

Assignment 2

Part I

本次实验选择 pytorch 实现简单的 RNN 加法器，训练循环神经网络实际是根据输入数字序列，计算输出结果的概率。

简单加法器的实现：

简单的 RNN 模型由三个模块组成：输入层、输出层和隐藏层。

在所示的加法器模型中有三个基本组成 `embed_layer`、`rnn` 和 `dense`。`Embed_layer` 是将输入的数字转换为 0-9 的高维数字序列，应该是对于原输入在高维上将该特征放大并且分散；`rnn` 就是两层的简单模型，用于处理高维数字序列；`dense` 实际上就是一个将得到的高维输出重新映射会原来最初的低位数字，作为最终的输出。

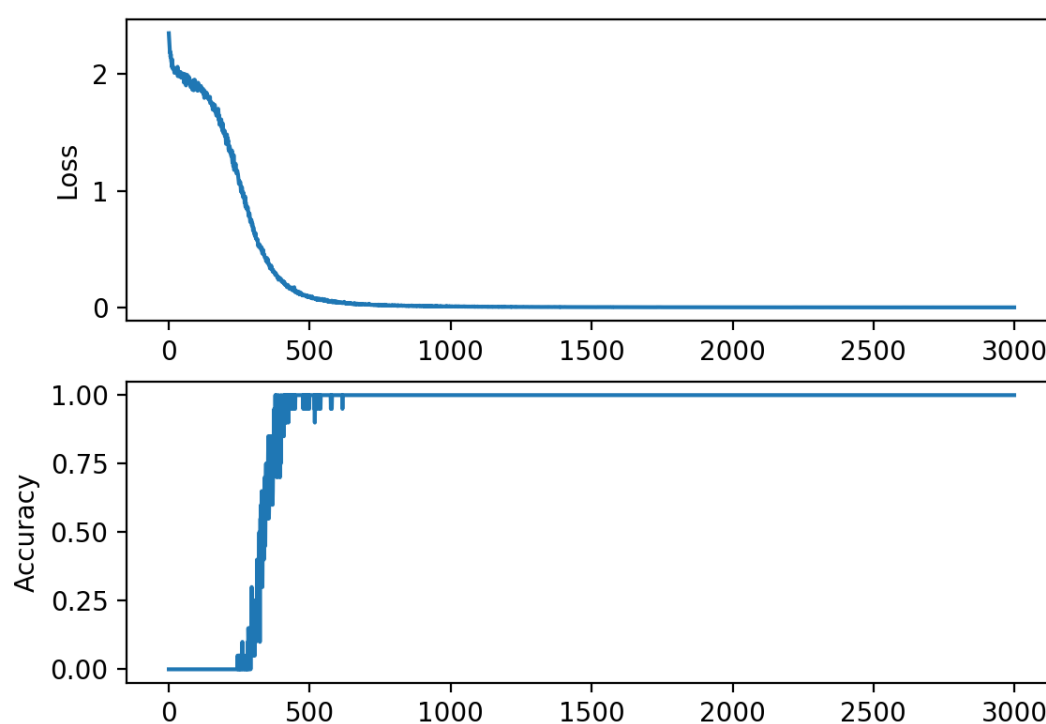
在本阶段最主要的工作就是完成 `forward` 函数，主要就是输入数据的处理以及输出结果的生成过程：

首先输入的是 10 维的 `num1`、`num2`，经过 `embed_layer` 得到两个 32 维空间上的向量，然后将两个向量连接就得到普通 RNN 模型的 64 维输入数据。然后当前使用的普通 RNN 输入输出的维度都是 64 维，堆叠两层，最后将该 RNN 的输出映射回原来 10 维的数据就得到最终的答案。

