交叉验证及简单代码学习

Reviewed

K折交叉验证的过程如下

以200条数据，十折交叉验证为例子，十折也就是将数据分成10组，进行10组训练，每组用于测试的数据为：数据总条数/组数，即每组20条用于valid，180条用于train，每次valid的都是不同的。

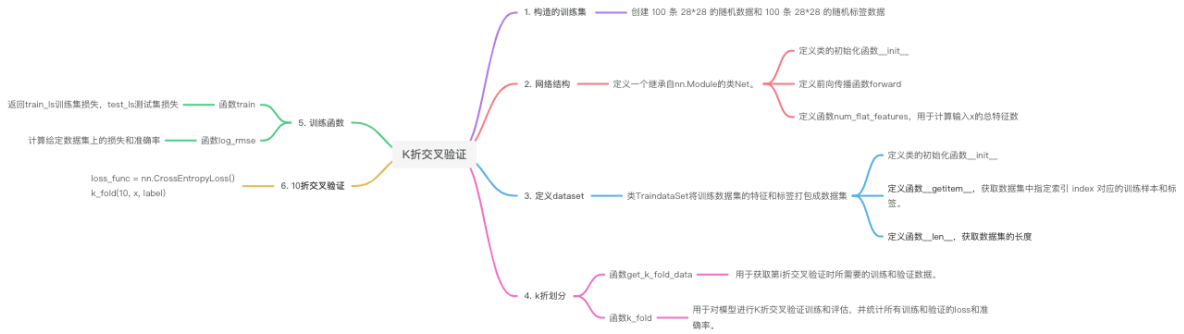
（1）将200条数据，分成按照 数据总条数/组数（折数），进行切分。然后取出第i份作为第i次的valid，剩下的作为train

