神经网络实践

助教：赖伟

522023330042@smail.nju.edu.cn

作业介绍

1.基于pytorch,实现简单的CNN网络。

导入相关包

In [1]:
```
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
```

数据加载与预处理

此处基于minist进行实验

In [2]:
```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

trainset = torchvision.datasets.MNIST(root='./data', train=True,
                                      download=True, transform=transform)
train_loader = torch.utils.data.DataLoader(trainset, batch_size=64,
                                           shuffle=True)

testset = torchvision.datasets.MNIST(root='./data', train=False,
                                     download=True, transform=transform)
test_loader = torch.utils.data.DataLoader(testset, batch_size=64,
                                          shuffle=False)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz to ./data\MNIST\raw
\train-images-idx3-ubyte.gz
100%|██████████| 9.91M/9.91M [00:08<00:00, 1.13MB/s]
Extracting ./data\MNIST\raw\train-images-idx3-ubyte.gz to ./data\MNIST\raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz to ./data\MNIST\raw
\train-labels-idx1-ubyte.gz
100%|██████████| 28.9k/28.9k [00:00<00:00, 51.1kB/s]
Extracting ./data\MNIST\raw\train-labels-idx1-ubyte.gz to ./data\MNIST\raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz to ./data\MNIST\raw\t
10k-images-idx3-ubyte.gz
100%|██████████| 1.65M/1.65M [00:02<00:00, 804kB/s]
```

```
Extracting ./data\MNIST\raw\t10k-images-idx3-ubyte.gz to ./data\MNIST\raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz to ./data\MNIST\raw\t
10k-labels-idx1-ubyte.gz
100%|██████████| 4.54k/4.54k [00:00<00:00, 2.20MB/s]
Extracting ./data\MNIST\raw\t10k-labels-idx1-ubyte.gz to ./data\MNIST\raw
```

模型的实现

最基础的CNN模型

In [ ]:
```python
class ConvNet(nn.Module):
    def __init__(self):
        super(ConvNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(1, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(128 * 3 * 3, 256),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(256, 10)
        )

    def forward(self, x):
        x = self.features(x)
        x = self.classifier(x)
        return x
```

模型训练评估

In [5]:
```python
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
model = ConvNet().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
total_step = len(train_loader)

# 训练
for epoch in range(20):
    if epoch % 5 == 0:
        print(f"Epoch {epoch}")
    for i, (images, labels) in enumerate(train_loader):
        images = images.to(device)
        labels = labels.to(device)

        outputs = model(images)
        loss = criterion(outputs, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

# 测试
model.eval()
with torch.no_grad():
    correct = 0
    total = 0
```

```
    for images, labels in test_loader:
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    print('简单CNN在MNIST上的测试精度为 {:.2f}%'.format(100 * correct / total))
```

```
Epoch 0
Epoch 5
Epoch 10
Epoch 15
简单CNN在MNIST上的测试精度为 96.40%
```

思考

（1）示例中实现了基础的CNN网络，思考ResNet应如何实现？可基于下列的基础代码完成block部分即可。

（2）注意力机制是transformer的核心，思考如何基于pytorch进行实现？可基于下面的基础代码完成单头注意力即可。

（3）为什么Batch Normalization更适用于图像任务而Layer Normalization更适用于文本任务（思考回答即可）？

In [12]:

```
import torch
import torch.nn as nn
import torch.nn.functional as F

#ResNet18实现
class BasicBlock(nn.Module):
    expansion = 1
    def __init__(self, in_channels, out_channels, stride=1, downsample=None):
        super(BasicBlock, self).__init__()
        # TODO: 请实现 forward 函数
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3,
                               stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3,
                               stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.downsample = downsample #用于尺寸不匹配时的映射
        self.relu = nn.ReLU(inplace=True)

    def forward(self, x):
        identity = x
        out = ...
        if self.downsample is not None:
            identity = self.downsample(x)
        out = ...
        out = self.relu(out)
        return out

class ResNet(nn.Module):
    def __init__(self, block, layers, num_classes=10):
        super(ResNet, self).__init__()
        self.in_channels = 64
        self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3)
        self.bn1 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU(inplace=True)
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

        # 主体的4个阶段，每个阶段包含若干个 BasicBlock
        self.layer1 = self._make_layer(block, 64, layers[0])
        self.layer2 = self._make_layer(block, 128, layers[1], stride=2)
        self.layer3 = self._make_layer(block, 256, layers[2], stride=2)
        self.layer4 = self._make_layer(block, 512, layers[3], stride=2)

        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512 * block.expansion, num_classes)

    def _make_layer(self, block, out_channels, blocks, stride=1):
        #构建每个 stage 的 layer（由多个 block 构成）
        downsample = None
```

```
        if stride != 1 or self.in_channels != out_channels * block.expansion:
            # 如果通道数或尺寸不同，则添加映射
            downsample = nn.Sequential(
                nn.Conv2d(self.in_channels, out_channels * block.expansion,
                          kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(out_channels * block.expansion)
            )

        layers = []
        # 第一个 block 需要带 stride 和 downsample
        layers.append(block(self.in_channels, out_channels, stride, downsample))
        self.in_channels = out_channels * block.expansion

        # 后续 block，输入输出通道一致，stride=1
        for _ in range(1, blocks):
            layers.append(block(self.in_channels, out_channels))

        return nn.Sequential(*layers)

    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)
        x = self.maxpool(x)

        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)

        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.fc(x)
        return x

#注意力机制
class SingleHeadSelfAttention(nn.Module):
    def __init__(self, embed_dim):
        super(SingleHeadSelfAttention, self).__init__()
        self.embed_dim = embed_dim
        # TODO: 请实现 forward 函数
        self.query = nn.Linear(embed_dim, embed_dim)
        self.key   = nn.Linear(embed_dim, embed_dim)
        self.value = nn.Linear(embed_dim, embed_dim)
        self.softmax = nn.Softmax(dim=-1)

    def forward(self, x):
        # x: [batch_size, seq_len, embed_dim]
        Q = ...
        K = ...
        V = ...

        scores = ...
        attn_weig
        out = torch.bmm(attn_weights, V)  # [batch_size, seq_len, embed_dim]

        return out
```