Assignment-2

循环神经网络

复旦大学计算机系

1. 概要

项目主要使用 Pytorch 深度学习框架，完成了应对加法任务的循环神经网络 RNN 的创建、训练与检验，同时探索、比较了不同参数下的 RNN 的学习能力（如数据集中整数长度参数，学习率设置，隐藏单元的初始化，RNN 层数，非线性激活函数的设置等），也构建了 LSTM/GRU 等 RNN 变体网络，并通过可视化的手段对于模型进行分析。

2. RNN

2.1 RNN 简介

循环神经网络（Recurrent Neural Networks, RNN）是能有效处理时间序列任务的神经序列模型，其优势在于考虑到输入之间的相关性，在训练过程中引入隐状态$\boldsymbol{h}_t$来作为“记忆”，记录以往输入的信息。具体公式为：

$$\boldsymbol{h}_t = f(\boldsymbol{U}_h \boldsymbol{x}_t + \boldsymbol{W}_h \boldsymbol{h}_{t-1} + \boldsymbol{b}_n)$$

其中$\boldsymbol{x}_t$为 t 时刻下输入，$\boldsymbol{h}_t$为 t 时刻下隐状态，$f(\cdot)$为非线性激活函数，如 tanh 或 ReLU 等。其预测$\boldsymbol{y}_t$的公式如下：

$$\boldsymbol{y}_t = softmax(\boldsymbol{W}_y \begin{bmatrix} \boldsymbol{h}_t \\ 1 \end{bmatrix})$$

2.2 代码实现

代码中构建的基础 RNN 模型定义在 myPTRNNModel 类中，其前向方法首先将输入通过一个 Embedding 层，然后通过 RNN 层，再经过一次线性变换后输出。本次实验中一般每 50 次迭代计算一次 Loss，每 50 次迭代进行一次 evaluate，结果经可视化处理之后输出。

Embedding 层一般将维度空间中的对象映射至另一个低维空间中，同时也包括了不同类对象之间的拓扑关系，这意味着 Embedding 后的向量表示不再是离散的、各类之间不具备相关性的（比如 One-hot），而是连续的，各类之间可以计算相似度的向量。

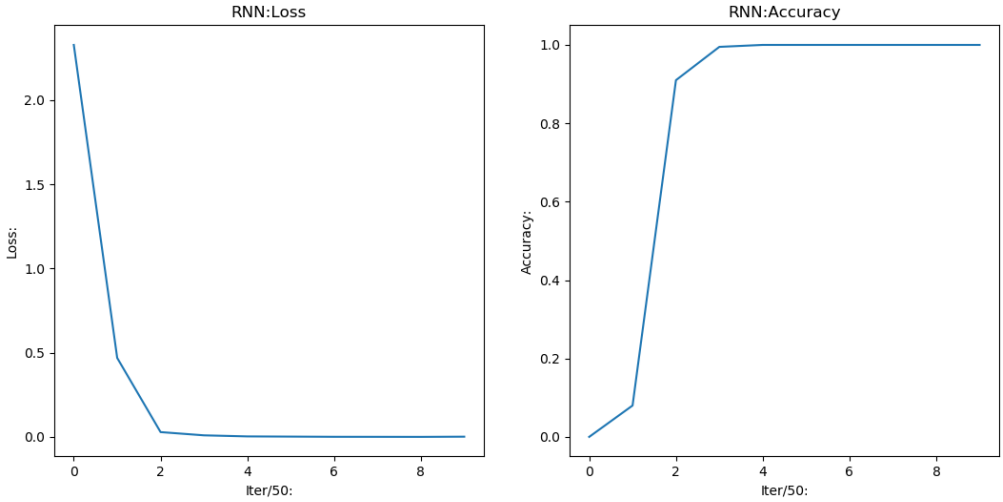
RNN 层的基本原理已经提及，要注意到 RNN 层的输入应该有两个，代码中应该是：

```
Output, self.h = self.rnn(input_num, self.h)
```

但是经常有代码省略等号右侧的 self.h，也即隐状态$\boldsymbol{h}_t$，参阅 Pytorch 官网中 RNN 文档，

其中描述有：若输入中不显式给出 h，则默认对应输入为全零张量，并且示例代码右侧 RNN 输入项包含了 self.h。关于省略 self.h 导致的差异问题我们将在 2.3 中进行讨论。

