基于神经网络的CIFAR-10图像分类模型

19335015 陈恩婷

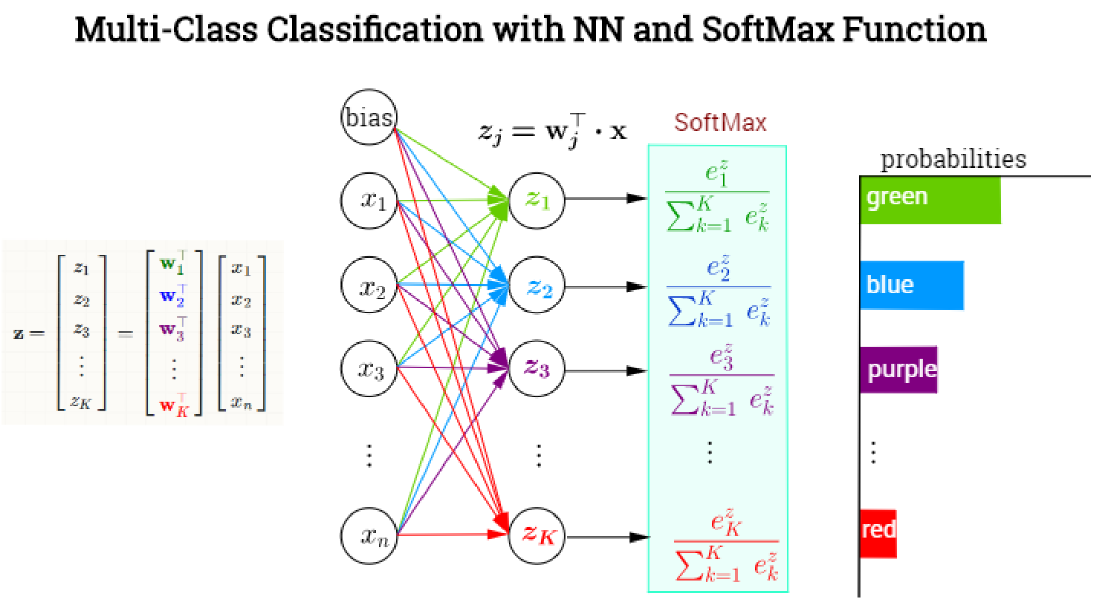
模型结果与训练方法

线性分类器（Softmax分类器）

根据ufldl.stanford.edu上的介绍:

> Softmax regression (or multinomial logistic regression) is a generalization of logistic regression to the case where we want to handle multiple classes.

所以，我们可以通过推广之前学习过的二元逻辑回归的方式，来实现一个Softmax分类器。Softmax分类器的大致结构和流程如下（图片来源）：



数据预处理

实验所需的CIFAR-10数据集存放在C:/Users/豹豹/OneDrive - 中山大学/大三下/机器学习与数据挖掘/Assignment2/data路径下。参考实验要求内的代码，可以通过以下函数读入数据集:

```
import pickle
import numpy as np s

dir = r'C:/Users/豹豹/OneDrive - 中山大学/大三下/机器学习与数据挖掘/Assignment2/data'

def load_data():
    X_train = []
    Y_train = []
    for i in range(1, 6):
        with open(dir + r'/data_batch_' + str(i), 'rb') as fo:
            dict = pickle.load(fo, encoding='bytes')
            X_train.append(dict[b'data'])
            Y_train += dict[b'labels']
    X_train = np.concatenate(X_train, axis=0)
    with open(dir + r'/test_batch', 'rb') as fo:
```

```
        dict = pickle.load(fo, encoding='bytes')
        X_test = dict[b'data']
        Y_test = dict[b'labels']

    X_train = np.array(X_train, dtype=np.float32).T
    X_test = np.array(X_test, dtype=np.float32).T
    Y_train = np.array(Y_train, dtype=np.int32)
    Y_test = np.array(Y_test, dtype=np.int32)

    return X_train, Y_train, X_test, Y_test
```

