第二次小作业：CNN 初探

521030910376 应之未

在本次小作业中，我主要参考了 jittor 官方教程中提供的实现思路和过程。官方教程链接和 Github 仓库链接可点击以下文字访问：官方教程链接 与 Github 仓库链接。

作业代码也可以在本链接中找到。本作业代码可以在配置好 jittor 和 GPU 后直接运行，下载 MNIST 数据集的过程已经包含在代码中。

1. 在原 MNIST 数据集上的实验效果

经过测试，在原数据集上训练得到的模型，在测试集上的 ACC 为 0.9795。

```
Train Epoch: 7 [870/938 (93%)]  Loss: 0.048671
Train Epoch: 7 [880/938 (94%)]  Loss: 0.017915
Train Epoch: 7 [890/938 (95%)]  Loss: 0.045354
Train Epoch: 7 [900/938 (96%)]  Loss: 0.089920
Train Epoch: 7 [910/938 (97%)]  Loss: 0.078007
Train Epoch: 7 [920/938 (98%)]  Loss: 0.094441
Train Epoch: 7 [930/938 (99%)]  Loss: 0.042300
Test Acc = 0.9795
```

图 1: 在原 *MNIST* 数据集上的实验效果

2. 训练集划分

我按照要求对训练集进行划分，将所有（0, 1, 2, 3, 4）的图像仅保留 10%，剩余部分不变。这部分的代码如下图所示：

```
with gzip.open(data_root + filename[0], 'rb') as f:
    self.mnist["imagesf"] = np.frombuffer(f.read(), np.uint8, offset=16).reshape(-1,28, 28)
with gzip.open(data_root + filename[2], 'rb') as f:
    self.mnist["labelsf"] = np.frombuffer(f.read(), np.uint8, offset=8)
self.mnist["labels"] = []
self.mnist["images"] = []
for i in range(len(self.mnist["labelsf"])):
    if i % 10 != 0:
        if self.mnist["labelsf"][i] >= 5:
            self.mnist["labels"].append(self.mnist["labelsf"][i])
            self.mnist["images"].append(self.mnist["imagesf"][i])
    else:
        self.mnist["labels"].append(self.mnist["labelsf"][i])
        self.mnist["images"].append(self.mnist["imagesf"][i])
self.mnist["labels"] = np.array(self.mnist["labels"])
self.mnist["images"] = np.array(self.mnist["images"])
```

图 2: 对 *MNIST* 数据集进行训练集划分的代码

经过该训练集划分后，训练集从原来的 60000 张图片变成了 32400 余张图片，在 batch_size 保持不变的前提下，训练的轮数大致减小为原来的一半。

3. 在划分后的 MNIST 数据集上的实验效果

经过测试，在划分后的数据集上训练得到的模型，在测试集上的 ACC 为 0.943。其中，在小数字上的 ACC 为 0.9368，在大数字上的 ACC 为 0.9740。

```
Train Epoch: 7 [460/507 (91%)]  Loss: 0.097544
Train Epoch: 7 [470/507 (93%)]  Loss: 0.012802
Train Epoch: 7 [480/507 (95%)]  Loss: 0.029138
Train Epoch: 7 [490/507 (97%)]  Loss: 0.039965
Train Epoch: 7 [500/507 (99%)]  Loss: 0.013514
Test Small Acc = 0.9367581241486671
Test Big Acc = 0.974079407529315
Test Acc = 0.9549
```

图 3: 在划分后的 *MNIST* 数据集上的实验效果

可以发现，由于训练集和测试集不再满足独立同分布的性质，因此在数字（0,1,2,3,4）上的准确率远远低于数字（5,6,7,8,9）。因此，为了解决这个问题，我对数据集进行了如下的预处理。

4. 数据预处理

虽然数字（0,1,2,3,4）的训练集样本少了，但是由于在测试集上每个数字依然是基本等概率分布的，因此我们需要提高数字 (0,1,2,3,4) 在训练时的权重。在本次作业中，我将每个数字（0,1,2,3,4）在训练集中重复十次，尽量弥补小数字数量较少造成的问题。

数据预处理的部分代码如下：

```python
# regenerate
self.mnist["labels"] = []
self.mnist["images"] = []
for i in range(len(self.mnist["labelsb"])):
    if self.mnist["labelsb"][i] <= 4:
        for j in range(10):
            self.mnist["labels"].append(self.mnist["labelsb"][i])
            self.mnist["images"].append(self.mnist["imagesb"][i])
    else:
        self.mnist["labels"].append(self.mnist["labelsb"][i])
        self.mnist["images"].append(self.mnist["imagesb"][i])
self.mnist["labels"] = np.array(self.mnist["labels"])
self.mnist["images"] = np.array(self.mnist["images"])
```

图 4: 对处理过的 *MNIST* 数据集进行数据预处理的代码

5. 数据预处理后的实验效果

经过测试，在划分后的数据集上，对于经过数据预处理后训练得到的模型，在测试集上的 ACC 为 0.9701。其中，在小数字上的 ACC 为 0.9558，在大数字上的 ACC 为 0.9852。

```
Train Epoch: 7 [860/932 (92%)]  Loss: 0.058803
Train Epoch: 7 [870/932 (93%)]  Loss: 0.007704
Train Epoch: 7 [880/932 (94%)]  Loss: 0.017680
Train Epoch: 7 [890/932 (95%)]  Loss: 0.000641
Train Epoch: 7 [900/932 (97%)]  Loss: 0.001750
Train Epoch: 7 [910/932 (98%)]  Loss: 0.003264
Train Epoch: 7 [920/932 (99%)]  Loss: 0.022779
Train Epoch: 7 [930/932 (100%)] Loss: 0.003689
Test Small Acc = 0.9558279820976844
Test Big Acc = 0.9851882328738942
Test Acc = 0.9701
```

图 5: 数据预处理后的实验效果

可以发现，在大数字上，经过数据预处理后训练得到的模型的效果与原先基本一致，而在小数字上，新模型的效果有显著提高，但仍然比不上对于原 MNIST 数据集上的效果。

6. 总结

本次作业体现了数据集的重要性，在完成过程中，我通过查阅资料了解到了一些应对数据分布不好的数据集的处理办法。在本次作业的完成过程中，我与张致远和卜家梓两位同学进行了讨论。