最终 RNN 模型能够在 200 次迭代内，针对数字长度（MAXLEN）为 10 的数据集达到 Accuracy 为 1.0 的成绩（在 pt_main()函数中实现），具体截图如下：

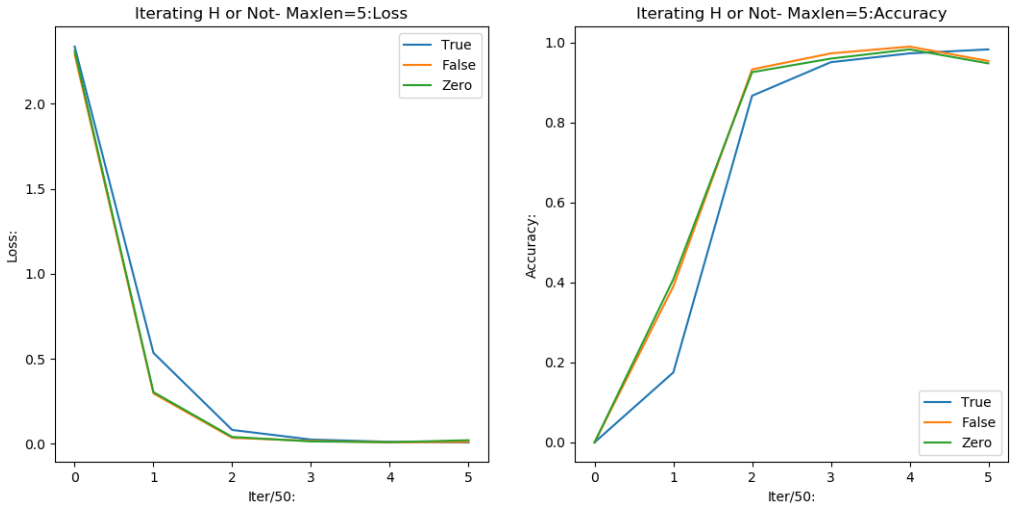


2.3 实验与分析

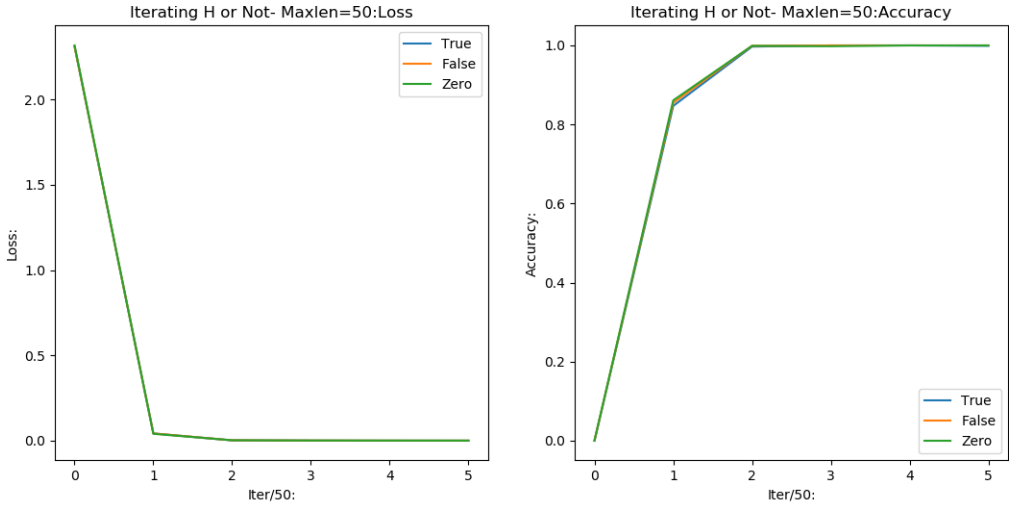
2.3.1 实验（代码右侧 self.h 的取舍问题）

主要问题：在实际实验中，会发现代码中右侧包含 self.h 时，loss 下降更慢，且每次迭代所用时间更多；而右侧不包含 self.h 的 RNN 每次迭代时间异常快，其迭代时间甚至不到包含情况下的四分之一，与只给全零张量下的迭代时间一致。