（2）将每组中的train数据利用DataLoader和Dataset，进行封装。

（3）将train数据用于训练，epoch可以自己定义，然后利用valid做验证。得到一次的train_loss和 valid_loss。

（4）重复（2）（3）步骤，得到最终的 averge_train_loss和averge_valid_loss

K折交叉验证思维导图



K折交叉验证程序及注释

```
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, Dataset
import torch.nn.functional as F
from torch.autograd import Variable

#####构造的训练集####
# 创建 100 条 28*28 的随机数据和 100 条 28*28 的随机标签数据
x = torch.rand(100, 28, 28)
y = torch.randn(100, 28, 28)
x = torch.cat((x, y), dim=0)  # 将数据和标签按照行的维度连接起来
label = [1] * 100 + [0] * 100  # 创建了一个包含200个元素的列表label，前100个元素的值为1，后100个元素的值为0
label = torch.tensor(label, dtype=torch.long)  # 将列表label转换为张量（tensor），并将数据类型设置为整数类型。


######网络结构##########
class Net(nn.Module):  # 定义一个继承自nn.Module的类，类名为Net。
    # 定义Net
```

```
    def __init__(self):
        '''
        定义类的初始化函数__init__
        '''
        super(Net, self).__init__()  # 调用了父类的__init__()方法

        # 定义了三个线性层（全连接层）
        self.fc1 = nn.Linear(28 * 28, 120)  # 输入层到隐藏层的全连接层
        self.fc2 = nn.Linear(120, 84)  # 隐藏层到隐藏层的全连接层
        self.fc3 = nn.Linear(84, 2)  # 隐藏层到输出层的全连接层

    def forward(self, x):
        '''
        定义前向传播函数forward
        :param x: 模型的输入，大小(batch_size, 1, 28, 28)
        :return: 神经网络模型的预测结果，二维张量(batch_size, 2)
        '''
        x = x.view(-1, self.num_flat_features(x))  # 将x进行展平(batch_size, 1, 28, 28)-->(batch_size, 28 * 28)，即输入数据从二维矩阵转换为一维向量

        x = F.relu(self.fc1(x))  # 对第一层全连接层的输出进行ReLU激活函数处理
        x = F.relu(self.fc2(x))  # 对第二层全连接层的输出进行ReLU激活函数处理
        x = self.fc3(x)  # 最后一层输出层不进行激活函数处理
        return x  # 返回输出值（神经网络模型的预测结果）

    def num_flat_features(self, x):
        '''
        定义函数num_flat_features，用于计算输入x的总特征数
        :param x: 模型的输入，大小为(batch_size, 1, 28, 28)
        :return: x的总特征数
        '''
        size = x.size()[1:]  # 获取x的大小（不包括batch维度）
        num_features = 1
        # 计算x的总特征数
        for s in size:
            num_features *= s
        return num_features  # 返回输出值（x的总特征数）


##########定义dataset##########
class TraindataSet(Dataset):
    '''
    将训练数据集的特征和标签打包成数据集，方便进行训练。
    '''
    def __init__(self, train_features, train_labels):
        '''
        类的初始化方法
        :param train_features: 训练数据集的特征
        :param train_labels: 训练数据集的标签
        '''
        # 将传入的训练特征和训练标签赋值给对象的属性 x_data 和 y_data
        self.x_data = train_features
        self.y_data = train_labels
        self.len = len(train_labels)  # 获取数据集中样本数量

    def __getitem__(self, index):
        '''
        获取数据集中指定索引 index 对应的训练样本和标签。
        :param index: 欲获取指定数据的索引
        :return: 给定索引 index 对应的训练样本和标签
        '''
        return self.x_data[index], self.y_data[index]

    def __len__(self):
        '''
        获取数据集的长度
        :return: 数据集的长度
        '''
        return self.len


########k折划分############
def get_k_fold_data(k, i, X, y):  ###此过程主要是步骤（1）
    '''
    用于获取第i折交叉验证时所需要的训练和验证数据。
    :param k: 表示交叉验证的折数。
    :param i: 表示当前取第几折作为验证集。
    :param X: 表示输入的数据集。
    :param y: 表示输入的标签。
    :return: 返回第i折交叉验证时所需要的训练和验证数据：
    X_train：用于训练的数据特征
    y_train：用于训练的数据标签
```

```
    X_valid：用于验证的数据特征
    y_valid：用于验证的数据标签
    '''

    assert k > 1  # 确保折数大于1
    fold_size = X.shape[0] // k  #  每折包含的数据个数:数据总条数/折数（组数）

    X_train, y_train = None, None  # 初始化训练数据和标签

    for j in range(k):  # j 表示当前数据集的组数（第几折），fold_size 表示每个数据集的大小
        idx = slice(j * fold_size, (j + 1) * fold_size)  # 生成每折数据的索引
        # 对数据进行切片操作。切片的起始位置是 j * fold_size，结束位置是 (j + 1) * fold_size，步长默认为1

        # 从输入数据中取出当前折的数据
        X_part, y_part = X[idx, :], y[idx]  # X[idx, :]：X数据的第 j 份（折）的特征数据，y[idx]：第 j 份的标签数据。

        if j == i:  # 第i折做验证集
            X_valid, y_valid = X_part, y_part  # 当前折的数据设为验证数据

        # 将不是第i折的数据作为训练数据集
        elif X_train is None:  # 在第一次进入该else语句时，X_train和y_train都是None，所以将X_part和y_part赋值给它们
            X_train, y_train = X_part, y_part
        else:  # 在之后的每次循环中，都将X_part和y_part沿着dim=0竖直地拼接到X_train和y_train末尾。
            X_train = torch.cat((X_train, X_part), dim=0)  # dim=0增加行数，竖着连接（dim=0 表示在第0维（即行）上进行连接）
            y_train = torch.cat((y_train, y_part), dim=0)

    # print(X_train.size(),X_valid.size())
    return X_train, y_train, X_valid, y_valid


def k_fold(k, X_train, y_train, num_epochs=3, learning_rate=0.001, weight_decay=0.1, batch_size=5):
