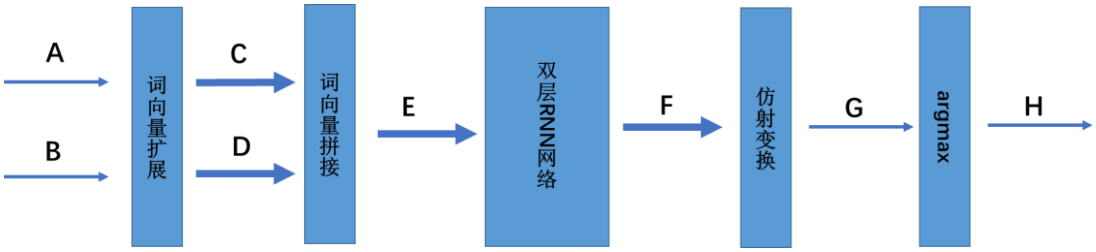
Assignment 2

Part 1

1. 模型描述



1）词向量扩展：循环神经网络的训练通常需要在高维空间，因此通过嵌入层将单个输入扩展至高维度。其中，A 和 B 均是(batch, maxlen)的二维矩阵，分别表示两个加数。在词扩展中，A 和 B 中的每个元素被替换特定向量，分别生成 C 和 D，均是(batch, maxlen, extension)的三维矩阵。

2）词向量拼接：考虑加法操作，以 a=(a1,a2,…)和 b=(b1,b2,…)为例，a 和 b 均是逆序的。首先，计算个位之和 a1+b1，产生和的个位数和进位，然后计算十位数……在 RNN 训练过程中，两个加数的相同位的数据同步输入，产生的和作为输出，进位信息隐含在隐状态中。因此，需要将 C 和 D 中对应位置的词向量进行拼接，生成(batch, maxlen, 2 * extension)的三维矩阵 E。

3）RNN 层：在 RNN 模型定义中，我们取 input_size=output_size=2*extension，层数为 2。查阅 Pytorch 的官方文档可知，RNN 的输入的规格是（seq_len, batch, input_size）的，输出的规格是(seq_len, batch, output_size)。显然加数的最大位数 maxlen 即对应于 seq_len，因此需互换 E 的第 1 维度和第二维度，而 E 的第三维度保持不变。在我的代码中，在词向量扩展前输入即已进行转置，因而 E 无须互换维度。于是，输出 F 是(maxlen, batch, output_size)的三维矩阵。

4）仿射变换：仿射变换压缩输出 F，将 output_size=64 压缩为 10，生成(maxlen, batch, 10)的矩阵 G。从概率的角度来看，我们可以把向量 G[i,j,:]理解为 A[i,j]+B[i,j]产生的相应位的和值的概率。

5）argmax：argmax 过程即依据概率大小决策，对向量 G[i,j,:]中最大概率对应的下标作为相应位的和值，生成(maxlen, batch)的二维矩阵 H，即求和结果。

2. 训练模型

以交叉熵损失函数作为评判标准，以 AdamW 算法作为学习算法，学习率取值 0.001，以批量作为基本单元进行模型训练。

3. 运行结果

(注：在 Part 1 中，extension=32, maxlen=11, RNN 层数为 2)

No. of Step	Value of Loss
0	2.328
50	1.916
…	…
1300	0.00108

…	…
2950	0.0001679

对于 20000 个测试样例，其准确率达到了 1.0000，效果良好。

Part 2

Part 1 所构造的模型已经能够良好地完成整数加法的任务，无须进一步提升模型能力。因此，关注于如何简化模型。改变 RNN 层数以及 Extension 值，观察模型的能力变化情况。为体现区分度，且权衡训练时间，将 maxlen 增加至 50。

1.单层 RNN & Extension=32

使用单层 RNN，设置 extension=32，观察训练过程，结果如下：

No. of Step	Value of Loss
0	2.158
50	0.47
…	…
1500	0.0098
…	…
2950	0.001598

对于 20000 个测试样例，其准确率依旧达到了 1.0000。可见，模型能力虽然简化，但依旧能较好地适用于 50 位内整数的加法运算。同时，观测到在最后的 steps 中，loss 值虽然整体仍处于减小态势，但出现了起伏情况。