同时，使用对应的 Adam 优化器进行优化同时可控制步长因子等参数，此时就得到了简单构成的加法器。

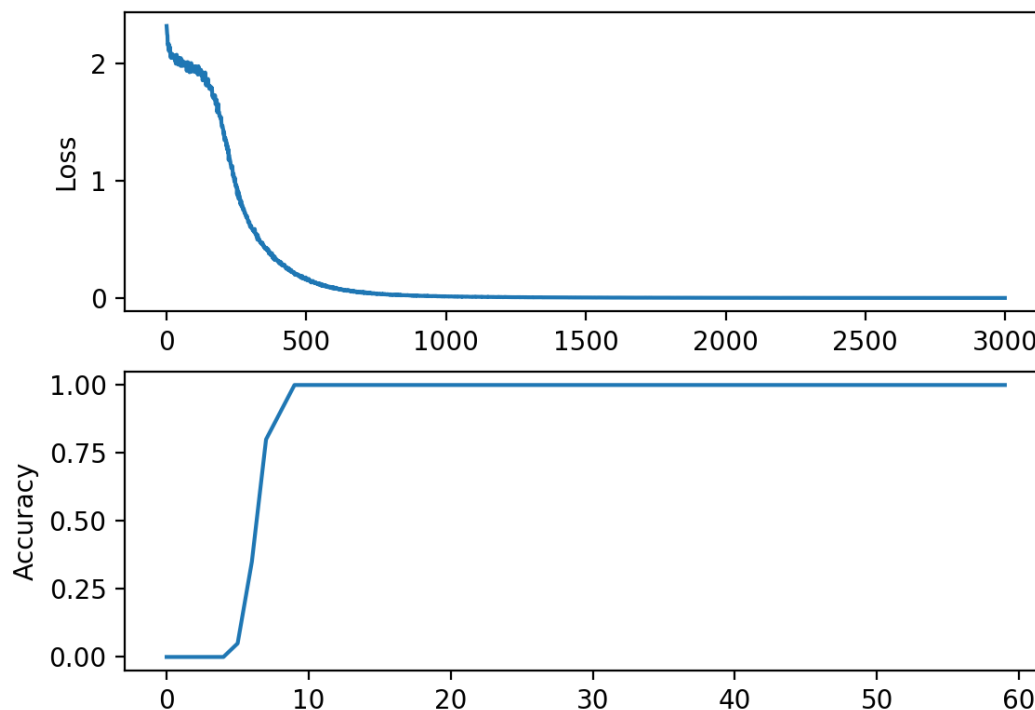
简单加法器测试：

虽然目前只是使用最基本的 RNN，但是对于简单的加法器模型已经能够获得趋近 100% 的正确率。



如图所示，随着模型学习迭代次数增加，损失函数 loss 降低并且最终趋近于 0，在 >500 的状态下就可以稳定收敛在 0，而同样随着迭代次数 >400，正确率就基本稳定在 100%。

