Assignment 2

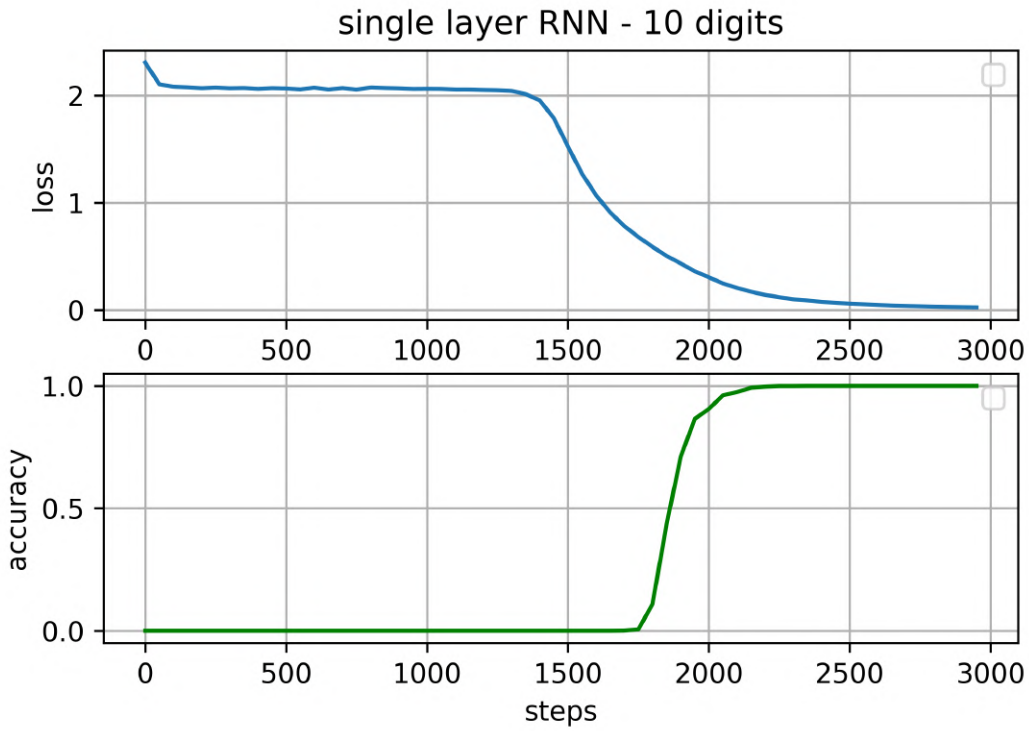
Part 1 RNN 模拟全加器

原理

- 模型由embedding层，单层RNN和一层全连接层组成
- embedding层将作为输入的两个数的每一位数字映射到一个32维向量
- 接着，需要将两个数字每一位的embedding拼接起来，形成64维向量序列，作为RNN的输入
- 输入的两个数是经过倒转的，此时会按从低位到高位的顺序输入RNN，RNN给出一个输出序列
- 输出序列中每个向量经过全连接层再做softmax后得到0-9的概率，选择最大的作为预测，再反转序列就得到了两数之和的预测