2.单层 RNN & Extension=16

使用单层 RNN，设置 extension=16，观察训练过程，结果如下：

No. of Step	Value of Loss
0	2.573
50	0.69
…	…
1500	0.1488
…	…
2900	0.0223
2950	0.0205

Extension 值减半，使得 RNN 的输入维度减半，RNN 模型的能力也随之下降。最终损失值比情况 1 大一个数量级。同时，对于 20000 个测试样例，其准确率是 0.9673。尽管准确率较高，但出现了错误的预测结果。当要求高准确度时，该模型的能力已有所欠缺。

3.单层 RNN & Extension=8

使用单层 RNN，设置 extension=8，观察训练过程，结果如下：

No. of Step	Value of Loss
0	2.628

50	1.098
…	…
1500	0.317
…	…
2900	0.1610
2950	0.1558

从损失值的变化，我们可以看出，当 extension=8 时，模型的能力大大降低。对于 20000 个测试样例，其准确率是 0.0579。可见，此时，该模型已不再适用于整数的加法运算。

（接下来，恢复 RNN 的层数至双层，观察模型能力的回升情况）

4. 双层 RNN & Extension=8

使用双层 RNN，设置 extension=8，观察训练过程，结果如下：

No. of Step	Value of Loss
0	2.432
50	0.85
…	…
1500	0.3024
…	…
2900	0.0342
2950	0.0318

从损失值的变化情况，我们可以发现损失值较情况 3 显著减小。同时，对于 20000 个测试样例，其准确率是 0.9136。可见，模型的能力显著回升。但是，当要求高准确率时，模型能力依旧有所不足。

5. 双层 RNN & Extension=16

使用双层 RNN，设置 extension=16，观察训练过程，结果如下：

No. of Step	Value of Loss
0	2.075
50	0.527
…	…
1500	0.03757
…	…
2900	0.0030
2950	0.0027

从损失值的变化情况，我们可以发现损失值比情况 4 再一次显著减小。同时，对于 20000 个测试样例，其准确率是 1.0000。可见，模型的能力已经回升至不弱于情况 1。

Part 3

在 Part 2，我们分析了模型的复杂程度（即 RNN 层数和 Extension 取值）对模型能力的影响，在该部分，我们分析学习率取值对模型收敛速度的影响。以 maxlen=50,单层 RNN、extension=32 为例。

1）当 learning_rate=0.001，train_steps=3000 时

No. of Step	Value of Loss
0	2.158
50	0.47
…	…
1500	0.0098
…	…
2950	0.001598

对于 20000 个测试样例，其准确率是 1.0000

2）当 learning_rate=0.01，train_steps=1000 时

No. of Step	Value of Loss
0	2.613
50	0.4297
…	…
500	0.00266
…	…
950	0.0007

对于 20000 个测试样例，其准确率是 1.0000。从损失值的变化情况，可以看出模型的收敛速度明显加快。

3）当 learning_rate=0.1，train_steps=1000 时

No. of Step	Value of Loss
0	2.2479
50	0.3147
…	…
400	0.0496
450	0.0548
…	…
600	0.01816
650	0.03223
…	…
850	0.0138
900	0.0215
950	0.0161

对于 20000 个测试样例，其准确率是 0.72915。从损失值的变化情况，我们可以观察到显著的起伏现象，模型收敛能力严重受损。

总结

RNN 的层数直接影响 RNN 的复杂程度，Extension 的值决定输入 RNN 的数据空间维度。两者都能影响模型的能力。双层 RNN 和 Extension=32 的组合，能够极佳地适用于整数加法的模拟任务。考虑简化模型，对于高准确率的要求，单层 RNN 和 Extension=32，双层 RNN 和 Extension=16 的组合，也能良好地适用于整数加法的模拟任务。同时 learning_rate 取值 0.01 既能保证良好的收敛性，又能获得较快的收敛速度。

附言

1.运行指令

程序使用 Pytorch 框架，直接键入“python source.py”即可。

2.变更模型

默认使用 myPTRNNModel()模型。若要选择 myAdvPTRNNModel()模型，须在 source.py 中改变相应注释行。其中 myAdvPTRNNModel()采用的是简化版模型，单层 RNN 和 Extension=32 的组合，learning_rate=0.01。