Report

问题描述

设计一个基于 RNN 的神经网络来模拟加法器。

part1：学习编写基于 Tensorflow 2.0 或者 PyTorch 的代码，完成代码的剩余部分。

part2：修改数字的长度，分析模型的表现并改进模型。

代码说明

1、代码运行方式：

在命令行中定位到相应文件夹下，输入命令 python source.py，即可运行程序。提交的代码中，改进前和改进后的模型均会训练 1000 个 batch。运行程序后，程序会先训练改进前的模型，在命令行中显示该模型 loss 的变化和模型训练完成后的准确率，并展示 loss 的变化曲线图；之后，程序会训练改进后的模型，同样在命令行中显示该模型 loss 的变化和模型训练完成后的准确率，并展示 loss 的变化曲线图。

2、其它说明：

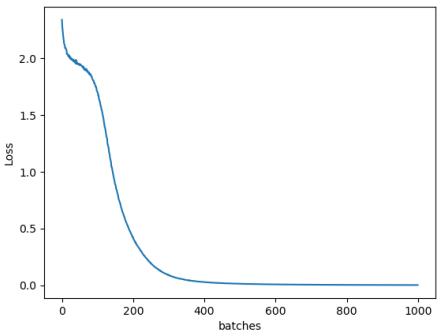
所有的代码是基于 PyTorch 完成的。为了便于完成此次任务的要求并进行进一步的实验，在提供的原始程序上进行了一些细节方面的修改和优化，代码的整体架构和逻辑均没有发生改变。

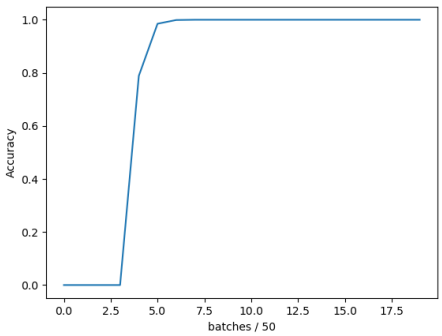
Part 1

观察已有代码可以发现，只有 forward 部分是需要补充完整的。在这部分中，将输入的两个数字分别进行 embedding，连接后通过两层的 RNN，最后通过一个全连接层得到结果。由于 argmax 的操作已经在其它函数中定义了，不需要再 forward 部分中再次定义。

在补充代码的过程中发现，需要在调用 RNN 时添加 batch_first=True 的语句才能正常运行。

由于数字长度较短，仅为 10，考虑将原定的运行 3000 个 batch 修改为运行 1000 个 batch。学习率为 0.001。通过记录每个 batch 后 loss 的值，在每 50 个 batch 后对模型的效果进行评估并计算 accuracy，可以得到 loss 和 accuracy 的变化图。具体如下：