初始结果

使用 Tensorflow2.0 ， myTFRNNModel在10位数上的表现如下图



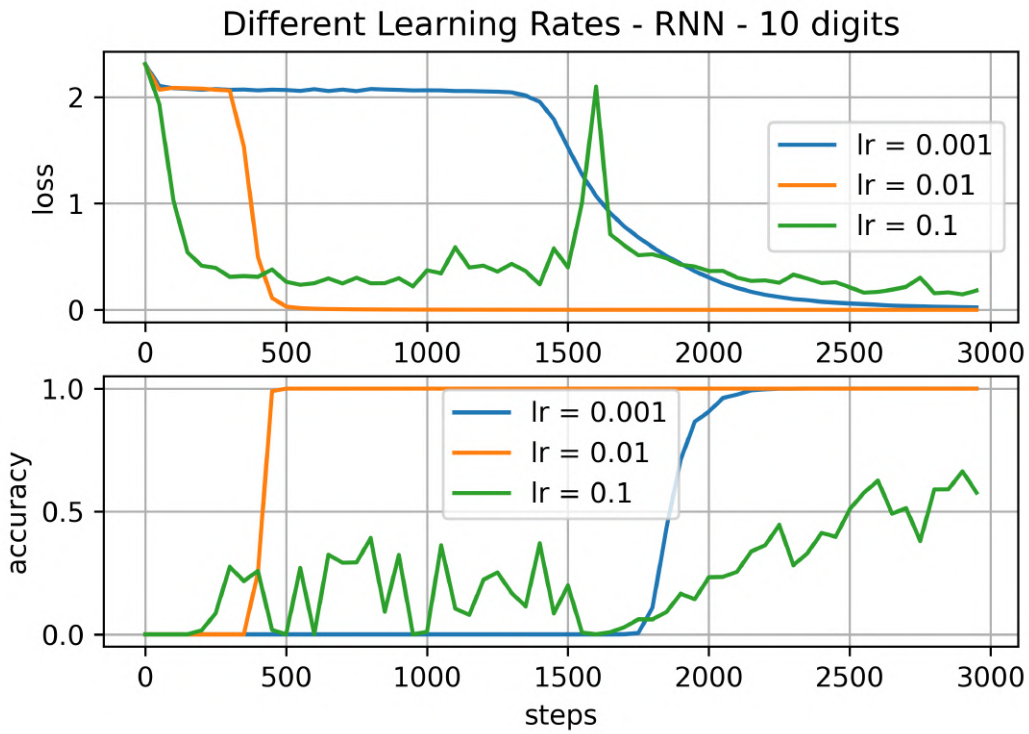
可以看到，在1750个step之后准确率迅速上升，并在2000个steps后接近1的准确率

Part 2 不同位数对初始模型的影响

在开始进行不同位数的试验之前，首先预选择一个较为合理的模型类别、模型层数与学习率

选择学习率

在初始结果中可以很清楚的看到，在模型训练开始很长的一段steps中，模型的误差、损失都处于平台期，几乎不下降。而且整体来看，模型的准确率和误差函数非常平滑，没有抖动，考虑是学习率太低导致的。在初始模型的基础上，分别选择三种学习率：0.001, 0.01,0.1，输入位数仍为10位的情况下，得到结果如下：



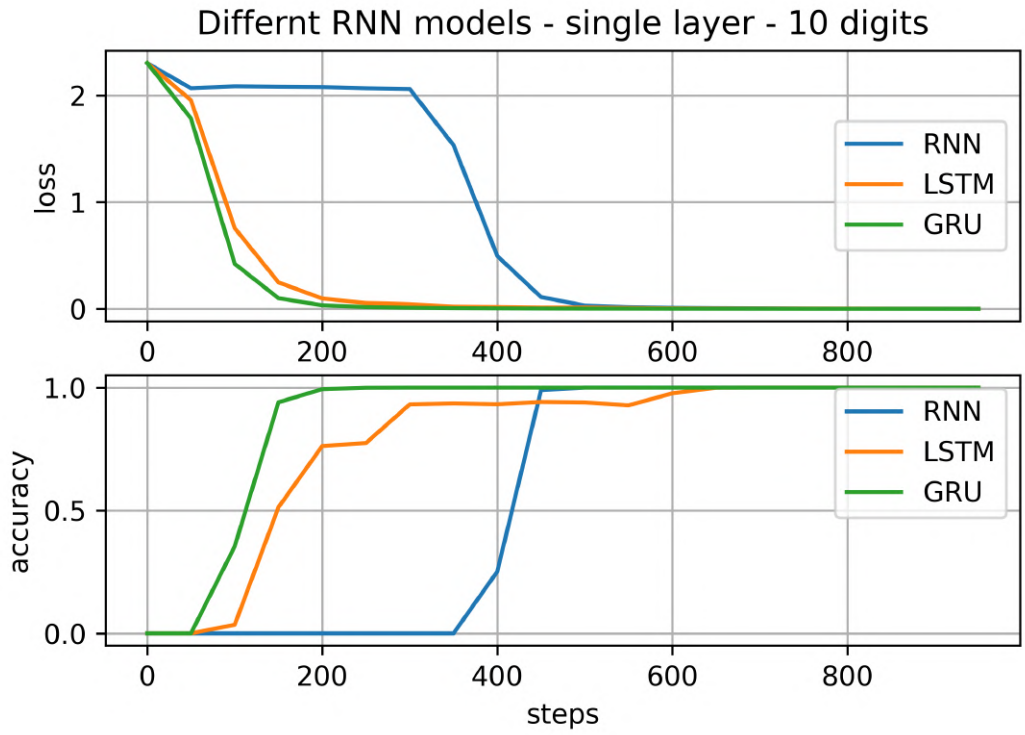
可以看出，学习率为0.01附近时能显著降低收敛所需的steps，并能保证稳定性。学习率为0.1则过大，模型抖动剧烈。因为在原始模型上，学习率设置为0.01左右就能在少于500个steps上收敛，之后讨论模型改进时只显示前1000个steps，并默认设置学习率为0.01