执行完以上函数后，X_train, Y_train, X_test, Y_test的大小形状如下：

```
(3072, 50000) (50000,) (3072, 10000) (10000,)
```

模型参数初始化

除超参数外，Softmax分类器的模型只需要用到W参数。其初始化方法如下：

```
# Random initialization of W
self.W = np.random.randn(10, 3072) * 0.0001
```

超参数的设置如下：

```
lr=1e-5, reg=1e-3, num_iters=2000, batch_size=200
```

优化方法

L2正则化

引自互联网的解释：

> L2正则化就是loss function后边所加正则项为L2范数的平方，加上L2正则相比于L1正则来说，得到的解比较平滑（不是稀疏），但是同样能够保证解中接近于0（但不是等于0，所以相对平滑）的维度比较多，降低模型的复杂度。

代码如下：

```
# Calculate the loss
# print(scores.shape, y.shape)
loss = -np.sum(np.log(scores[y, np.arange(len(y))]))
loss /= x.shape[1]
loss += reg * np.sum(self.W * self.W)

# Calculate the gradient
scores[y, range(scores.shape[1])] -= 1
grad = scores.dot(x.T)
grad /= x.shape[1]
grad += 2 * reg * self.W
```

多层感知机（MLP）

根据Lecture6-Neural Networks中的介绍，多层感知机（MLP，multiplayer perception)就是全连接神经网络（fully connected Neural Network）。按照一般的PyTorch神经网络设计方法可以很方便实现一个简单的MLP。本实验的MLP结构如下：

```
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(3072, 1000)
        self.fc2 = nn.Linear(1000, 500)
        self.fc3 = nn.Linear(500, 10)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

数据预处理

由于实现MLP需要用到PyTorch，所以需要在读入数据集并转为numpy array的基础上，把数据转化为torch tensor类型。代码如下：

```
# convert np array to torch tensor, Y to long tensor, then to one-hot
def np_to_tensor(X_train, Y_train, X_test, Y_test):
    X_train = torch.from_numpy(X_train)
    Y_train = torch.from_numpy(Y_train).long()
    X_test = torch.from_numpy(X_test)
    Y_test = torch.from_numpy(Y_test).long()
    # Y_train = torch.zeros(Y_train.shape[0], 10).scatter_(1, Y_train.view(-1, 
1), 1)
    # Y_test = torch.zeros(Y_test.shape[0], 10).scatter_(1, Y_test.view(-1, 1), 
1)
    print(X_train.shape, Y_train.shape, X_test.shape, Y_test.shape)
    return X_train, Y_train, X_test, Y_test
```

模型参数初始化

MLP模型参数采用PyTorch默认的初始化方法，不需要显示进行实现。具体的初始化方法在这里有解释：https://discuss.pytorch.org/t/how-are-layer-weights-and-biases-initialized-by-default/13073/2

超参数的设置如下：

```
losses = model.train(X_train, Y_train, lr=1e-5, reg=1e-3, num_iters=1000, 
batch_size=200)
```

卷积神经网络（CNN）

在Lecture-6 Neural Networks中有讲到，MLP具有如下缺点：

1. 模型中参数的数目太大，容易出现过拟合
2. 全连接网络没有考虑到数据的结构特性

而卷积神经网络可以针对图像数据的特点，很好地改善上述问题。本实验的CNN结构如下：

