加载数据集

笔记本:	pytorch		
创建时间:	2023/3/28 11:47	**更新时间:**	2023/3/28 15:28
作者:	22qdnlbn		
URL:	about:blank		

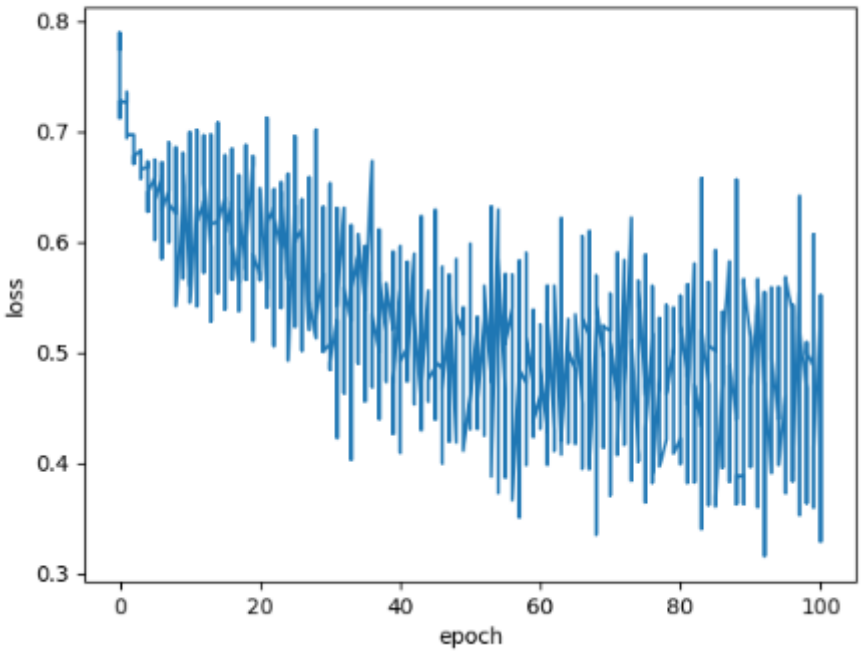
1、加载数据集
①我们可以一次加载完所有的数据集，但是这种情况是当我们的数据集样本数量和特证数量都不多时。
②如果我们加载的是图像数据集或者语音数据集的话，我们可以采用将数据样本的文件名加载到内存中，当我们需要该数据样本时，根据文件名到硬盘中去读取即可。

2、理解术语
epoch：epoch是我们所有的数据样本都进行了前馈和反馈的过程
iteration：iteration是epoch中的一个迭代过程，与bath-size一起用。如：当一个数据集有1000个样本时，batch-size为100，那么iteration就为10。
batch-size: 每次迭代时的样本数量

3、根据代码讲解：

```
# 时间：2023/3/28 10:37
# cky
import numpy as np
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
#Dataset是一个抽象类，不能实例化，必须继承之后再实例化
#Dataset的 len，getitem 是魔法函数 必须重载
class MyDataset(Dataset):
    def __init__(self,filepath):
        xy=np.loadtxt(filepath,delimiter=',',dtype=np.float32) #是numpy类型
        self.x=torch.from_numpy(xy[:,:-1]) #转换为tensor类
        self.y=torch.from_numpy(xy[:,[-1]])
        self.len=xy.shape[0]
    def __len__(self):
        return self.len
    def __getitem__(self, item):
        return self.x[item],self.y[item]
dataset=MyDataset('diabetes.csv.gz')
train_data=DataLoader(dataset=dataset,shuffle=True,batch_size=64,num_workers=0)
class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.linear1=torch.nn.Linear(8,6)
        self.linear2=torch.nn.Linear(6,2)
        self.linear3=torch.nn.Linear(2,1)
        self.sigmoid=torch.nn.Sigmoid()
        self.relu6=torch.nn.ReLU6()
    def forward(self,x):
        x=self.relu6(self.linear1(x))
        x=self.relu6(self.linear2(x))
        x=self.sigmoid(self.linear3(x))
        return x
model=Model()
```

```
loss=torch.nn.BCELoss(reduction='mean')
optim=torch.optim.SGD(params=model.parameters(),lr=0.05)
epoch_list=[]
loss_list=[]
#training cycle
for ephch in range(300):
    for i,(inputs,outputs) in enumerate(train_data,1):
        epoch_list.append(ephch)
        y_hat=model(inputs)
        cost=loss(y_hat,outputs)
        loss_list.append(cost.item())
        print(ephch,i,cost.item())
        optim.zero_grad()
        cost.backward()
        optim.step()
plt.plot(epoch_list,loss_list)
plt.ylabel('loss')
plt.xlabel('epoch')
plt.show()
```