选择模型类别

相对于最基础的RNN模型，keras还提供了更高级的LSTM与GRU模型供选择

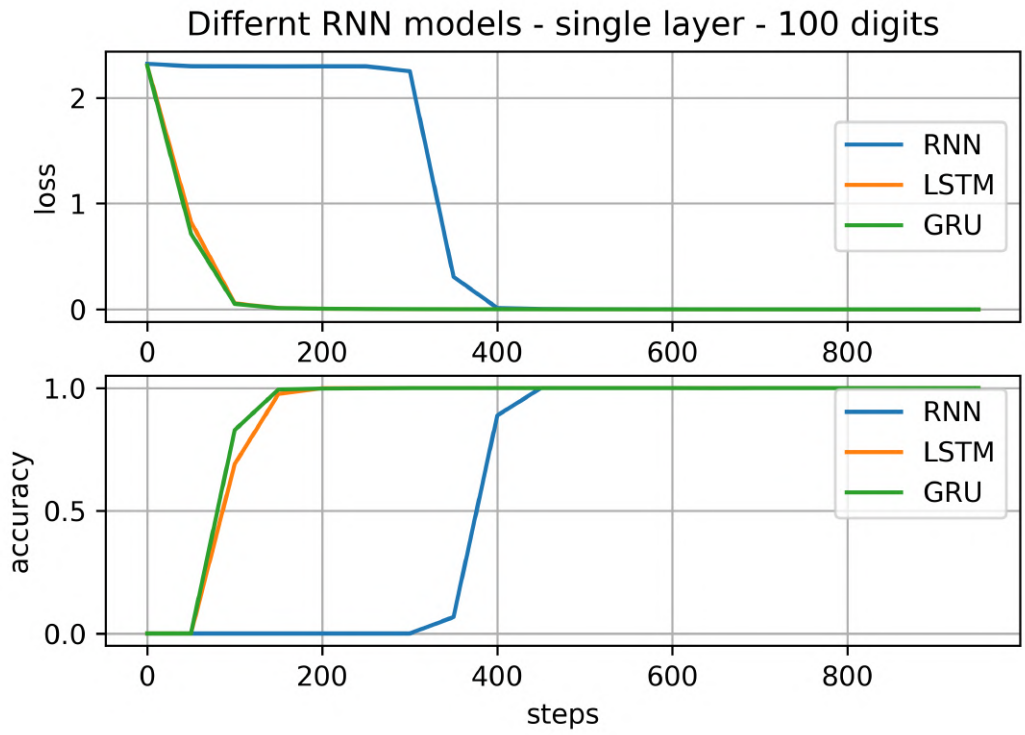
模拟加法器的任务其实局部性非常好，模型需要传递的只有进位信息，而这个信息只由当前输入和前一时刻的进位信息决定。因此无论RNN还是带门电路的LSTM、RNN都能很好地完成任务。但是带门电路的模型有**遗忘门**，可以**忘记一些无关的信息**，比如前一时刻的加和。这使它们能更快收敛。