```python
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

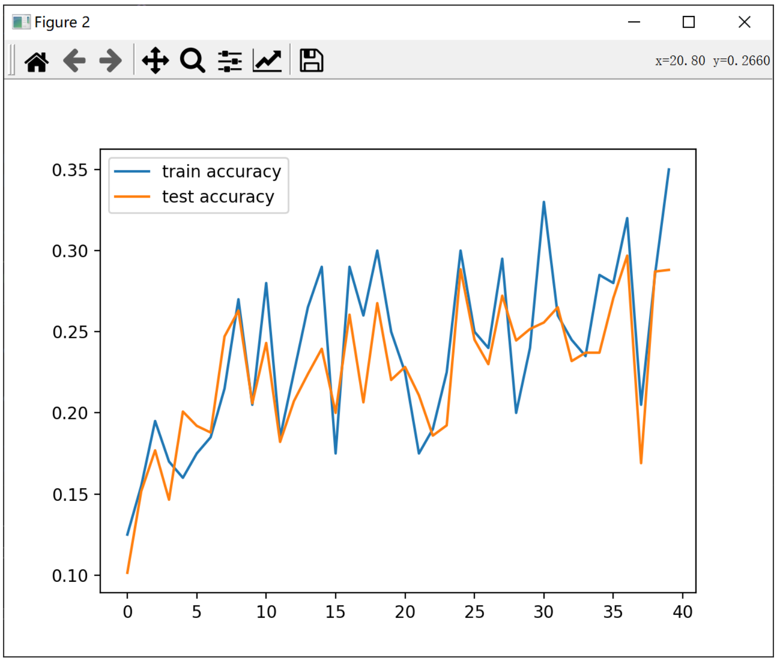
模型参数初始化

CNN的模型参数初始化同样使用PyTorch的默认初始化方法。

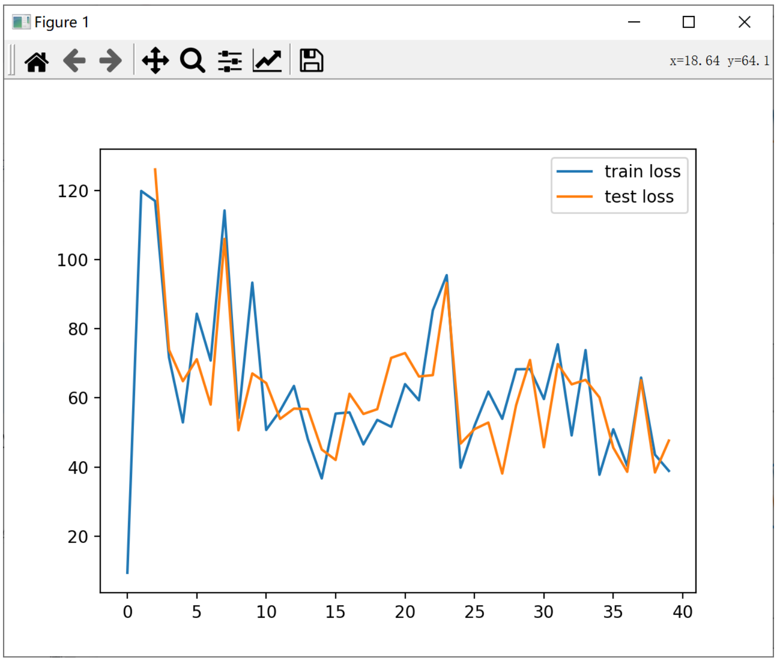
实验结果与讨论

线性分类器（Softmax分类器）

Softmax分类器训练过程中的正确率如下：



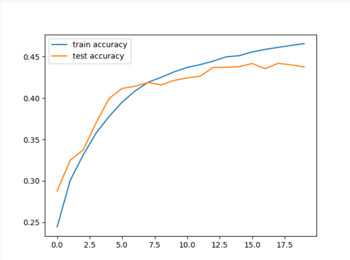
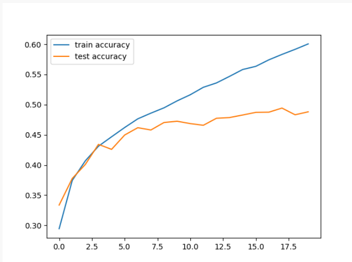
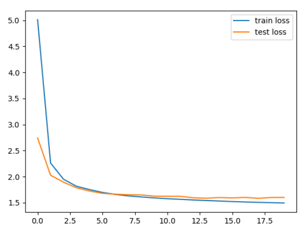
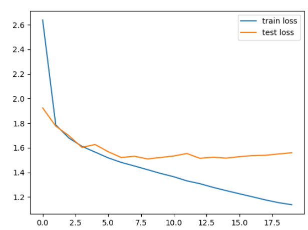
损失变化如下：