针对代码疑问，进行测试（见图一、图二），其中 True 为代码包含右侧 self.h 的情况，False 为代码不包含右侧 self.h 的情况，Zero 为代码右侧显式包含全零张量的情况，图一中选取数据集的数字长度为 5，图二中选取的数据集数字长度为 50（为保证结果的有效性，以下结果都是重复实验 5 次的平均值）。



图一 数字长度为 5 情况下的不同迭代方法 Loss/Accuracy 折线图



图二 数字长度为 50 情况下的不同迭代方法 Loss/Accuracy 折线图

其中可以发现，输入不包含 self.h 的 RNN 确实将隐状态 h 默认设为全零张量，甚至在 Maxlen=5 的情况下，其 loss 下降速度与 Accuracy 上升速度都优于包含 self.h 的 RNN 能力。这可能是因为 self.h 记录了多位数值的历史信息，且由于 5 位数字表达的进位信息太少，从而在早期训练中对预测产生了干扰，但能够在后期恢复；而在 Maxlen=50 的情况下，由于每次 50 位数字的进位信息足够多，早期对 RNN 产生影响也小。关于该猜测，我们将在 3.1 节中讨论。

令人奇异的是，甚至每次迭代都不对 self.h 进行更新，并每次用全零张量参与迭代（Zero 情况），也能完成较好的完成任务，并且在 Maxlen=50 时与加入 self.h 进行迭代效果几乎相同。

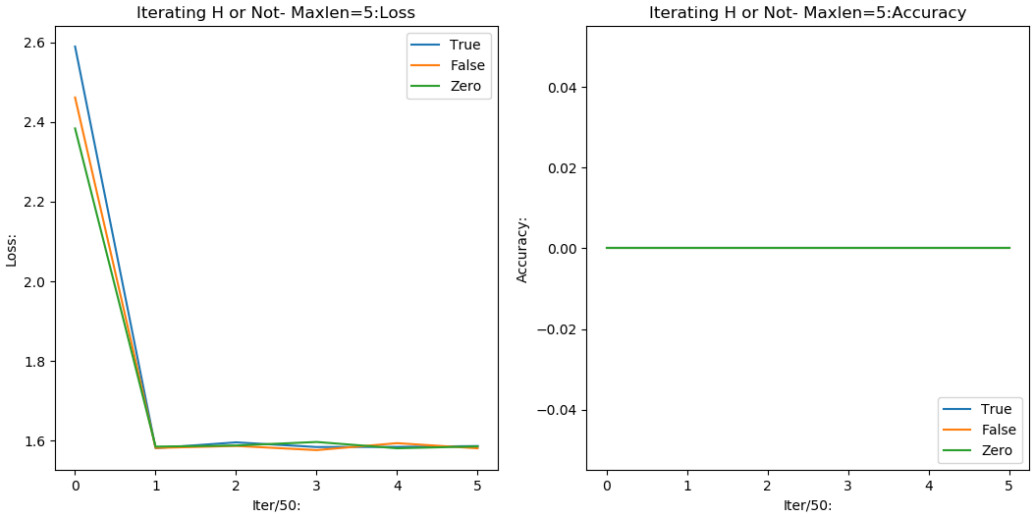
2.3.2 原因分析（为什么不将隐藏变量 self.h 作为输入也能收敛）

可能的原因主要有三个：

1. Embedding 和线性变换本身能力可以完成任务
2. 实验中 RNN 的层数为 2 层，第一层的输出可能让中间结果带有了前序的信息，让第二层即使在缺少隐状态的时刻也能预测准确。
3. Pytorch 的 RNN 特殊实现所导致的。

第一种原因：不成立

图三结果能够说明，这并不是因为 embedding 和线性变换本身的能力足够强到能完成任务所导致的。（图三中是不包含 RNN 时的 Loss 和 Accuracy 变化情况）。