在长度为10，学习率为0.01的情况下，使用不同模型，得到结果如下：



可以看出GRU收敛所用steps最少。同时，训练用时LSTM快于RNN，而GRU最快，这可能由于GRU模型的参数更少。

当位数达到100位时，不同模型的表现如下：



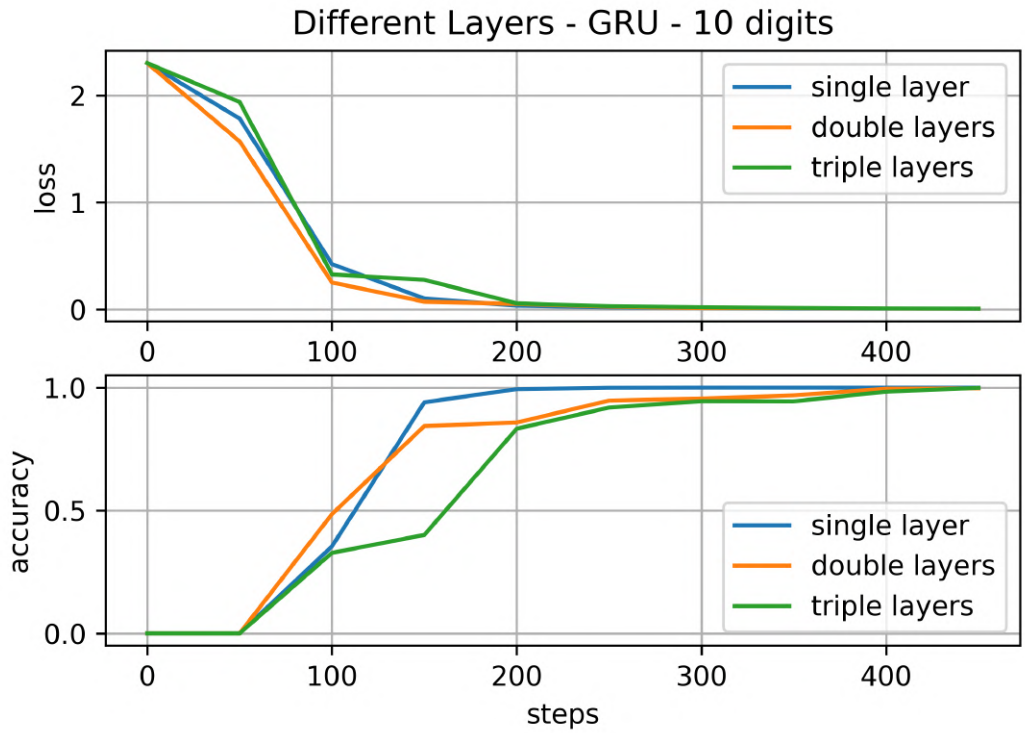
GRU的表现略好于LSTM，但两者非常接近，同时GRU此时训练仅略快与LSTM，并都为RNN训练时间的一半。因此，暂时考虑使用GRU模型进行后续试验，同时当位数更高时保留使用LSTM的选项

另一方面，从这个结果中可以看到无论是RNN，LSTM还是GRU，尽管加数的长度提升了10倍，但它们收敛所需的steps没有太大变化，尤其是LSTM模型甚至大大优于10位时的表现，这可能是因为LSTM的训练需要更多数据，虽然训练的steps一样，但位数变为10倍相当于多提供了10倍的训练数据。这也说明了任务比较简单，同时基于RNN的模型能力都很强大。因此讨论的一个重点在于如何**更快地让模型学会做加法。**

