CIFAR10 预测报告

WeYeah Zhang

2023/10/15

1 Task1

1.1 问题描述

Task 1: per batch training/testing

Please denfine two function named `train_batch` and `test_batch`. These functions are essential for training and evaluating machine learning models using batched data from dataloaders.

To do:

1. Define the loss function i.e nn.CrossEntropyLoss().
2. Take the image as the input and generate the output using the pre-defined SimpleNet.
3. Calculate the loss between the output and the corresponding label using the loss function.

Figure 1: 任务 1 的具体要求

1.2 问题解答

答： 补全train_batch和test_batch函数的答案部分如图所示

```
##################### Write your answer here ##################
output = model(image)
loss_fn =nn.CrossEntropyLoss()
loss = loss_fn(output , target)
###############################################################
```

Figure 2: 任务 1 的答案展示

1.3 实验记录

1.3.1 数据预处理

本实验在 CIFAR10 数据集上进行实验，对训练数据进行了随机裁剪、水平翻转、转换为 PyTorch Tensor，并进行了像素标准化。测试数据只进行了 Tensor 转换和相同的像素标准化。创建了训练和测试数据集对象，并加载到数据加载器中，同时定义了类别标签的名称列表，以便后续的模型训练和评估。

```python
# cifar10 transform
transform_cifar10_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
])

transform_cifar10_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
])

train_set = torchvision.datasets.CIFAR10(root='../data', train=True,
                                        download=True, transform=transform_cifar10_train)
train_dataloader = torch.utils.data.DataLoader(train_set, batch_size=BATCH_SIZE,
                                          shuffle=True, num_workers=2)

test_set = torchvision.datasets.CIFAR10(root='../data', train=False,
                                       download=True, transform=transform_cifar10_test)
test_dataloader = torch.utils.data.DataLoader(test_set, batch_size=BATCH_SIZE,
                                         shuffle=False, num_workers=2)

class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

Figure 3: 数据预处理工作

1.3.2 模型介绍

模型使用 SGD（随机梯度下降法）作为优化器，并采用动态调整学习率的方法
其中，ConvNet的具体结构解释如下:

1. 第一层卷积层: 输入通道数 =3，输出通道数 =4，卷积核大小 =3x3
2. 最大池化层: 2x2 大小的池化窗口，步幅 =2
3. 第二层卷积层: 输入通道数 =4，输出通道数 =8，卷积核大小 =3x3
4. 第一个全连接层: 输入维度 =8x6x6，输出维度 =32
5. 第二个全连接层: 输入维度 =32，输出维度 =10

```python
class ConvNet(nn.Module):
    def __init__(self):
        super(ConvNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 4, 3)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(4, 8, 3)
        self.fc1 = nn.Linear(8 * 6 * 6, 32)
        self.fc2 = nn.Linear(32, 10)
```

Figure 4: 使用的模型源代码

1.3.3 前向传播过程

前向传播的具体过程为:

1. 第一层卷积层 + ReLU 激活函数 + 最大池化层
2. 第二层卷积层 + ReLU 激活函数 + 最大池化层
3. 将特征张量展平，以便进行全连接层的处理
4. 第一个全连接层 + ReLU 激活函数
5. 第二个全连接层，用于输出分类结果

```python
def forward(self, x):
    x = self.pool(torch.relu(self.conv1(x)))
    x = self.pool(torch.relu(self.conv2(x)))
    x = x.view(-1, 8 * 6 * 6)
    x = torch.relu(self.fc1(x))
    x = self.fc2(x)
    return x
```

Figure 5: 前向传播的具体过程

1.3.4 超参数设定

```python
SEED = 1  # 随机数种子，用于复现随机性操作的结果

NUM_CLASS = 10  # 分类问题的类别数量

# Training
BATCH_SIZE = 128  # 每个训练批次中包含的样本数量
NUM_EPOCHS = 30  # 训练迭代的总轮数
EVAL_INTERVAL = 1  # 用于设定多少个 epoch 后进行模型性能评估
SAVE_DIR = './log'  # 保存训练日志和模型检查点的目录

# Optimizer
LEARNING_RATE = 1e-1  # 学习率，用于控制权重更新的步长
MOMENTUM = 0.9  # 动量参数，用于加速权重更新
STEP = 5  # 学习率调度的步数
GAMMA = 0.5  # 学习率调度的衰减率
```

Figure 6: 超参数具体设定

1.4 实验结果

1.4.1 代码问题

在进行实验时，我发现每次实验的结果都不同。原因是作业给的源代码设置的随机种子并没有被正确使用，在此基础上我对代码进行了修改，添加了如下几行, 此时再做实验可以稳定保证结果可复现：

Listing 1: increased code