观察上图可以发现，约 300 个 batch 后，模型的准确率就可以达到 100%。loss 的值在前 300 个 batch 大幅度下降，之后会小幅度下降。

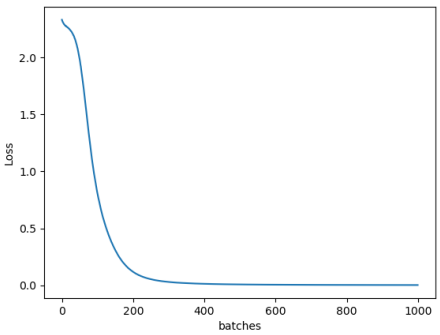
Part 2

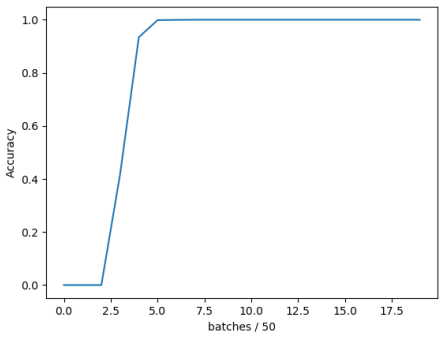
1、增加数字长度

由于 numpy 的 randint 函数不支持随机生成位数过长的大整数，我修改了 data.py 中的数据生成部分和调用数据的函数接口，以便支持生成大整数。

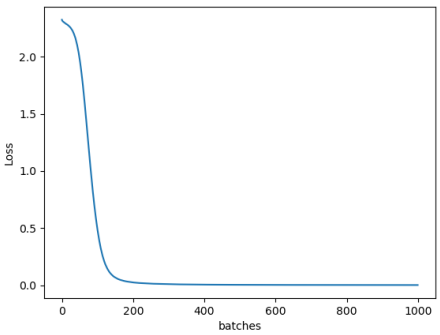
选择数字长度为 100,500,1000 进行测试，学习率为 0.001，对每种数字长度均运行 1000 个 batch，在每 50 个 batch 后对模型的效果进行评估并计算 accuracy。得到以下结果。

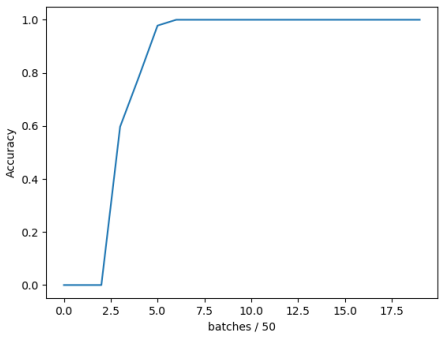
数字长度为 100 时，模型准确率为 100%，loss 和 accuracy 的变化图如下：



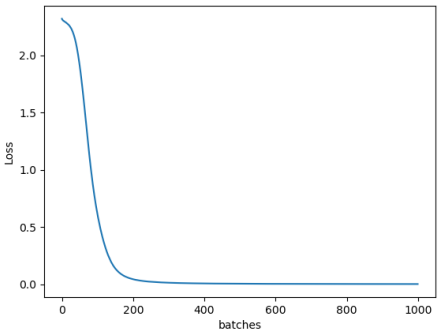


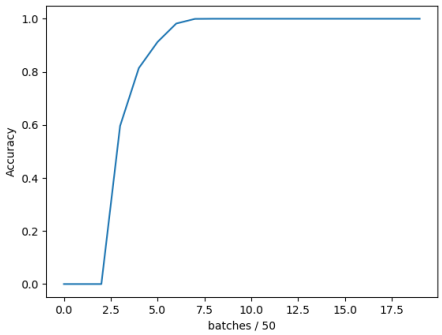
数字长度为 500 时，模型准确率为 100%，loss 和 accuracy 的变化图如下：





数字长度为 1000 时，模型准确率为 100%，loss 和 accuracy 的变化图如下：



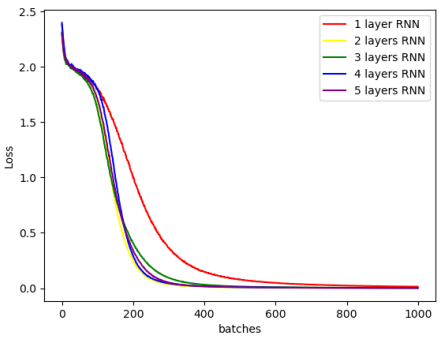


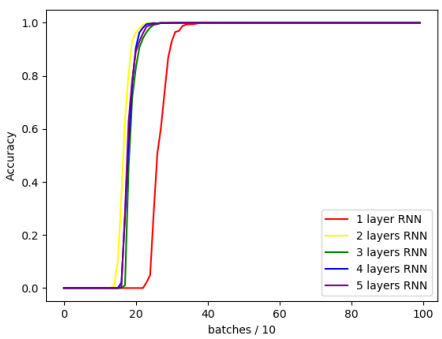
由于电脑 CPU 的限制，当数字长度为 1000 时运行时间已经很长，所以没有继续增加数字长度。通过以上的实验可以发现，该模型对于实验范围内的数字长度，都有 100%的准确率。