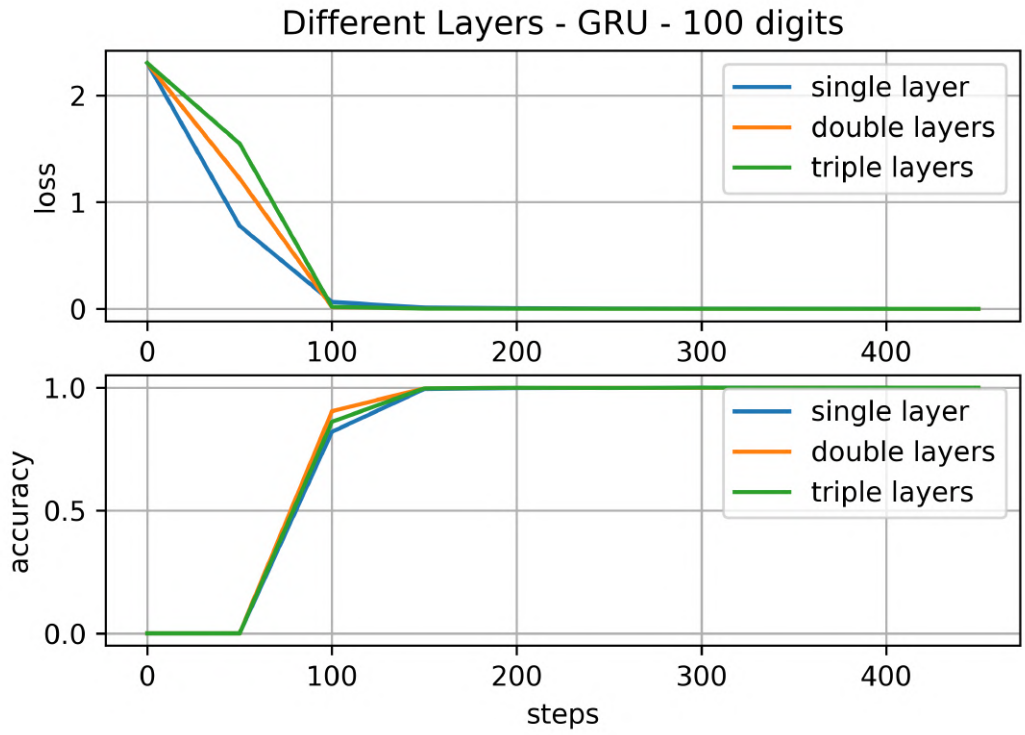
选择模型层数

构建单层、双层与三层的GRU模型，观察是否对表现有影响

10位数据时的结果如下：

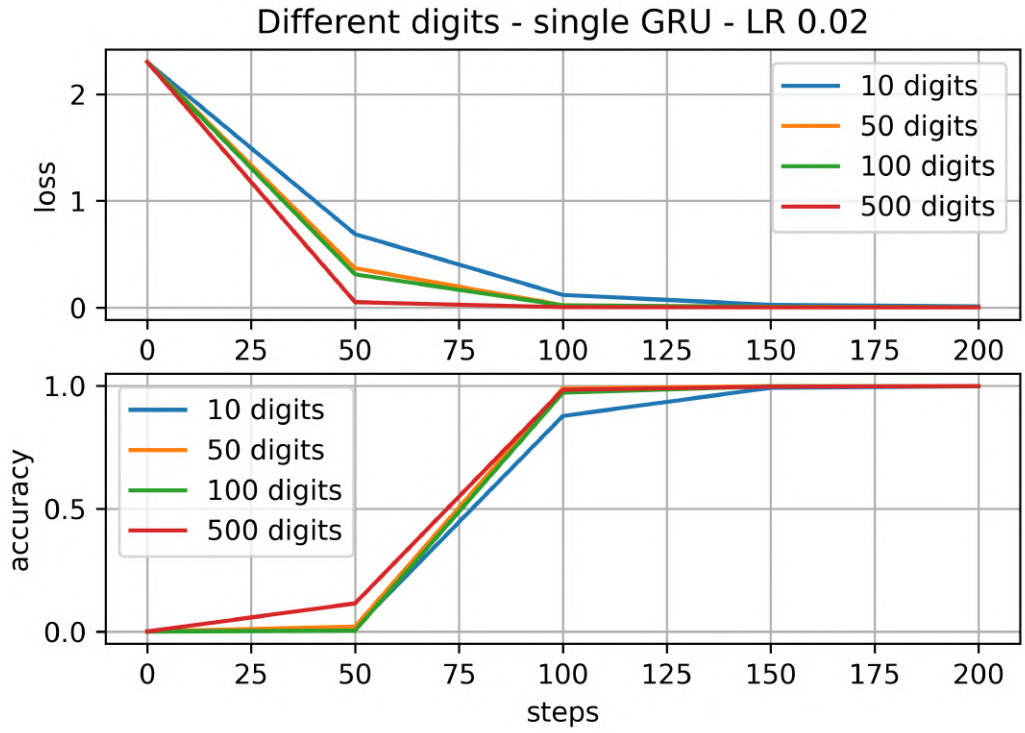


数据为10位时三个模型的区别不大，同时双层和三层花费的训练时间几乎线性增加，当数据增加到100位时也得到了差不多的结论，数据长度在10-100区间中时，模型层数的影响并不大，单层模型就能有很好的表现。不过这里也出现了和上面类似的情况，数据长度增加时，多层模型的表现比原来更好了，因此，当位数更高时保留使用多层模型的选项