上图由于正确率计算过于频繁，可能得出图像抖动严重，所以改为每 50 次迭代计算一次正确性，得到更加清晰的下图：



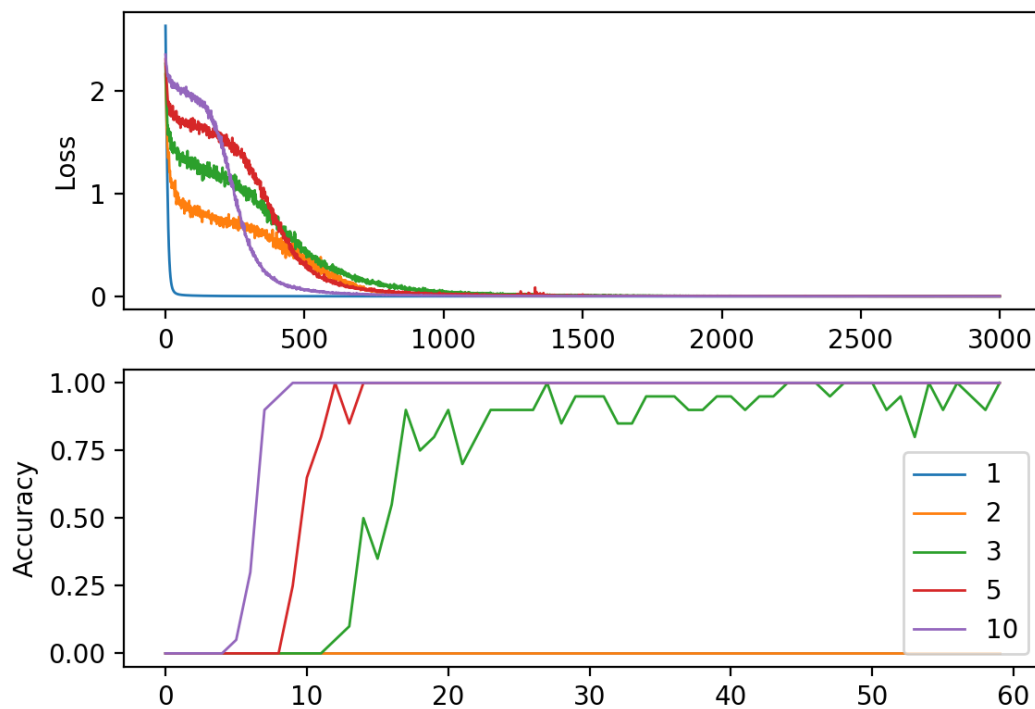
Part II

虽然在上述实验就可以得到结果比较优秀的加法器，但是理论上可以对其进行多个方面的优化，使得其在性能以及效果上达到更优。

考虑训练集位数的调整：

目前模型中使用的是固定且相同长度的训练集和测试集，但是在实际操作中，可能存在训练集与测试集不匹配，特别是当我们希望从特定的训练集中得到一般情况下的模型。

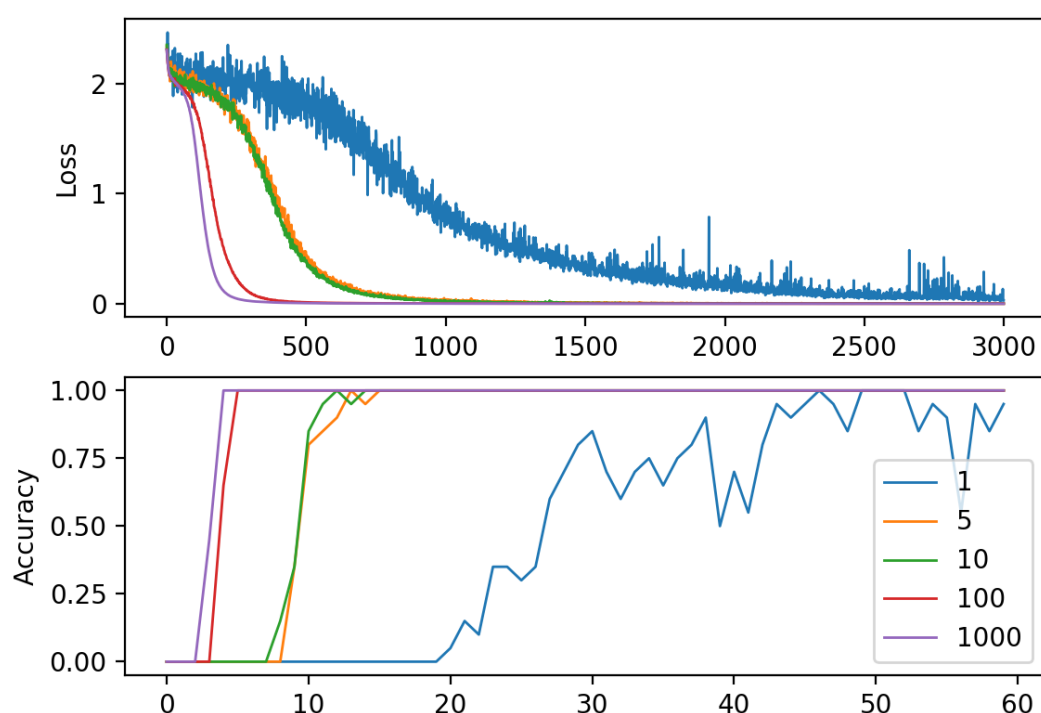
就当前加法器模型而言，希望能够从低位数测试集中获得适用于高位数测试集的模型，于是尝试测试对于相同长度测试集下，不同长度训练集的表现：



最终得出当训练集长度过于短（此处是 1、2），虽然 loss 能够很快收敛到 0，但是得到的模型在应用到高位数加法中正确率不能保证，甚至接近于 0，但是随着训练集位数的增加，在一定的迭代次数后，总能使得正确率达到 100%，而且随着位数的增加，正确率也更快达到 100%。根据条件以及需求，可以选择低位数的数据集进行更多次的迭代，可能达到更快的训练。