    '''
    用于对模型进行K折交叉验证训练和评估，并统计所有训练和验证的loss和准确率。
    :param k: 折数
    :param X_train: 训练集的特征矩阵
    :param y_train: 训练集的标签
    :param num_epochs: 迭代次数，默认值为3
    :param learning_rate: 学习率，默认值为0.001
    :param weight_decay: 权重衰减参数，默认值为0.1
    :param batch_size: 每次迭代的样本数，默认值为5
    :return: 无，直接打印输出结果
    train_loss_sum：所有折的训练损失之和。
    valid_loss_sum：所有折的验证损失之和。
    train_acc_sum：所有折的训练精度之和。
    valid_acc_sum：所有折的验证精度之和。
    '''
    train_loss_sum, valid_loss_sum = 0, 0  # 所有训练集和验证集的loss之和
    train_acc_sum, valid_acc_sum = 0, 0  # 所有训练集和验证集的准确率之和

    # 循环遍历k份数据集
    for i in range(k):
        data = get_k_fold_data(k, i, X_train, y_train)  # 获取k折交叉验证的训练和验证数据
        net = Net()  # 实例化模型
        # 模型训练，返回训练集和验证集的loss和准确率
        train_ls, valid_ls = train(net, *data, num_epochs, learning_rate, weight_decay, batch_size)

        print('*' * 25, '第', i + 1, '折', '*' * 25)  # 输出当前是第几折交叉验证
        print('train_loss:%.6f' % train_ls[-1][0], 'train_acc:%.4f\n' % valid_ls[-1][1], \
              'valid loss:%.6f' % valid_ls[-1][0], 'valid_acc:%.4f' % valid_ls[-1][1])  # 输出当前折的训练损失(train_loss)、训练精度(train_acc)、
        # [-1]表示取列表中最后一个元素，[0]表示取损失，[1]表示取精度。
        train_loss_sum += train_ls[-1][0]  # 将当前折的训练损失累加到训练损失总和中。
        valid_loss_sum += valid_ls[-1][0]  # 将当前折的验证损失累加到验证损失总和中。
        train_acc_sum += train_ls[-1][1]  # 将当前折的训练精度累加到训练精度总和中。
        valid_acc_sum += valid_ls[-1][1]  # 将当前折的验证精度累加到验证精度总和中。
    print('#' * 10, '最终k折交叉验证结果', '#' * 10)
    print('train_loss_sum:%.4f' % (train_loss_sum / k), 'train_acc_sum:%.4f\n' % (train_acc_sum / k), \
          'valid_loss_sum:%.4f' % (valid_loss_sum / k), 'valid_acc_sum:%.4f' % (valid_acc_sum / k))



#########训练函数##########
def train(net, train_features, train_labels, test_features, test_labels, num_epochs, learning_rate, weight_decay,
          batch_size):
    '''
    :param net: 神经网络模型
    :param train_features: 训练数据的特征
    :param train_labels: 训练数据的标签
    :param test_features: 测试数据的特征
    :param test_labels: 测试数据的标签
    :param num_epochs: 训练轮数
    :param learning_rate: 学习率
```

```
    :param weight_decay: 权重衰减
    :param batch_size: 每个批次的数据量
    :return: train_ls训练集损失，test_ls测试集损失
    '''

    train_ls, test_ls = [], []  # 定义空列表，存储train_loss,test_loss
    dataset = TraindataSet(train_features, train_labels)  # 将训练数据集的特征和标签打包成数据集
    train_iter = DataLoader(dataset, batch_size, shuffle=True)  # 构建数据迭代器
    ### 将数据封装成 Dataloder 对应步骤（2）将每组中的train数据利用DataLoader和Dataset，进行封装。

    # 定义优化器使用Adam算法，将学习率、权重衰减作为参数传递给优化器
    optimizer = torch.optim.Adam(params=net.parameters(), lr=learning_rate, weight_decay=weight_decay)

    for epoch in range(num_epochs):
        # 对于每个epoch，将数据集分成一个个batch，对每个batch进行训练
        for X, y in train_iter:
            output = net(X)  # 将特征数据输入网络，得到输出
            loss = loss_func(output, y)  # 计算预测结果和真实结果的损失
            optimizer.zero_grad()  # 梯度清零，防止累计梯度导致梯度爆炸
            loss.backward()  # 反向传播，计算每个参数对损失函数的梯度
            optimizer.step()  # 更新参数，根据优化算法更新参数

        # 计算每个epoch的训练集和测试集的损失函数值，将其添加到train_ls和test_ls列表中
        train_ls.append(log_rmse(0, net, train_features, train_labels))
        if test_labels is not None:
            test_ls.append(log_rmse(1, net, test_features, test_labels))
    # print(train_ls,test_ls)
    return train_ls, test_ls  # 返回train_ls和test_ls列表


def log_rmse(flag, net, x, y):
    '''
    计算给定数据集上的损失和准确率
    :param flag: 整数类型，表示当前数据集是训练集还是测试集，1 表示测试集，0 表示训练集；
    :param net: 神经网络模型；
    :param x: 输入特征；
    :param y: 标签。
    :return: 损失和准确率
    '''

    if flag == 1:  # 当 flag=1 时表示验证集，flag=0 时表示训练集
        net.eval()  # 切换到评估模式，这会关闭 Dropout 和 BatchNorm
    output = net(x)  # 前向传播，计算预测值
    result = torch.max(output, 1)[1].view(y.size())  # 找到每个样本预测值最大的下标
    '''
    torch.max(output, 1) 表示在 output 张量的第 1 维上取最大值，返回一个元组 (values, indices)，其中 values 表示最大值，indices 表示最大值的下标。
    通过 [1] 取出下标张量
    使用 .view(y.size()) 来将其形状调整为与真实标签 y 相同的形状
    '''
    corrects = (result.data == y.data).sum().item()  # 计算预测正确的样本数
    accuracy = corrects * 100.0 / len(y)  # 计算预测准确率
    loss = loss_func(output, y)  # 计算损失
    net.train()  # 切换回训练模式

    return (loss.data.item(), accuracy)  # 返回损失和准确率


loss_func = nn.CrossEntropyLoss()  # 申明loss函数
k_fold(10, x, label)  # k=10,十折交叉验证
```