改变输入数据位数

经过反复试验，最终选定单层GRU模型，学习率设为0.02，在10位，50位，100位，500位上的结果如下：

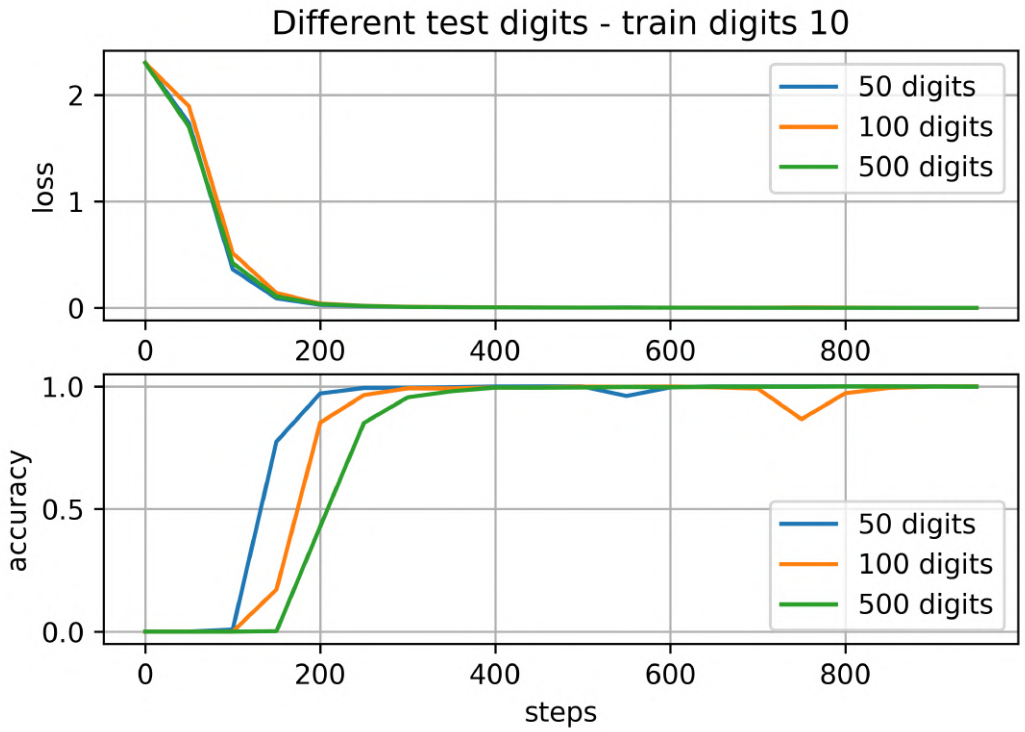


可以看到，在不同的输入位数下，模型的表现相近且都不错，都能在50个steps中loss迅速下降，并在100个steps时几乎收敛，而最开始的RNN模型需要2000个steps才能收敛。这说明模型能在更短的时间内学会模拟加法运算，即模型的学习能力显著增强。

同时，模型似乎能更快的学到输入位数更多的数据。这是由两个因素影响的，首先，更多位数意味着更多的训练数据，但同时对模型的挑战也更大，因为犯错的几率更大了，这要求模型更充分理解进位的含义。

模型泛化能力

在观察训练数据时我发现，训练数据在0-最大数的范围内均匀分布，但这样就会造**成绝大部分的数据都是高位数据**，比如如果指定输入为500位，那么一半的输入数据和全部的测试数据都为499位，这实际上造成了训练数据的倾斜。同时，如果模型真的学会了加法运算，那么它应该是掌握了加法的规律，这样的模型应当具有**泛化能力**。即使在10位的输入上训练，也应该能解决500位的加法问题。



上图是10位训练数据，50、100、500位测试数据的情况。可以看到，随着位数增加，模型做加法的正确率有所下降。不过都能在200个steps以前loss迅速下降，体现了模型具有较强的泛化性。

Part3 模型使用

```
python source.py
```

将展示初始模型和改进后的模型在100位输入上的训练过程