```python
# random seed Python code here
SEED = 1  # 随机数种子，用于复现随机性操作的结果
random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
torch.cuda.manual_seed(SEED)
torch.backends.cudnn.deterministic = True
```

1.4.2 结果展示

Figure 7: 训练集和测试集的损失函数随着迭代次数的变化

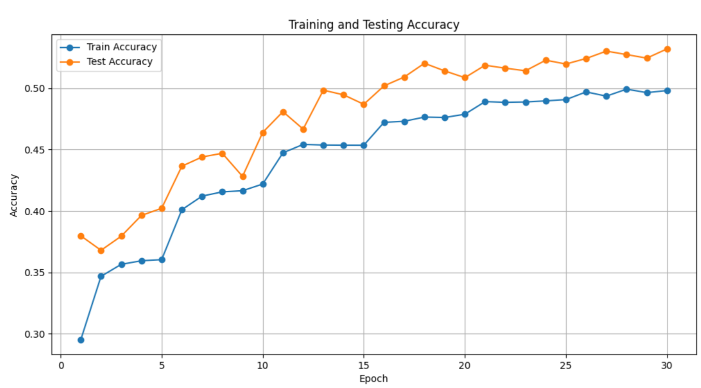


Figure 8: 训练集和测试集的准确率随着迭代次数的变化

1.4.3 模型和参数修改

进行实验时，我发现了模型的准确率比较低，于是我修改了部分网络结构以及超参数，并进行了多次实验，具体修改如下：

选用结构更为复杂的 ResNet18

Listing 2: 选取 ResNet18 作为新的网络结构

```python
#here is my change of the model
    model = models.resnet18(weights=None)
    num_features = model.fc.in_features
    model.fc = nn.Linear(num_features, NUM_CLASS)
    model.to(device)
```

并在不同的超参数之间使用随机搭配的方法，共进行 20 组实验，以寻找最佳的超参数搭配

Listing 3: 随机选取超参数过程

```python
#here is the loop process
    learning_rates = [0.001, 0.01, 0.1, 0.5]
    momentums = [0.1, 0.5, 0.9]
    steps = [1, 5, 10]
    gammas = [0.1, 0.5, 0.9]
    num_iterations = 20
    EVAL_INTERVAL = 5

    for iteration in range(num_iterations):
        learning_rate = random.choice(learning_rates)
        momentum = random.choice(momentums)
        gamma = random.choice(gammas)
        step = random.choice(steps)
        result = train_and_evaluate(learning_rate, momentum, gamma, step, EVAL_INTERVAL)
```

结果如下表格所示，其中每项实验的 $epoch$ 均等于 20，结果显示

$$lr = 0.1$$

$$momentum = 0.5$$

$$Gamma = 0.9$$

$$Step = 5$$

时测试集正确率最高，为 79.28%

1.4.4 最终结果展示

在上述最佳超参数选定的情况下，再改变参数

$$epoch = 30$$

$$batchsize = 64$$

得到的结果如下图所示，最终模型在测试集上的准确率能达到 81.72%

Learning Rate	Momentum	Gamma	Step	test Accuracy
0.01	0.9	0.1	5	tensor(0.7457, device='cuda:0', dtype=torch.float64)
0.001	0.5	0.5	5	tensor(0.5530, device='cuda:0', dtype=torch.float64)
0.5	0.1	0.1	5	tensor(0.7016, device='cuda:0', dtype=torch.float64)
0.001	0.5	0.5	10	tensor(0.5908, device='cuda:0', dtype=torch.float64)
0.001	0.9	0.5	5	tensor(0.6692, device='cuda:0', dtype=torch.float64)
0.01	0.9	0.1	5	tensor(0.7427, device='cuda:0', dtype=torch.float64)
0.001	0.1	0.1	10	tensor(0.5001, device='cuda:0', dtype=torch.float64)
0.001	0.5	0.9	1	tensor(0.5481, device='cuda:0', dtype=torch.float64)
0.5	0.9	0.1	10	tensor(0.1000, device='cuda:0', dtype=torch.float64)
0.01	0.5	0.5	10	tensor(0.7410, device='cuda:0', dtype=torch.float64)
0.01	0.5	0.1	10	tensor(0.7271, device='cuda:0', dtype=torch.float64)
0.01	0.5	0.5	1	tensor(0.5754, device='cuda:0', dtype=torch.float64)
0.5	0.9	0.9	1	tensor(0.6450, device='cuda:0', dtype=torch.float64)
0.01	0.9	0.9	5	tensor(0.7889, device='cuda:0', dtype=torch.float64)
0.001	0.9	0.5	10	tensor(0.7077, device='cuda:0', dtype=torch.float64)
0.5	0.9	0.9	1	tensor(0.2910, device='cuda:0', dtype=torch.float64)
0.1	0.5	0.9	5	tensor(0.7928, device='cuda:0', dtype=torch.float64)
0.5	0.9	0.1	5	tensor(0.5196, device='cuda:0', dtype=torch.float64)
0.01	0.9	0.5	5	tensor(0.7777, device='cuda:0', dtype=torch.float64)
0.01	0.5	0.9	10	tensor(0.7411, device='cuda:0', dtype=torch.float64)

Figure 9: 随机选取 20 组不同的超参数后得到的结果

Figure 10: 最终修改模型和参数后的损失函数变化

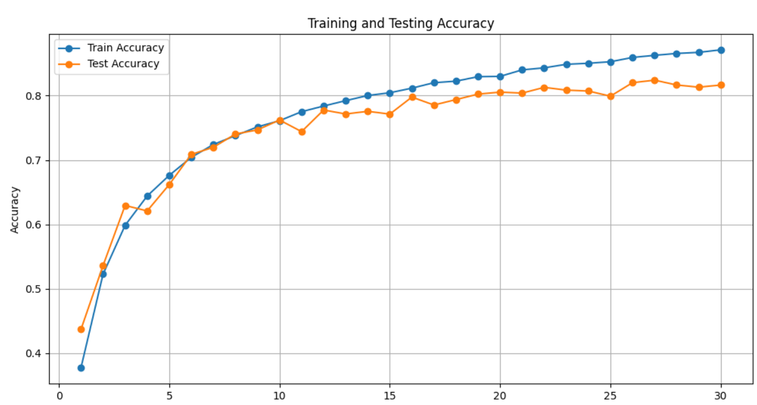


Figure 11: 最终修改模型和参数后的预测准确率变化

2 Task2

2.1 任务 2 的具体要求

任务 2 主要在于补全预测标签和预测概率的表达式，具体函数输入输出限制如下图所示

Task 2: Instance inference

The task is to visualizes an image along with model prediction and class probabilities.

To do:

1. Calculate the prediction and the probabilities for each class.

In [ ]:

```
inputs, classes = next(iter(test_dataloader))
input = inputs[0]
```

In [ ]:

```
##################### Write your answer here ##################
# input: image, model
# outputs: predict_label, probabilities
# predict_label is the index (or label) of the class with the highest probability from the probabilities.
###############################################################

probabilities =
predict_label =
```

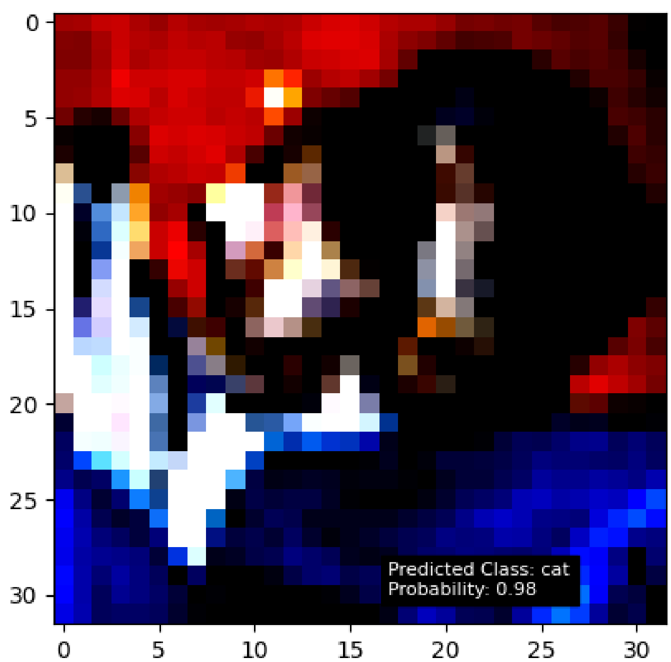
2.2 具体代码

具体的代码块编写如下：

```
#here is the code block of task2
inputs, classes = next(iter(test_dataloader))
    inputs = inputs.to(device)
    input = inputs[0]
    input = input.to(device)

    with torch.no_grad():
        model.eval()
        output = model(input.unsqueeze(0))
        probabilities = torch.softmax(output, dim=1)
        predict_label = torch.argmax(probabilities, dim=1).item()

    predicted_class = class_names[predict_label]
    predicted_probability = probabilities[0][predict_label].item()
    image = input.cpu().numpy().transpose((1, 2, 0))
    plt.imshow(image)
```

```
    plt.text(17, 30, f'Predicted Class:
    {predicted_class}\nProbability: {predicted_probability:.2f}',
             color='white', backgroundcolor='black', fontsize=8)
    plt.show()

    print('Print probabilities for each class:')
    for i in range(len(class_names)):
        print(f'{class_names[i]}: {probabilities[0][i].item():.4f}')
```

2.3 结果展示




```
Print probabilities for each class:
airplane: 0.0001
automobile: 0.0002
bird: 0.0002
cat: 0.9785
deer: 0.0004
dog: 0.0199
frog: 0.0004
horse: 0.0001
ship: 0.0002
truck: 0.0001
```

2.4 代码修改

在最终输出结果中，我发现模型的可视化结果非常差。仔细寻找了原因，发现是在数据预处理阶段对图像进行了标准化操作，但是在最终输出图片时并没有还原回到原始图像，于是我重新修改了代码

```
#here is the changed code
image = (image * np.array([0.2023, 0.1994, 0.2010])) + np.array([0.4914, 0.4822, 0.4465])
image = np.clip(image, 0, 1)
```

最终得到了理想化的结果，分类结果没有发生改变，但是可视化程度增强了