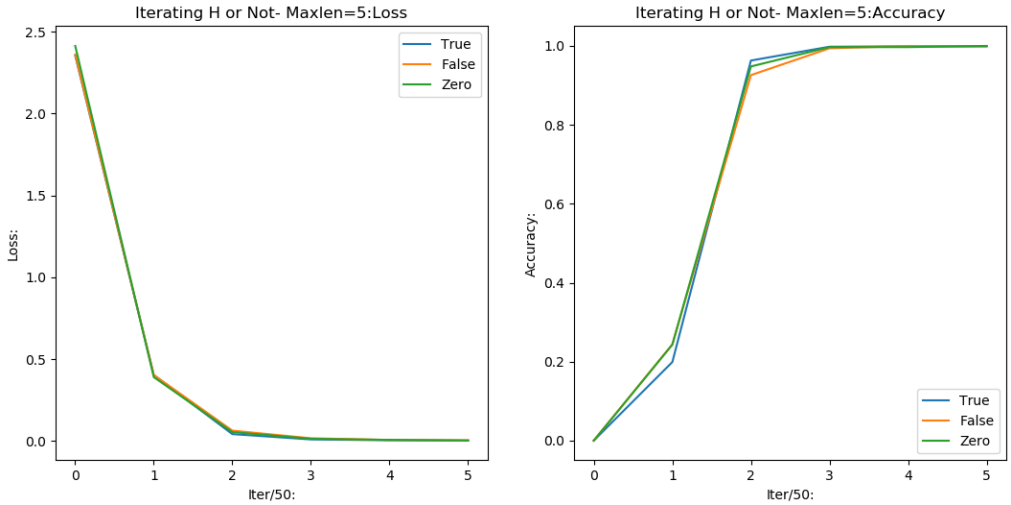


图三：不包含 RNN 下的模型 Loss 和 Accuracy 结果

第二种原因：不成立

图四结果显示，RNN 层数为 1 时，各类情况的 Loss/Accuracy 结果十分相近，而如果原因二成立，各类情况差别应当巨大，因为只有 True 情况才真正保留了前序的信息。

但该结果可能暗示一个猜测：RNN 层数多时，模型对 self.h 的干扰也更为敏感，因此在 RNN 层数为 1 时，预测所受干扰更小，能力随之提升。由于加法任务并不要求长时间的保留输入信息和记忆，只需要上一个片段时刻（短程）的输入即可，其余多位的信息可能成为了干扰。关于该猜测我们将在 3.3 节中讨论。



图四：RNN 层数为 1 时的模型 Loss 和 Accuracy 结果

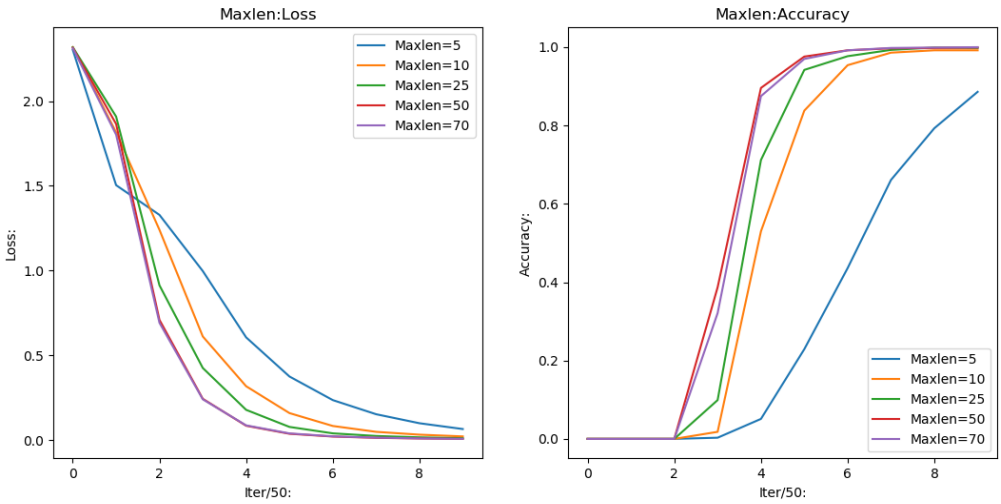
第三个原因需要进一步通过阅读 Pytorch 源码来验证，可能是 RNN 代码中包含了一次早于时序预测的线性变换，使得前序信息被分享至后序，其关键取决于 Pytorch 内 RNN 是否存在其余的信息转递路径。