如图所示，程序每50次迭代取样一次正确率和损失，在2000次迭代后，测试集上的最终正确率为28.81%。

多层感知机（MLP）

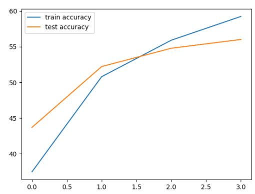
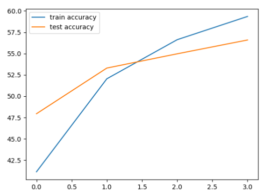
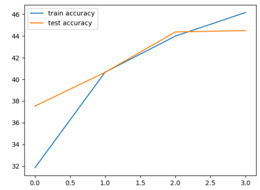
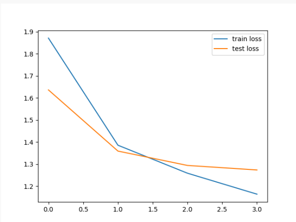
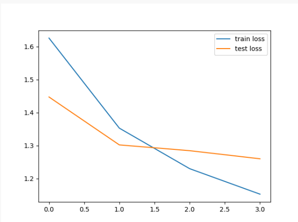
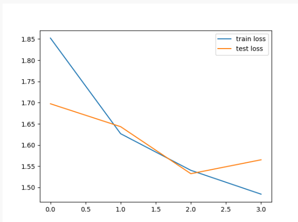
在MLP 实验中，我探究了不同网络层数和不同神经元数量对模型性能的影响。实验结果如下表：

全连接层数	2	3	4
神经元数量	self.fc1 = nn.Linear(3072, 500) self.fc3 = nn.Linear(500, 10)	self.fc1 = nn.Linear(3072, 1000) self.fc2 = nn.Linear(1000, 500) self.fc3 = nn.Linear(500, 10)	self.fc1 = nn.Linear(3072, 2000) self.fc4 = nn.Linear(2000, 1000) self.fc2 = nn.Linear(1000, 500) self.fc3 = nn.Linear(500, 10)
学习率	lr=1e-4	lr=1e-4	lr=1e-4
L2正则化系数	reg=1e-3	reg=1e-3	reg=1e-3
epoch数	epochs = 20	epochs = 20	epochs = 20
batch大小	batch_size=64	batch_size=64	batch_size=64
正确率变化			程序运行太慢，无法得到结果
损失变化			程序运行太慢，无法得到结果
测试集最终正确率	43.77%	48.81%	程序运行太慢，无法得到结果

可以看到，当MLP的网络层数较小时，增加网络层数有助于提高分类正确率，而当网络层数增加到一定程度，则会出现神经元数量太多，程序运行太慢而无法得到结果的情况。

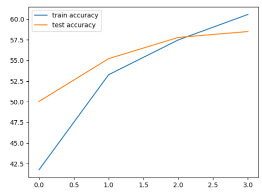
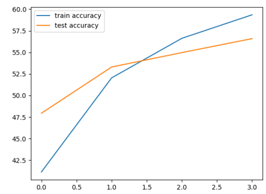
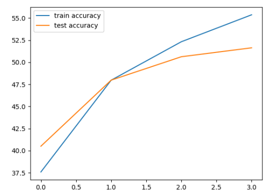
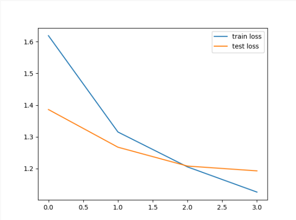
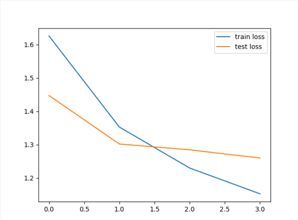
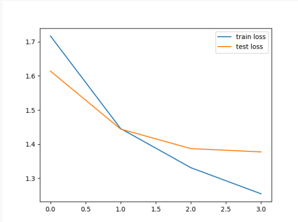
卷积神经网络（CNN）