十折交叉验证结果：

- ********** 第 1 折 **********
  train_loss:0.041583 train_acc:100.0000
  valid loss:0.025219 valid_acc:100.0000
  ********** 第 2 折 **********
  train_loss:0.037519 train_acc:100.0000
  valid loss:0.026306 valid_acc:100.0000
  ********** 第 3 折 **********
  train_loss:0.041386 train_acc:100.0000
  valid loss:0.026438 valid_acc:100.0000
  ********** 第 4 折 **********

train_loss:0.040735 train_acc:100.0000
valid loss:0.024389 valid_acc:100.0000
************************* 第 5 折 *************************
train_loss:0.039734 train_acc:100.0000
valid loss:0.024530 valid_acc:100.0000
************************* 第 6 折 *************************
train_loss:0.037305 train_acc:90.0000
valid loss:0.400836 valid_acc:90.0000
************************* 第 7 折 *************************
train_loss:0.041748 train_acc:95.0000
valid loss:0.379797 valid_acc:95.0000
************************* 第 8 折 *************************
train_loss:0.043323 train_acc:90.0000
valid loss:0.429456 valid_acc:90.0000
************************* 第 9 折 *************************
train_loss:0.041039 train_acc:100.0000
valid loss:0.345698 valid_acc:100.0000
************************* 第 10 折 *************************
train_loss:0.038138 train_acc:100.0000
valid loss:0.330852 valid_acc:100.0000
########## 最终k折交叉验证结果 ##########
train_loss_sum:0.0403 train_acc_sum:100.0000
valid_loss_sum:0.2014 valid_acc_sum:97.5000