计算得到其输出值，即对每个训练样本计算每个类别的得分：

$$\boldsymbol{O} = \boldsymbol{X} \cdot \boldsymbol{W}^T + \boldsymbol{b}$$

其中 $\boldsymbol{X}$ 是一个形状为 (m, n) 的矩阵，包含m个训练样本，每个训练样本有n个特征。$\boldsymbol{W}$ 是一个形状为 (K, n) 的权重向量矩阵，输出中有 K 个类别。

步骤 2：应用 softmax 将输出分数转换为概率：

$$\hat{\mathbf{Y}} = \operatorname{softmax}(\mathbf{O})$$

步骤 3：计算损失函数。我们希望模型对目标类别预测概率高，对其他类别预测概率低。这可以通过*交叉熵损失*（cross-entropy loss）函数实现，用于计算所有标签分布的预期损失值，它是分类问题最常用的损失之一。对于标签 y 和模型预测 $\hat{y}$，损失函数为：

$$L(y, \hat{y}) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} \left[y_k^{(i)} \log(\hat{y}_k^{(i)}) \right]$$

在这个公式中，目标标签y是*独热编码*，即如果 $\boldsymbol{x}^{(i)}$ 的目标类别是 k，则 $y_k^{(i)}$ 是 1，否则 $y_k^{(i)}$ 是 0。

ps：当只有两个类时，这个损失函数等同于逻辑回归的损失函数。

步骤 4：计算损失函数相对于每个权重向量和偏置的梯度。这个推导的详细解释可以在这里找到。

$$\nabla_{\boldsymbol{w}_k} L = \sum_{i=1}^{m} \boldsymbol{x}^{(i)} \left[\hat{y}_k^{(i)} - y_k^{(i)} \right]$$

$$\nabla_{\boldsymbol{b}_k} L = \sum_{i=1}^{m} \left[\hat{y}_k^{(i)} - y_k^{(i)} \right]$$

步骤 5：更新每个类别 k 的权重和偏置：

$$\boldsymbol{w}_k = \boldsymbol{w}_k - \eta \nabla_{\boldsymbol{w}_k} L$$

$$b_k = b_k - \eta \nabla_{b_k} L$$

其中 η 是学习率。

2. 图像分类

使用softmax回归实现多类图像分类。

2.1 PyTorch

PyTorch是一个由Facebook的人工智能研究团队开发的开源深度学习框架。它不仅是最受欢迎的深度学习框架之一，而且也是最强大的深度学习框架之一。官方手册

2.2 torchvision

这里我们将使用torchvision包，它是服务于PyTorch深度学习框架的，主要用来构建计算机视觉模型。torchvision主要由以下几部分构成：

- `torchvision.datasets`：一些加载数据的函数及常用的数据集接口；
- `torchvision.models`：包含常用的模型结构（含预训练模型），例如AlexNet、VGG、ResNet等；
- `torchvision.transforms`：常用的图片变换，例如裁剪、旋转等；
- `torchvision.utils`：其他的一些有用的方法。

建议激活conda环境后安装在环境中。 从Pytorch官网复制适合你操作系统的pytorch版本的下载指令：

In [ ]:
```
conda install pytorch torchvision -c pytorch
```

验证是否安装成功：

In [ ]:
```
import torch
import torchvision
```

装完成后，进入到python里，输入：import torch 和 import torchvision 没有任何提示则说明安装成功。

2.3 图像分类数据集（Fashion-MNIST）

Fashion-MNIST是一个10类别服饰分类数据集。 图像分类数据集中最常用的是手写数字识别数据集MNIST，但作为基准数据集过于简单。这里我们使用一个图像内容更加复杂的Fashion-MNIST，其中，每个类别由训练数据集中的 6000 张图像和测试数据集中的 1000 张图像组成（即训练集和测试集分别包含 60000 和 10000 张图像）。测试数据集不会用于训练，只用于评估模型性能。

我们可以通过torchvision的 `torchvision.datasets` 来下载这个数据集。第一次调用时会自动从网上获取数据。我们通过参数 `train` 来指定获取训练数据集或测试数据集（testing data set）。测试数据集也叫测试集（testing set），只用来评价模型的表现，并不用来训练模型。

另外我们还指定了参数 `transform = transforms.ToTensor()` 使所有数据转换为 `Tensor` ，如果不进行转换则返回的是PIL图片。 `transforms.ToTensor()` 将尺寸为 (H x W x C) 且数据位于[0, 255]的PIL图片或者数据类型为 `np.uint8` 的NumPy数组转换为尺寸为(C x H x W)且数据类型为 `torch.float32` 且位于[0.0, 1.0]的 `Tensor` 。

> 注意： 由于像素值是0到255的整数，所以刚好是uint8所能表示的范围，包括 `transforms.ToTensor()` 在内的一些关于图片的函数就默认输入的是uint8型，若不是，可能不会报错但是可能得不到想要的结果。所以，**如果用像素值(0-255整数)表示图片数据，那么一律将其类型设置成uint8，避免不必要的bug。**

