小作业二:MNIST 手写数字识别

519030910383 张哲昊

1. MNIST 数据集

MNIST 数据集是机器学习领域中非常经典的一个数据集，由 60000 个训练样本和 10000 个测试样本组成，每个样本都是一张 28 * 28 像素的灰度手写数字图片。本次任务将实验百度飞桨实现 MNSIT 数据集上的数字识别任务，并加以探索。

2. 网络结构

本次实验所使用的网络结构为 LeNet，具体架构如下：
6 个 5×5 的卷积，sigmoid 激活，最大池化，sigmoid 激活，16 个 5×5 的卷积，最大池化，120 个 4×4 的卷积，特征拉平，全连接，sigmoid 激活，全连接。前向传播代码如下：

```
class LeNet(paddle.nn.Layer):
    def __init__(self, num_classes=1):
        super(LeNet, self).__init__()
        # 创建卷积和池化层
        # 创建第1个卷积层
        self.conv1 = Conv2D(in_channels=1, out_channels=6, kernel_size=5)
        self.max_pool1 = MaxPool2D(kernel_size=2, stride=2)
        # 尺寸的逻辑：池化层未改变通道数；当前通道数为6
        # 创建第2个卷积层
        self.conv2 = Conv2D(in_channels=6, out_channels=16, kernel_size=5)
        self.max_pool2 = MaxPool2D(kernel_size=2, stride=2)
        # 创建第3个卷积层
        self.conv3 = Conv2D(in_channels=16, out_channels=120, kernel_size=4)
        # 尺寸的逻辑：输入层将数据拉平[B,C,H,W] -> [B,C*H*W]
        # 输入size是[28,28]，经过三次卷积和两次池化之后，C*H*W等于120
        self.fc1 = Linear(in_features=120, out_features=64)
        # 创建全连接层，第一个全连接层的输出神经元个数为64， 第二个全连接层输出神经元个数为分类标签的类别数
        self.fc2 = Linear(in_features=64, out_features=num_classes)
    # 网络的前向计算过程
    def forward(self, x):
        x = self.conv1(x)
        # 每个卷积层使用Sigmoid激活函数，后面跟着一个2x2的池化
        x = F.sigmoid(x)
        x = self.max_pool1(x)
        x = F.sigmoid(x)
        x = self.conv2(x)
        x = self.max_pool2(x)
        x = self.conv3(x)
        # 尺寸的逻辑：输入层将数据拉平[B,C,H,W] -> [B,C*H*W]
        x = paddle.reshape(x, [x.shape[0], -1])
        x = self.fc1(x)
        x = F.sigmoid(x)
        x = self.fc2(x)
        return x
```

3. 数据集压缩

按照要求将所有（0, 1, 2, 3, 4）的图像仅保留各百分之十，剩余部分不变。代码如下：

```
for batch_id, data in enumerate(train_loader()):
    img = data[0]
    label = data[1]
    if batch_id % 10 != 0:
        mask = (label >= 5)
    else:
        mask = (label >= 0)
    img = img[mask]
    if img.shape[0] == 0:
        continue
    img = img.reshape((-1,1,28,28))
    label = label[mask]
```

4. 改进算法

参考 Focal Loss for Dense Object Detection(ICCV2017) 这篇文章，提出的 Focal Loss 替代传统分类任务中的交叉熵损失函数，用于解决每一类数据数量不均衡问题。如下图介绍所示：该损失函数的核心思想为，对于模型已经非常有把握的类别（即交叉熵损失较小的类别）适当减少模

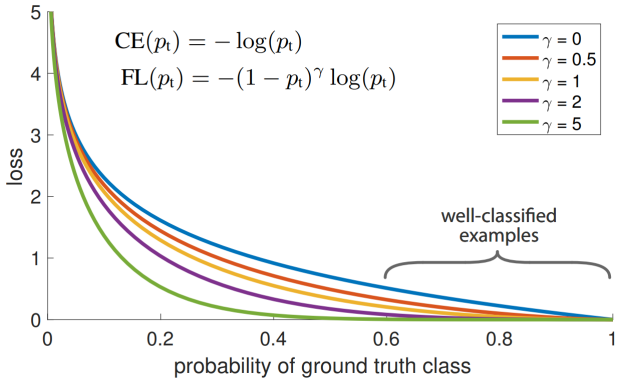

Figure 1. We propose a novel loss we term the *Focal Loss* that adds a factor $(1 - p_t)^\gamma$ to the standard cross entropy criterion. Setting $\gamma > 0$ reduces the relative loss for well-classified examples ($p_t > .5$), putting more focus on hard, misclassified examples. As our experiments will demonstrate, the proposed focal loss enables training highly accurate dense object detectors in the presence of vast numbers of easy background examples.

型参数向着提高这些类别性能的趋势。相反，对于一些现在模型不太有把握的类别（即交叉熵损失大的类别），适当增加其在模型参数更新方向上的权重。当损失函数

$$FL(p_t) = -(1-p_t)^\gamma \log(p_t)$$

中参数 $\gamma > 0$ 时，每个类别的数据不平衡问题就会得到抑制。具体代码实现如下：

```
for gamma in
    [0.2,0.5,0.8,1,1.5,2,2.5,3,4,5,10,15]:
    use_gpu = True
    paddle.device.set_device('gpu:0') if use_gpu
    else paddle.device.set_device('cpu')
    print('start training ... ')
    model.train()
    for epoch in range(EPOCH_NUM):
        for batch_id, data in enumerate(
    train_loader()):
            img = data[0]
            label = data[1]
            if batch_id % 10 != 0:
                mask = (label >= 5)
            else:
                mask = (label >= 0)
            img = img[mask]
            if img.shape[0] == 0:
                continue
            img = img.reshape((-1,1,28,28))
            label = label[mask]
            # 计算模型输出
            logits = model(img)
            # 计算损失函数
            loss_func = paddle.nn.CrossEntropyLoss
(reduction='none')
            loss = loss_func(logits, label)
            with paddle.no_grad():
                pt = paddle.exp(-loss)
            total_loss = (1-pt)**gamma*loss
            avg_loss = paddle.mean(total_loss)
            avg_loss.backward()
            opt.step()
            opt.clear_grad()
```

改进算法后的准确率为：98.0%

5. 实现细节与结果分析

改进算法中，由于需要平衡 0-4 与 5-9 的准确率，实验首先采用较小的 γ 值，训练 10 个 epoch 使得整体的准确率得到 90% 以上。然后逐渐将准确率调大，形成类似于 fintuning 的效果，使得原本对 0-4 较为忽略的模型更加倾向于在 0-4 类上取得高的准确率。γ 的选取序列为 [0.2,0.5,0.8,1,1.5,2,2.5,3,4,5,10,15]。经过大量时间的调试，发现该训练策略相较于其他策略，效果更加明显，使得模型准确率与减少数据量之前差距微乎其微。其他参数分别是：

从上述表格可以看出，数据集的压缩会影响最终

表 1: 超参数的选择与实验环境

超参数/环境	选择/取值
学习率	0.01
优化器	paddle.optimizer.Momentum
批大小	100
EPOCH	10
显卡	GeForce RTX 2080Ti

表 2: 结果对比

原数据准确率	减少数据后的准确率	改进后的准确率
98.7%	92.0%	98.0%

的准确率，同时改进算法能够较为有效地解决这一问题。