用RNN实现加法器

问题描述

给定两个整数，使用RNN计算两个数的和。输入以从低到高的数组给出，输出需要给出一个长度相同的答案序列。

模型1

使用了pytorch版本。

1. 将输入的两个数组通过embedding层变为32维的向量
2. 将两个32维向量拼成64维向量，通过rnn层
3. 通过输入64维，输出10维的全连接层
4. 对最后的结果取最大值作为预测

重复1000个batch，使用交叉熵作为损失函数，使用Adam方式优化。

遇到的问题

为了加速并行运算，pytorch中rnn训练的张量维度是[seq_len,batch_size,hidden_layer_size]

但是其他的隐藏层中，batch_size是第一位，所以使用rnn层时需要开启batch_first=True参数进行训练。

结果

运行结果非常好，loss在0-1000batch中不断下降，最后准确率为1.

同时和tensorflow实现进行对比，loss下降速度更快（第500个batch下降到0.01与第3000个batch下降到0.03，但是正确率都是1），但是每个batch运行速度相对较慢（约5s/50batch与约1s/50batch）。

因为问题非常简单，同时在时间维度，每次只需要记忆1个bit就能够正确处理进位，所以模型表现非常好。

更困难的问题

可以把输入的数字从10^9级别变为10^{100}级别。

模型1

还是使用模型1进行训练，结果依然非常好，正确率为1.loss下降速度和原来相仿，但是运行速度显著变慢。

模型2

因为加法的结果之和当前两个数，以及之前一位的进位有关。所以embedding时不需要将10种情况映射到32维向量，只需要映射到大于10维，最优情况时就可以做到10维向量组成one-hot向量表示数字，剩下一维表示进位。所以将embedding改为13维，对应rnn和全连接层改为26维。

这样得到的运行结果同样准确率达到了1，但是loss进一步变大。运行时间没有变快。

进一步缩小模型：

模型3

1. 将输入的两个数组通过embedding层变为13维的向量
2. 将两个13维向量拼成26维向量，通过rnn层，但是rnn层的输出为13维
3. 通过输入13维，输出10维的全连接层
4. 对最后的结果取最大值作为预测

经过1000个batch之后，loss为0.2，准确率为0.359。把参数降得很低之后并不能做到理论最好情况，或许延长训练时间可以达到收敛。

经过一些参数的调整，如果继续使用rnn，那么最好的方式是调大网络的参数，让网络快速形成记忆，这样可以使用更少的batch就得到很好的结果。

模型4

1. 将输入的两个数组通过embedding层变为64维的向量
2. 将两个64维向量拼成128维向量，通过rnn层
3. 通过输入128维，输出10维的全连接层
4. 对最后的结果取最大值作为预测

这样经过100个batch之后得到正确率为0.999的网络。虽然每个batch速度很慢，但是batch数变小，很快就可以得到结果。

因为加法和当前位置的关系很大，并且不同位置出现相同数字（包括进位）的结果是一样的。另一个思路是我们使用比较小的数字进行学习，然后再使用大的数字进行测试。

模型5

1. 将输入的两个数组通过embedding层变为64维的向量
2. 将两个64维向量拼成128维向量，通过rnn层
3. 通过输入128维，输出10维的全连接层
4. 对最后的结果取最大值作为预测

我们运行1000个batch进行学习，但是训练集上生成的数字是3位数+3位数，测试集上生成的数字是99位数+99位数。这样很快就能训练结束。并且准确率是1.

最后的上传代码中myAdvPTRNNModel为模型5.

运行指令：

```
python source.py
```