考虑一次训练所选取的样本数：

当前模型每次训练采用的样本数为 20，改变每次迭代的样本数可以改变训练的效率，采用不同大小的样本数得到反馈：



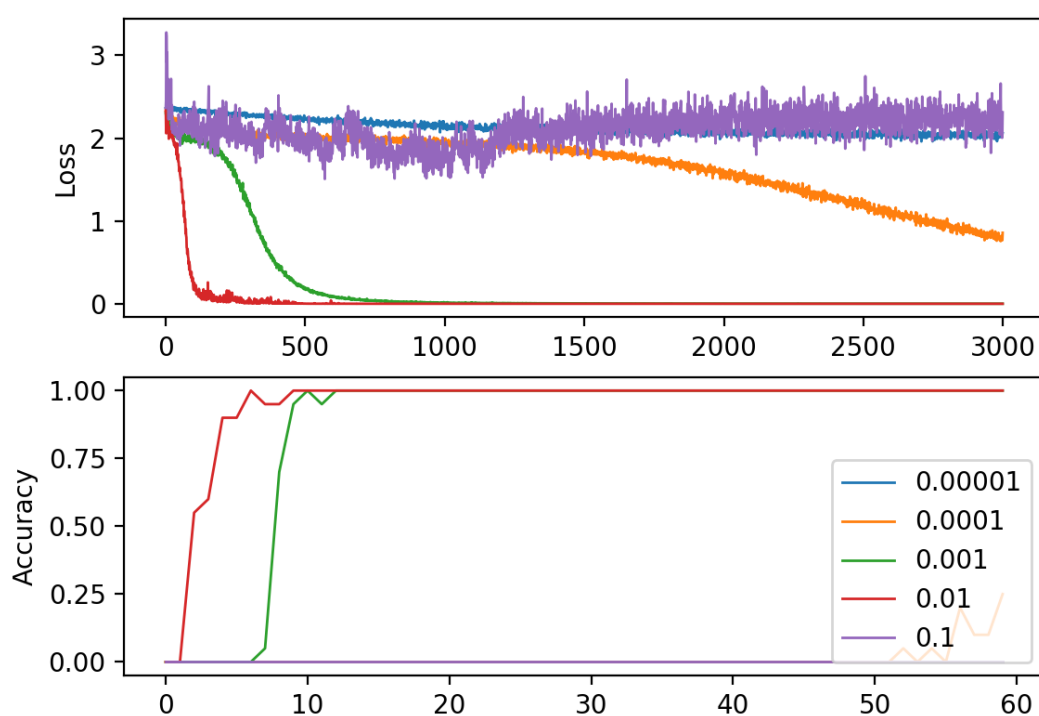
可以从反馈中观察到只会对每次迭代训练的效率造成影响,但是不会影响到训练完成后的效果(即使是 1, 也能在 10000 以内达到收敛, 完成训练)不过采用大样本数进行迭代,虽然能够在少数迭代次数下完成训练,但是每次迭代时间要更长。所以需要选择合适的样本数,能够在最短的时间中完成训练,接下来的测试为了方便展示,就选择每次训练样本集数量为 10。

针对 RNN 模型的优化:

在本次加法器模型中,最重要的部分就是循环神经网络部分,该部分的优化可以带来最直观的变化反馈:

1、步长因子对于模型的影响:

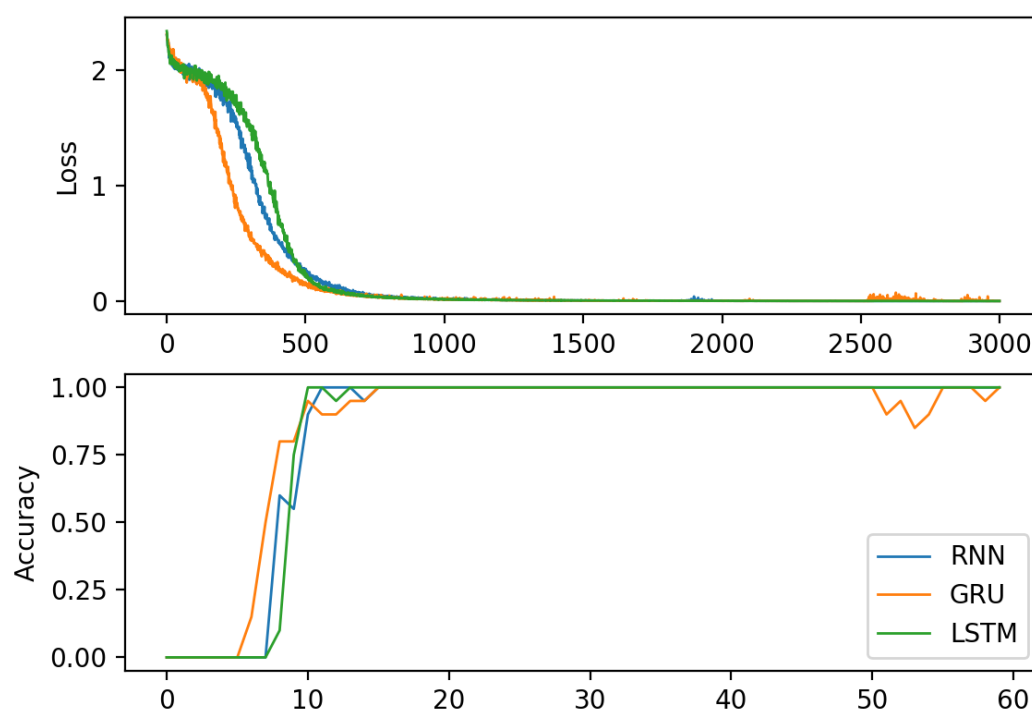
步长因子的不同影响模型对于当前训练集的学习效果,尝试改变步长在 10^{-5} ~1 之间,同时保持最初的加法器模型的其他参数,进行测试:



最终可以发现学习率的影响相当明显，并且得出结论，使用过高或者过低的学习率都可能会收敛问题，过高的学习率会导致收敛过程中效果的波动而不能最终收敛，最终也难以达到高正确率，而过低的学习率则会导致模型需要过多的迭代次数才能最终收敛，对于基本的 RNN 大约在 0.01 附近时可以有最优的收敛效果。

2、使用不同的模型代替 RNN

原模型使用的就是最基本的 RNN 模型，考虑可以使用 GRU 网络或者 LSTM 代替 RNN，比较在相同参数下三者的效果，由于发现步长因子取 0.01，在其他模型中效果较差，所以改为 0.001。

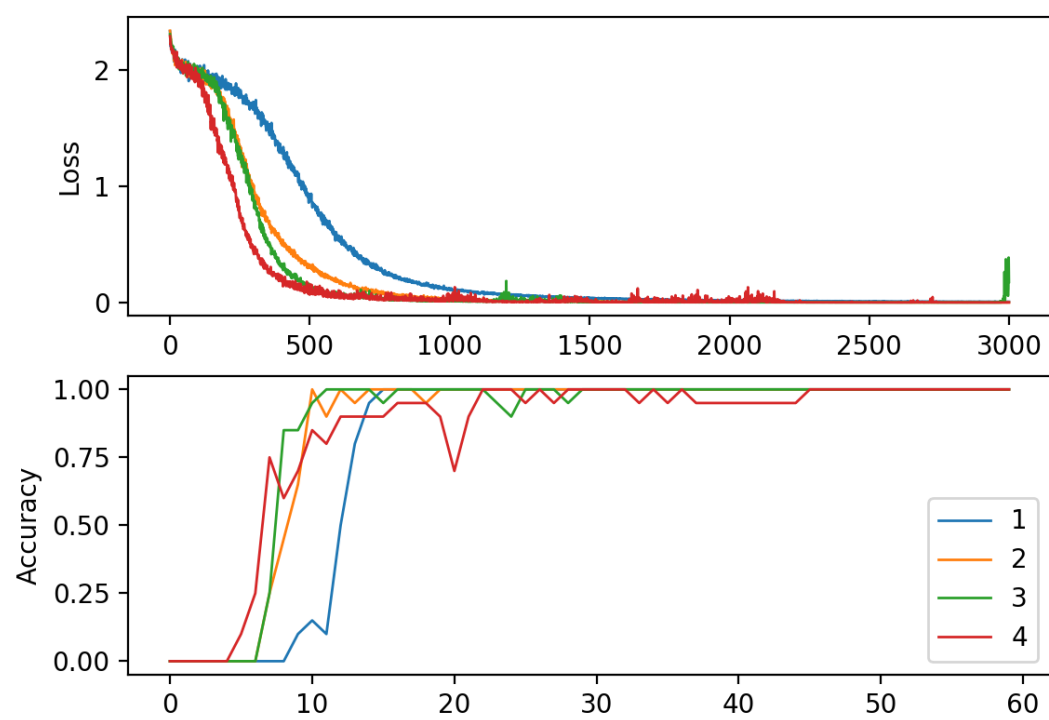


最后发现 GRU 模型在用作加法器上效果明显较差, 正确率不能稳定而且执行时间最长, 而 LSTM 也比 RNN 训练要慢, 在目前条件下 RNN 表现最优, 后续进行进一步探究。

