第三次小作业：RNN 初探

521030910376 应之未

在本次小作业中，我主要参考了 jittor 官方文档库中提供的 RNN 代码示例。本次小作业的代码也可以在本链接中找到。本作业代码可以在配置好 jittor 后直接运行，下载 MNIST 数据集的过程已经包含在代码中。

1. RNN 网络

不同于第二次小作业中 CNN 网络是由 jittor 官方教程给出，在本次作业中，我利用 jittor 官方文档库中提供的 LSTMCell 函数手动搭建了 RNN 网络，因此添加此节以展示我使用的 RNN 网络的构建方式。

直观上，我采用了作业描述中的“将图片 (28*28) 视为由 28 条一维数据组成的，通过 RNN 网络学习这 28 行数据，来输出一个分类，并与 label 相比较来计算损失函数并更新神经网络”进行神经网络的构建。伪代码展示如下：

```
class RNN(Module):
    def __init__(self):
        super(RNN, self).__init__()
        self.rnn = nn.LSTMCell(input_size, hidden_size)
        self.out = nn.Linear(hidden_size, output_size)

    def execute(self, x):
        hx = jt.zeros(batch_size, hidden_size)
        cx = jt.zeros(batch_size, hidden_size)
        for i in range(x.shape[0]):
            hx, cx = self.rnn(x[i], (hx, cx))
        out = self.out(hx)
        return out
```

2. 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 0.9808。

```
Test Small Acc = 0.9883245767659078
Test Big Acc = 0.9728450936021394
Test Acc = 0.9808
```

图 1: 在原 *MNIST* 数据集上的实验效果

3. 训练集划分

与第二次小作业中对训练集进行划分方式一致，我将所有（0, 1, 2, 3, 4）的图像仅保留 10%，剩余部分不变，具体实现不再赘述。这部分的伪代码如下：

```
如果 是训练集:
    从文件中读取原训练集标签存入labels_f中;
    从文件中读取原训练集图像, reshape后存入images_f中;
    for i in range(len(labels_f)):
        如果i能被10整除:
            将images_f[i]存入images中;
            将labels_f[i]存入labels中;
        否则:
            如果labels_f[i]>4:
                将images_f[i]存入images中;
                将labels_f[i]存入labels中;
否则 (是测试集):
    从文件中读取原测试集标签存入labels_f中;
    从文件中读取原测试集图像, reshape后存入images_f中;
```

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。

4. 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 0.9521。其中，在小数字上的 ACC 为 0.9200，在大数字上的 ACC 为 0.9860。

```
Test Small Acc = 0.9200233508464681
Test Big Acc = 0.9860111088253446
Test Acc = 0.9521
```

图 2: 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）。训练集中大数字的比例增加导致训练模型时更加关注在大数字上的表现，甚至比原模型在大数字上的识别效果更好，而在小数字上的表现则相比原先有了大幅度的退步。因此，为了解决这个问题，我对数据集进行了如下的预处理。

5. 对有偏差的训练集数据进行预处理

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，我将新训练集中的每个标签为数字（0,1,2,3,4）的图像在训练集中重复十次，使得大小数字在数据清洗后的训练集上数量相同，尽量弥补小数字数量较少造成的问题，同时不丢失大数字的丰富样本数量。

数据预处理的伪代码如下：

```
读取分布有偏差的训练集标签存入labels_b中；
读取分布有偏差的训练集图像，reshape后存入images_b中；
for i in range(len(labels_b)):
    如果labels_b[i]小于4：
        for j in range(10):
            将images_f[i]存入images中；
            将labels_f[i]存入labels中；
    否则(labels_b[i]大于4)
        将images_f[i]存入images中；
        将labels_f[i]存入labels中；
```

6. 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 0.974。其中，在小数字上的 ACC 为 0.9649，在大数字上的 ACC 为 0.9835。

```
Test Small Acc = 0.9649737302977233
Test Big Acc = 0.9835424809709936
Test Acc = 0.974
```

图 3: 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，不过仍然比不上对于原 MNIST 数据集上的效果。

7. CNN 和 RNN 网络的异同与感受

CNN 和 RNN 网络的结构有明显不同。CNN 采用卷积层和池化层的结构，比较符合我们传统对图像提取特征时对应的卷积和滤波的操作，卷积层可以有效地提取输入数据的空间特征，而池化层则可以降低数据维度从而减小计算量，理论上说能够有效地处理图像和语音等数据；而 RNN 则采用循环结构，注重数据之间的关联性，将上一个时间步的输出作为下一个时间步的输入，能够处理序列数据，比如自然语言。

但是在本次作业中吗，就在 MNIST 数据集上分类这个任务而言，无论是 CNN 网络还是 RNN 网络都能取得很好的效果。那能不能将 CNN 和 RNN 网络进行结合，从而综合利用 CNN 和 RNN 的优势，提高模型的准确性和效率呢？（比如将 CNN 用于图像特征提取，然后将提取的特征输入到 RNN 中）在未来的作业中，我将对此予以尝试。

8. 总结

本次作业依旧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，也参考了在人工智能编程实践课程中叶南阳老师提供的对于 OOD 数据集的处理办法。