In [2]:
```
import torch
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import sys
sys.path.append("..")

# 加载数据集
mnist_train = torchvision.datasets.FashionMNIST(root='~/Datasets/FashionMNIST', train=True, download=True, transform=transform
mnist_test = torchvision.datasets.FashionMNIST(root='~/Datasets/FashionMNIST', train=False, download=True, transform=transform
```

In [2]:
```
print(type(mnist_train))
print(len(mnist_train), len(mnist_test))
```

```
<class 'torchvision.datasets.mnist.FashionMNIST'>
60000 10000
```

In [3]:
```
#访问样本
feature, label = mnist_train[0]
```

```
print(feature.shape, label)  # Channel x Height X Width
```

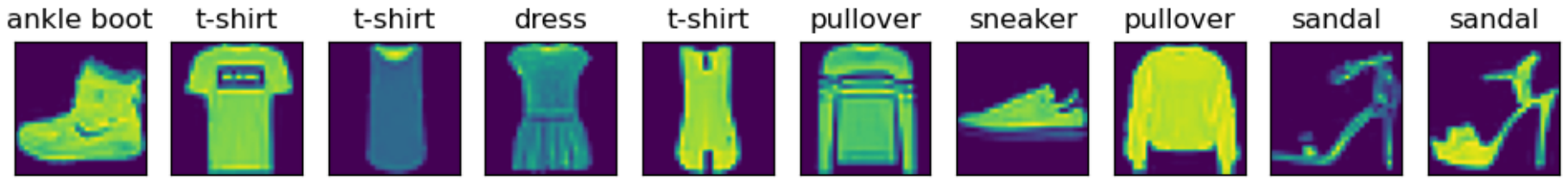
```
torch.Size([1, 28, 28]) 9
```

变量 `feature` 对应高和宽均为28像素的图像。由于我们使用了 `transforms.ToTensor()` ，所以每个像素的数值为[0.0, 1.0]的32位浮点数。需要注意的是， `feature` 的尺寸是 (C x H x W) 的，而不是 (H x W x C)。第一维是通道数，因为数据集中是灰度图像，所以通道数为1（若是RGB图像，则通道数为3）。后面两维分别是图像的高和宽。

In [3]:
```
def get_fashion_mnist_labels(labels):
    """返回Fashion-MNIST数据集的文本标签。"""
    text_labels = ['t-shirt', 'trouser', 'pullover', 'dress', 'coat',
                   'sandal', 'shirt', 'sneaker', 'bag', 'ankle boot']
    return [text_labels[int(i)] for i in labels]


def show_fashion_mnist(images, labels):
    _, figs = plt.subplots(1, len(images), figsize=(12, 12))
    for f, img, lbl in zip(figs, images, labels):
        f.imshow(img.view((28, 28)).numpy())
        f.set_title(lbl)
        f.axes.get_xaxis().set_visible(False)
        f.axes.get_yaxis().set_visible(False)
    plt.show()

# 展示图像及其标签
X, y = [], []
for i in range(10):
    X.append(mnist_train[i][0])
    y.append(mnist_train[i][1])
show_fashion_mnist(X, get_fashion_mnist_labels(y))
```



MNIST图片是28×28的二维图像，为了进行计算，我们将其转化为784维向量，即$X = (x_0, x_1, \ldots, x_{783})$。这里通过`view`函数将每张原始图像改成长度为`num_inputs`的向量。

In [9]:
```
num_inputs = 784
X = feature.view(-1, num_inputs)
print(X.shape)
```

```
torch.Size([1, 784])
```

2.4 小批量样本

我们将在训练数据集上训练模型，并将训练好的模型在测试数据集上评价模型的表现。为了提高计算效率并且充分利用计算资源，我们通常会对训练数据集分成小批量数据（mini batch）进行训练。

`mnist_train`是`torch.utils.data.Dataset`的子类，所以我们可以将其传入`torch.utils.data.DataLoader`来创建一个读取小批量数据样本的DataLoader实例。

在实践中，数据读取经常是训练的性能瓶颈，特别当模型较简单或者计算硬件性能较高时。PyTorch的`DataLoader`中一个很方便的功能是允许使用多进程来加速数据读取，可以通过参数`num_workers`来设置n个进程读取数据。

In [ ]:
```
import torch.utils.data as Data

batch_size =256
num_workers = 4
dataloader = Data.DataLoader(
    dataset=mnist_train,      # 数据集
    batch_size=batch_size,      # mini batch 的大小
    shuffle=True,               # 要不要打乱数据 (一般train时打乱，test时不打乱)
    num_workers=num_workers,              # 多线程来读数据
    drop_last=False,  #如果数据集大小不能被批次大小整除，是否丢弃最后一个不完整的批次。默认为False
)
```

之后就可以使用小批量随机梯度下降来优化模型的损失函数。 在训练模型时，迭代周期数`num_epochs`和学习率`lr`都是可以调的超参数，改变它们的值可能会得到分类更准确的模型。

In [ ]:
```
def train(train_dataloader,num_epochs,lr=0.1,):
    for epoch in range(1,num_epochs+1):
        # 对小批量样品进行训练
        for X,y in train_dataloader:
            # X是特征，y是标签
```

3. 动手实践

- 详见实验作业