注：由于是随机梯度下降，会有噪音的影响，如果是所有样本集的话，噪声很小：是单个样本的噪声影响不大。但是当使用mini-batch时，单个样本集的噪声就会变大，所以就导致我们的损失函数可能并不能总是朝着最优解的方向走，但是大体趋势是朝着最优解方向去的，使用随机梯度下降，我们不会找到最优解，会在最优解附近徘徊。

①prepare dataset

```
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
#Dataset是一个抽象类，不能实例化，必须继承之后再实例化
#Dataset的 len，getitem 是魔法函数 必须重载
class MyDataset(Dataset):
    def __init__(self,filepath):
        xy=np.loadtxt(filepath,delimiter=',',dtype=np.float32) #是numpy类型
        self.x=torch.from_numpy(xy[:,:-1]) #转换为tensor类
        self.y=torch.from_numpy(xy[:,[-1]])
        self.len=xy.shape[0]
```

```
    def __len__(self):
        return self.len
    def __getitem__(self, item):
        return self.x[item],self.y[item]
dataset=MyDataset('diabetes.csv.gz')
train_data=DataLoader(dataset=dataset,shuffle=True,batch_size=64,num_workers=0)
```

#DataLoader 是pytorch提供的数据加载器 我们初始化时要给四个变量赋值
dataset是我们使用的数据集，shuffer即是否要打乱，batch-size与机器硬件相吻合 我们都是采取8的倍数
num_workers 即使用几个线程来加载。

DataSet都要支持下标操作，如_getitem 就是根据下标来获得值。

②training cycle

```
for ephch in range(300):
    for i,(inputs,outputs) in enumerate(train_data,1):
或者for i,data in enumerate(train_data,1):  inputs,outputs=data
        epoch_list.append(ephch)
        y_hat=model(inputs)
```

enumerate多用于在for循环中得到计数，利用它可以同时获得索引和值，默认是从0开始，本例中我们从1开始了。

train_data 是我们之前实例化的一个对象，其是根据我们的mini-batch-size来进行迭代。由于我们在加载数据时，已经分为了x,y，所以train_data中的也是分为x和y的。**另外 dataloader会自动把我们加载的数据转变为Tenshor类型，所以就不需要我们在手动去转换了。**

4、其他数据集

torch也为我们提供了很多其他数据集，torchvision.datasets 中就包含了许多其他数据集。该库中的数据集都是继承自Torch.utils.data.Dataset. 都已经实现了len和getitem这些magic function

5、对于3的代码，自己又加了点计算准确率的

```
# 时间：2023/3/28 10:37
# cky
import numpy as np
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
#Dataset是一个抽象类，不能实例化，必须继承之后再实例化
#Dataset的 len，getitem 是魔法函数 必须重载
class MyDataset(Dataset):
    def __init__(self,filepath):
        xy=np.loadtxt(filepath,delimiter=',',dtype=np.float32) #是numpy类型
        self.x=torch.from_numpy(xy[:,:-1]) #转换为tensor类
        self.y=torch.from_numpy(xy[:,[-1]])
        self.len=xy.shape[0]
    def __len__(self):
        return self.len
    def __getitem__(self, item):
        return self.x[item],self.y[item]
dataset=MyDataset('diabetes.csv.gz')
train_data=DataLoader(dataset=dataset,shuffle=True,batch_size=64,num_workers=0)
class Model(nn.Module):
```

```
    def __init__(self):
        super(Model, self).__init__()
        self.linear1=torch.nn.Linear(8,6)
        self.linear2=torch.nn.Linear(6,2)
        self.linear3=torch.nn.Linear(2,1)
        self.sigmoid=torch.nn.Sigmoid()
        self.relu6=torch.nn.ReLU6()
    def forward(self,x):
        x=self.relu6(self.linear1(x))
        x=self.relu6(self.linear2(x))
        x=self.sigmoid(self.linear3(x))
        return x
model=Model()


loss=torch.nn.BCELoss(reduction='mean')
optim=torch.optim.SGD(params=model.parameters(),lr=0.05)
epoch_list=[]
loss_list=[]
#training cycle


for ephch in range(101):
    csum = 0
    for i,(inputs,outputs) in enumerate(train_data,1):
        epoch_list.append(ephch)
        y_hat=model(inputs)
        for y1,y2 in zip(y_hat,outputs):
            if round(y1.item())==round(y2.item()):
                csum+=1
        cost=loss(y_hat,outputs)
        loss_list.append(cost.item())
        optim.zero_grad()
        cost.backward()
        optim.step()
    print(ephch,str(csum/len(dataset)*100)+'%')
plt.plot(epoch_list,loss_list)
plt.ylabel('loss')
plt.xlabel('epoch')
plt.show()
```