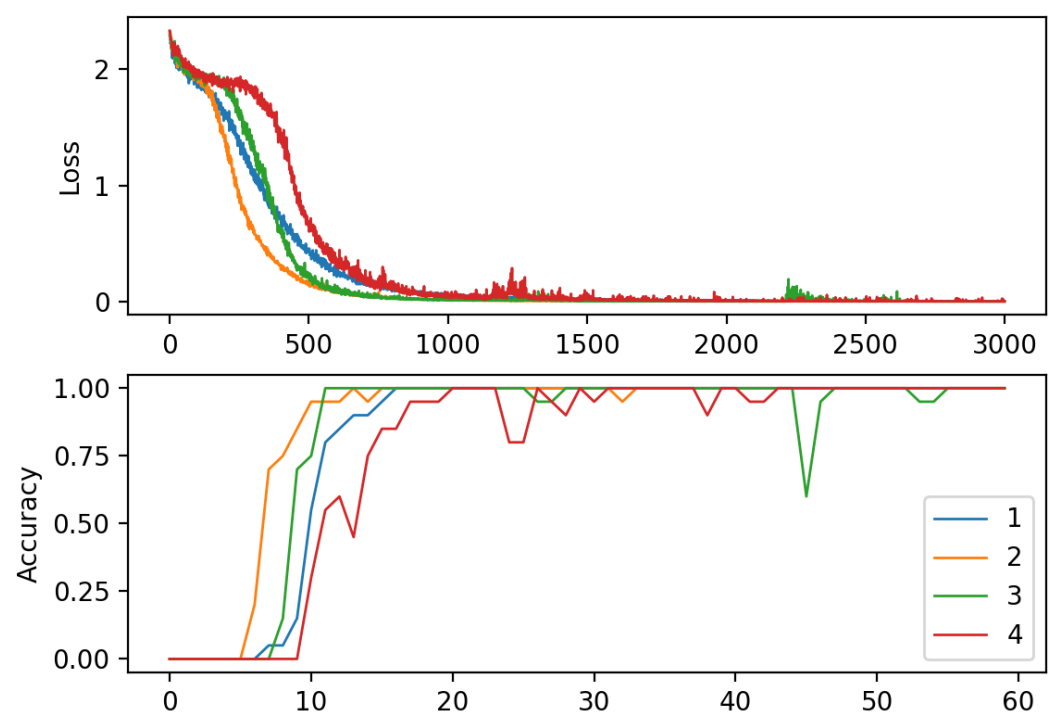
3、使用不同层数的模型

在最初的模型中, 我们使用 RNN 模型只进行 2 层的堆叠, 考虑实际层数不同带来的影响, 在 1-4 层中修改并且观察收敛效果, 这对于三种模型均进行考察。

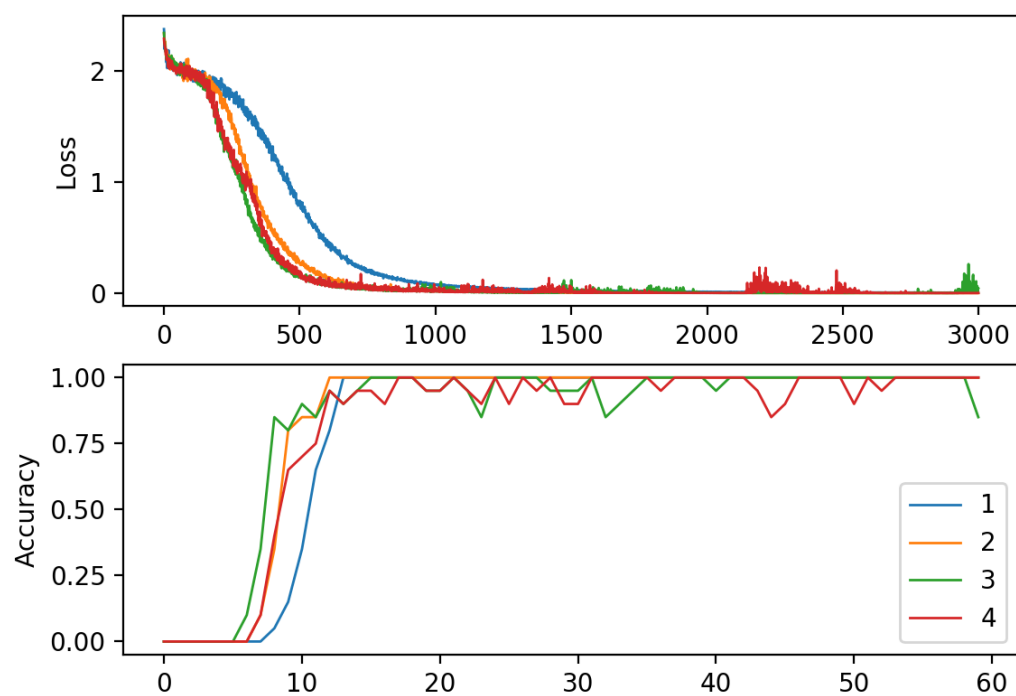
RNN: RNN-3 效果最佳



GRU: GRU-2 效果最佳



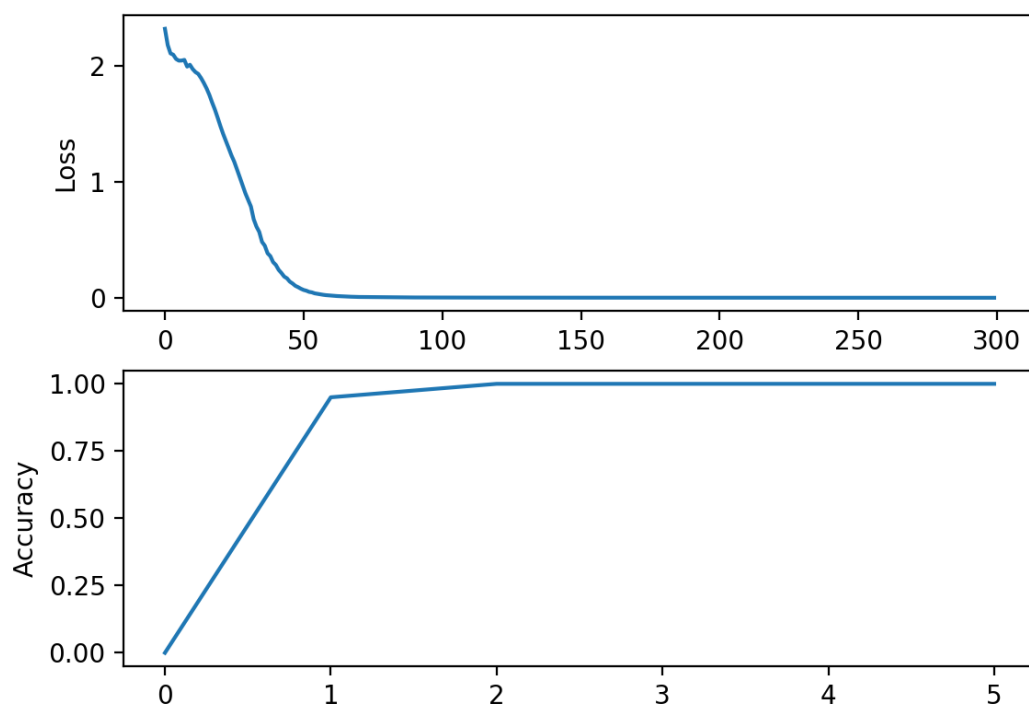
LSTM: LSTM-2 效果最佳



综上所述，再加上上一测试的结果，可以得到 GRU 和 LSTM 模型效果最优的 GRU-2 和 LSTM-2 逊色于 RNN-2，然而 RNN-3 效果最佳

最终优化结果：

最终优化后使用每次训练采用的样本数为 100(实际因条件选择合适大小吧)，选择 RNN 模型，层数为 3 层，步长因子设置为 0.01，可以得到如下结果：



```
step 0  loss 2.321716070175171  accuracy 0.0
step 50      loss 0.21531619131565094      accuracy 0.5
step 100     loss 0.02233988605439663      accuracy 0.95
step 150     loss 0.002150618704035878     accuracy 1.0
step 200     loss 0.000695545575581491     accuracy 1.0
```

基本上能在 100steps 左右 loss 收敛，并且正确率提高到 100%。

运行指令：

运行原始加法器模型和优化后的加法器：

```
python source.py
```