卷积层数的影响

卷积层数	**1**	**2**	**3**
第一层滤波器参数	self.conv1 = nn.Conv2d(3, 6, 4)	self.conv1 = nn.Conv2d(3, 6, 4)	self.conv1 = nn.Conv2d(3, 6, 4)
学习率	lr=1e-4	lr=1e-4	lr=1e-4
epoch数	epochs = 4	epochs = 4	epochs = 4
batch大小	batch_size=64	batch_size=64	batch_size=64
正确率变化			
损失变化			
测试集最终正确率	56.02%	56.59%	44.5%

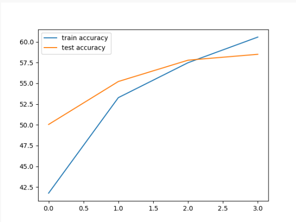
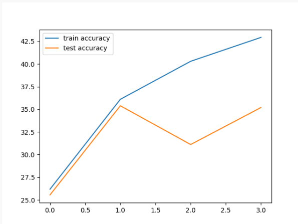
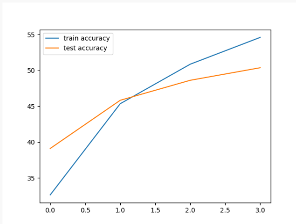
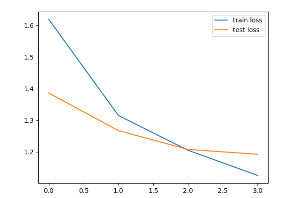
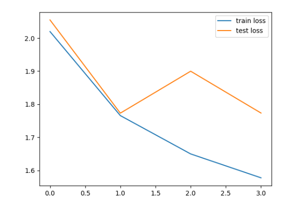
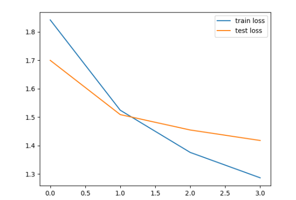
可见卷积层数不是越大越好，从一层增加到两层时有轻微改善，但是三层卷积层时性能下降了。

卷积核大小的影响

卷积层数	**2**	**2**	**2**
第一层滤波器参数	**self.conv1 = nn.Conv2d(3, 6, 3)**	**self.conv1 = nn.Conv2d(3, 6, 4)**	**self.conv1 = nn.Conv2d(3, 6, 5)**
学习率	lr=1e-4	lr=1e-4	lr=1e-4
epoch数	epochs = 4	epochs = 4	epochs = 4
batch大小	batch_size=64	batch_size=64	batch_size=64
正确率变化			
损失变化			
测试集最终正确率	58.5%	56.59%	51.64%

可见，当卷积核大小为(3，3)时，正确率最高，所以适当减小卷积核大小对提高本实验正确率有帮助。

不同训练模型的影响

卷积层数	**2**	**2**	**2**
训练模型	Adam	SGD	SGD Momentum
第一层滤波器参数	self.conv1 = nn.Conv2d(3, 6, 3)	self.conv1 = nn.Conv2d(3, 6, 3)	self.conv1 = nn.Conv2d(3, 6, 3)
学习率	lr=1e-4	lr=1e-4	lr=1e-4
epoch数	epochs = 4	epochs = 4	epochs = 4
batch大小	batch_size=64	batch_size=64	batch_size=64
正确率变化			
损失变化			
测试集最终正确率	58.5%	35.2%	50.38%

可见在本实验中，Adam训练模型效果最好。

主要结论和讨论

本实验得出的主要结论如下：

1. Softmax分类器，MLP和CNN模型性能比较：CNN>MLP>Softmax分类器；
2. 将MLP的全连接层数增大有助于提高正确率，但太大会导致程序运行过慢；
3. 卷积层数不是越大越好，要选择适宜的数量；
4. 适当减小卷积核大小有助于提高本实验正确率；
5. Adam训练模型比较适合本实验的分类任务。

总之，对于具体的分类任务，在训练模型时，需要考虑到数据本身的特性，另外对于模型的具体参数，也需要在参考现有经验的基础上，多做实验。