# 实验：深度学习通用训练技巧验证

## 一、 实验内容

深度学习常用正则项及提升模型鲁棒性技巧验证

## 二、 实验目标

（1）掌握权重衰减（Weight Decay）使用及其作用

（2）掌握 Dropout 使用及其工作原理，合理设置其参数

（3）掌握 Batch Normalization 使用及工作原理

（4）了解并实践常用的 Data Augmentation 方法

## 三、 实验内容

#### 3.1数据集准备

继续使用MNIST数据集代码如下

|  |
| --- |
| import torch  from torchvision import transforms  from torchvision import datasets  from torch.utils.data import DataLoader  import torch.optim as optim  import torch.nn as nn  # 1 prepare dataset  batch\_size = 64  # ToTensor 转换图片为张量  # Normalized an tensor image with mean and standard deviation  transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.1307,), (0.3081,))])  train\_dataset = datasets.MNIST(root='./data/mnist/', train=True, download=True, transform=transform)  train\_loader = DataLoader(train\_dataset, shuffle=True, batch\_size=batch\_size)  test\_dataset = datasets.MNIST(root='./data/mnist/', train=False, download=True, transform=transform)  test\_loader = DataLoader(test\_dataset, shuffle=False, batch\_size=batch\_size) |

#### 3.2模型定义(未增加dropout)

使用第一个实验定义的5层网络，在加上2层全连接

并在全连接层中间增加relu

卷积-池化-卷积-池化-全连接-全连接-全连接

|  |
| --- |
| model = nn.Sequential(  # 卷积1操作  # input: channel=1, out\_channel=10, kernel\_size5\*5;  # output: h\*w=(28-5+0+1)/1\*(28-5+0+1)/1=24\*24  nn.Conv2d(1, 10, kernel\_size=5),  nn.MaxPool2d(2),  nn.ReLU(),  # 卷积2操作 接收池化后的conv1 input(h\*w)=h\*w/2=24/2=12  # input: channel=10, out\_channel=20, kernel\_size5\*5;  # output: h\*w=(12-5+0+1)/1\*(12-5+0+1)/1=8\*8  nn.Conv2d(10, 20, kernel\_size=5),  nn.MaxPool2d(2),  nn.ReLU(),  nn.Flatten(),  # 全连接 接收池化后的conv2 input(h\*w)=h\*w/2=8/2=4  # input: 展开后为20\*(4\*4)=320  # output: 10 手写字10个分类  nn.Linear(320, 100),  nn.ReLU(),  nn.Linear(100, 50),  nn.ReLU(),  nn.Linear(50, 10),  ) |

#### 3.3定义损失函数/优化器

采用交叉熵损失函数

使用SGD优化器

|  |
| --- |
| # 3 交叉熵损失函数 优化器  criterion = torch.nn.CrossEntropyLoss().to(device)  optimizer = optim.SGD(model.parameters(), lr = 1e-3, momentum=0.9) |

#### 3.4定义训练验证函数

训练使用model.train(),测试使用model.eval()

|  |
| --- |
| # 4 training cycle forward, backward, update  def train(epoch, train\_cost, test\_cost, mo):  running\_loss = 0.0  for step, data in enumerate(train\_loader, 0):  mo.train()  inputs, target = data  inputs = inputs.to(device)  target = target.to(device)  optimizer.zero\_grad()  outputs = mo(inputs)  loss = criterion(outputs, target)  loss.backward()  optimizer.step()  running\_loss += loss.item()  train\_cost[step] = loss.item()  test\_losses = 0  mo.eval()  correct = 0  total = 0  # print(step)  with torch.no\_grad():  for s, l\_data in enumerate(test\_loader, 0):  images, labels = l\_data  images = images.to(device)  labels = labels.to(device)  l\_outputs = mo(images)  test\_loss = criterion(l\_outputs, labels)  test\_losses = test\_loss.item()  \_, predicted = torch.max(l\_outputs.data, dim=1)  total += labels.size(0)  correct += (predicted == labels).sum().item()  break  test\_cost[step] = test\_losses  if step % 300 == 299:  print('Epoch: %d |train loss: %.3f ' % (epoch + 1, running\_loss/300))  print('Epoch: %d |test loss: %.3f |accuracy: %d %%' % (epoch + 1, test\_losses, 100 \* correct / total))  running\_loss = 0.0  return train\_cost |

#### 3.5运行结果与曲线

并没有出现过拟合情况，但是发现训练loss与测试loss贴合

图形用户界面, 图表

描述已自动生成

#### 3.6修改模型，增加dropout

|  |
| --- |
| model = nn.Sequential(  # 卷积1操作  # input: channel=1, out\_channel=10, kernel\_size5\*5;  # output: h\*w=(28-5+0+1)/1\*(28-5+0+1)/1=24\*24  nn.Conv2d(1, 10, kernel\_size=5),  nn.MaxPool2d(2),  nn.ReLU(),  # 卷积2操作 接收池化后的conv1 input(h\*w)=h\*w/2=24/2=12  # input: channel=10, out\_channel=20, kernel\_size5\*5;  # output: h\*w=(12-5+0+1)/1\*(12-5+0+1)/1=8\*8  nn.Conv2d(10, 20, kernel\_size=5),  nn.MaxPool2d(2),  nn.ReLU(),  nn.Flatten(),  # 全连接 接收池化后的conv2 input(h\*w)=h\*w/2=8/2=4  # input: 展开后为20\*(4\*4)=320  # output: 10 手写字10个分类  nn.Linear(320, 100),  nn.ReLU(),  nn.Dropout(p=0.5),  nn.Linear(100, 50),  nn.ReLU(),  nn.Dropout(p=0.5),  nn.Linear(50, 10),  ) |

#### 3.1.6运行dropout结果与曲线

修改之后，测试loss与训练loss有了偏离。说明drop随机丢弃神经元生效

图表, 直方图

描述已自动生成