2、修改网络层数

因为对于不同长度的数字，模型准确率都为 100%，不能通过比较准确率来比较模型的性能。所以，我们尝试通过修改网络结构，观察模型的收敛情况，来对不同的模型进行比较和判断。

RNN 的层数是模型的一个重要参数。在这部分的实验中，数字长度为 10，学习率为 0.001，在每 10 个 batch 后对模型的效果进行评估并计算 accuracy，总计运行 1000 个 batch。将 RNN 的层数从 1 到 5 进行尝试，结果如下图：



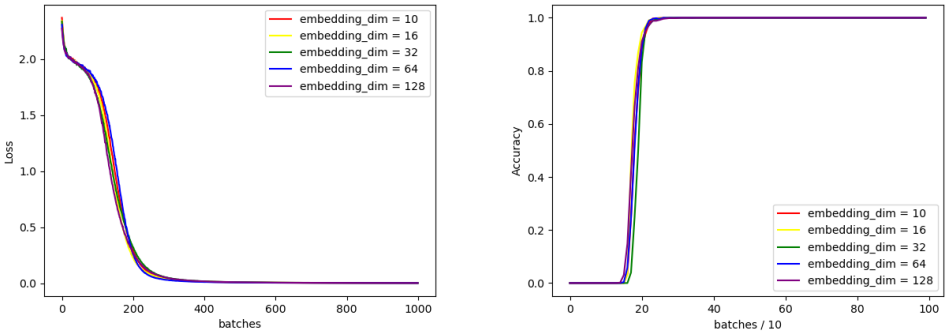


观察上图可以发现，RNN 的层数为 2-5 时，模型的表现非常接近，而 RNN 层数为 1 时模型表现相对较差。左图最下方的曲线和右图最上方的曲线均为 RNN 层数等于 2 的时候，所以层数为 2 时表现最好。

3、修改网络大小

除了 RNN 的层数，中间层的节点数量也会影响网络的结构。在 part1 中，每个数字从原来的 10 维，经过 embedding 后映射到 32 维中。

在这部分的实验中，数字长度为 10，学习率为 0.001，RNN 层数为 2，在每 10 个 batch 后对模型的效果进行评估并计算 accuracy，总计运行 1000 个 batch。将每个数字映射到的向量维数从 10,16,32,64,128 分别进行尝试结果如下图：

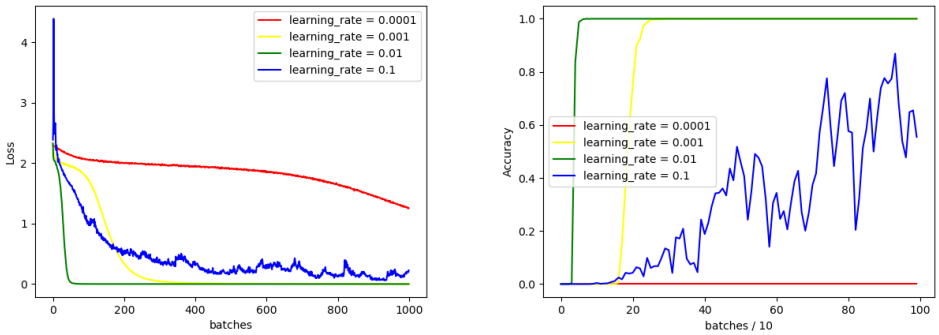


观察上图可以发现，embedding 的维数与模型的表现之间的关联并不明显。

4、修改学习率

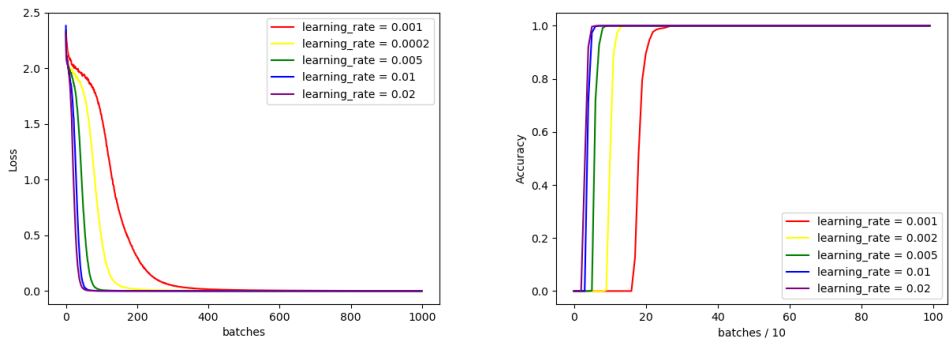
修改网络的结构后，与模型表现相关的另一个超参数是学习率。学习率的大小往往与模型的收敛速度相关。

在这部分的实验中，数字长度为 10，RNN 层数为 2，embedding 维数为 32，在每 10 个 batch 后对模型的效果进行评估并计算 accuracy，总计运行 1000 个 batch。将学习率按照 0.0001,0.001,0.01,0.1 的顺序分别进行尝试，结果如下图：



观察上图可以发现，学习率对模型的表现有非常大的影响。当学习率为 0.0001 时，学习率太小，学习的速度太慢，以至于模型收敛速度太慢，在 1000 个 batch 中无法得到收敛的结果。当学习率为 0.1 时，学习率太大，梯度可能会在最小值附近震荡，甚至无法收敛。

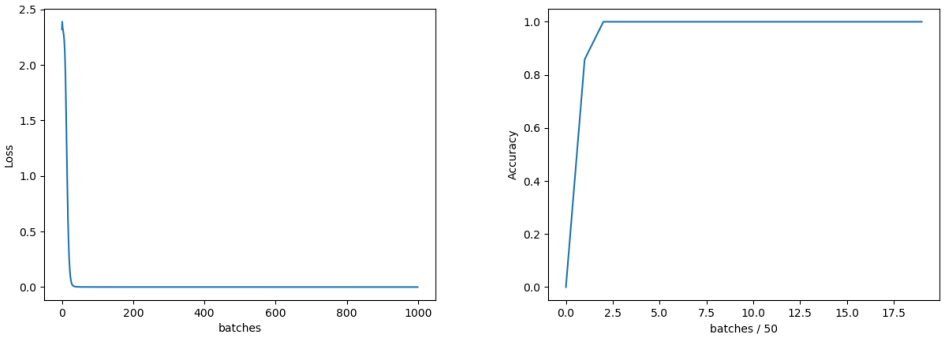
进一步寻找最合适的学习率，将学习率按照 0.001,0.002,0.005,0.01,0.02 的顺序分别进行尝试，结果如下图：



观察上图可以发现，学习率为 0.02 时，模型的学习速度最快，且不会出现学习率过大而无法收敛的情况。在训练约 100 个 batch 之后，模型即可获得 100%的准确率，

5、检验模型

由于电脑 CPU 的限制，在前面的实验中，数字长度均设置为 10。但为了更好地评估模型的性能，还是需要在数组长度较长的情况下对模型进行检验。在这部分中，超参数设定为前文得到的最优参数，将数字长度设置为 1000，在每 50 个 batch 后对模型的效果进行评估并计算 accuracy，总计运行 1000 个 batch。结果如下图：



改进后的模型可以在 100 个 batch 内收敛，使模型得到 100%的准确率。显然，此模型要优于改进之前的模型。

总结

这次实验由于电脑 CPU 的性能限制，在数字长度较大时模型训练过慢，导致在寻找最佳模型的时候设定的数字长度较小，可能会对模型的选择产生一定的影响，不过从结果上看，改进后的模型还是比

较优秀的。

在实验的过程中发现，不断增加数字的长度，模型的准确率依然会到达 100%，且数字长度对 loss 的变化曲线影响不大。产生这个现象可能是问题较为简单。由于加法具有很好的局部性，在某一位上加法产生进位的概率只有 1/2，只有连续的进位才会体现出 RNN 能够处理序列信息的特点。即使数字长度不断增大，由于出现连续进位的概率非常小，即使是简单的网络结构也能做出很好的预测。