3. RNN 的优化与分析

3.1 RNN 在数据集不同数字长度（Maxlen）下的性能分析:

3.1.1 实验结果

针对于 RNN 模型（学习率=0.001，RNN 层数=2，隐藏单元初始化为高斯随机采样，非线性激活函数为 ReLU），选择数据集不同的数字长度（5, 10, 25, 50, 70）来进行实验，以下图五为不同学习率下的 RNN 模型结果（下列结果均为五次平均值）:



图五：不同 Maxlen 下的 RNN 模型 Loss 和 Accuracy 结果

3.1.2 分析

图五能够说明，当 Maxlen 较小时，RNN 在训练中对于加法任务的学习迭代速度更慢，Accuracy 折线的上升更缓慢；而当 Maxlen 数值增大到一定程度时（50~70），RNN 在训练中对于加法任务的学习速度不再随 Maxlen 增大而提升。

在 2.3.1 节中曾提及一个猜测：当 Maxlen 较小时，训练数据可能不能很好的反映 RNN 所需要学习的进位信息，更容易受到隐状态（隐藏单元）关于前序多位数字的信息影响。

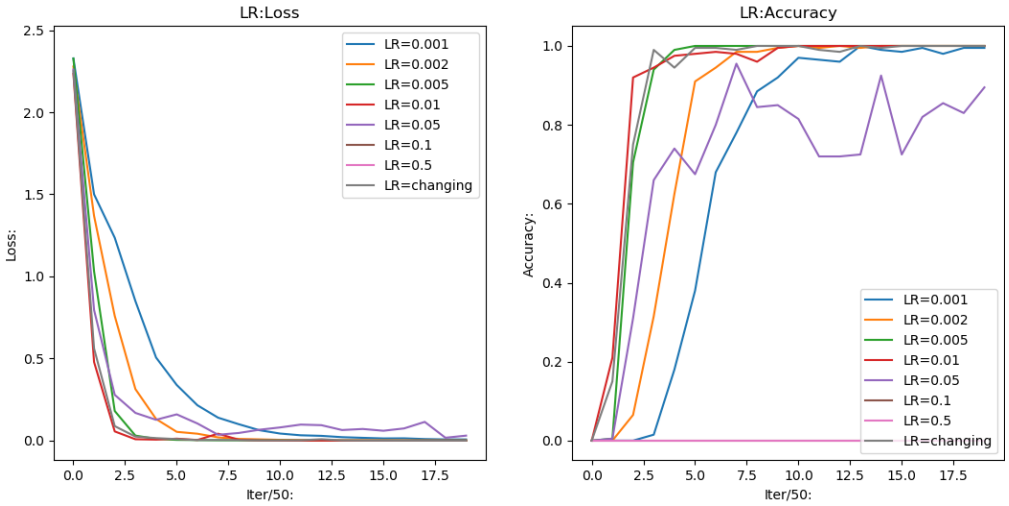
图五的结果与上述猜测并不矛盾：当 Maxlen 较小时，RNN 训练的迭代速度的确要慢于 Maxlen 较大的情况；当 Maxlen 较大时，可能是由于此时数据提供的进位信息已经达到饱和，并且由于长程依赖问题，RNN 受到的干扰也趋于稳定，因此会有学习速度不再受影响的现象。

因此，若要优化 RNN 在加法任务中的训练速度，则应当在 Maxlen 取值较合适（该情况下≈50）时进行训练。

3.2 RNN 在不同学习率设置下的性能分析

3.2.1 实验结果

针对于 RNN 模型（参数 Maxlen=5，RNN 层数=2，隐藏单元初始化为高斯随机采样，非线性激活函数为 ReLU），选择不同的学习率（0.001, 0.002, 0.005, 0.01, 0.05, 0.1, 0.5）来进行实验，以下图六为不同学习率下的 RNN 模型结果：



图六：不同学习率下的 RNN 模型 Loss 和 Accuracy 结果

3.2.2 分析

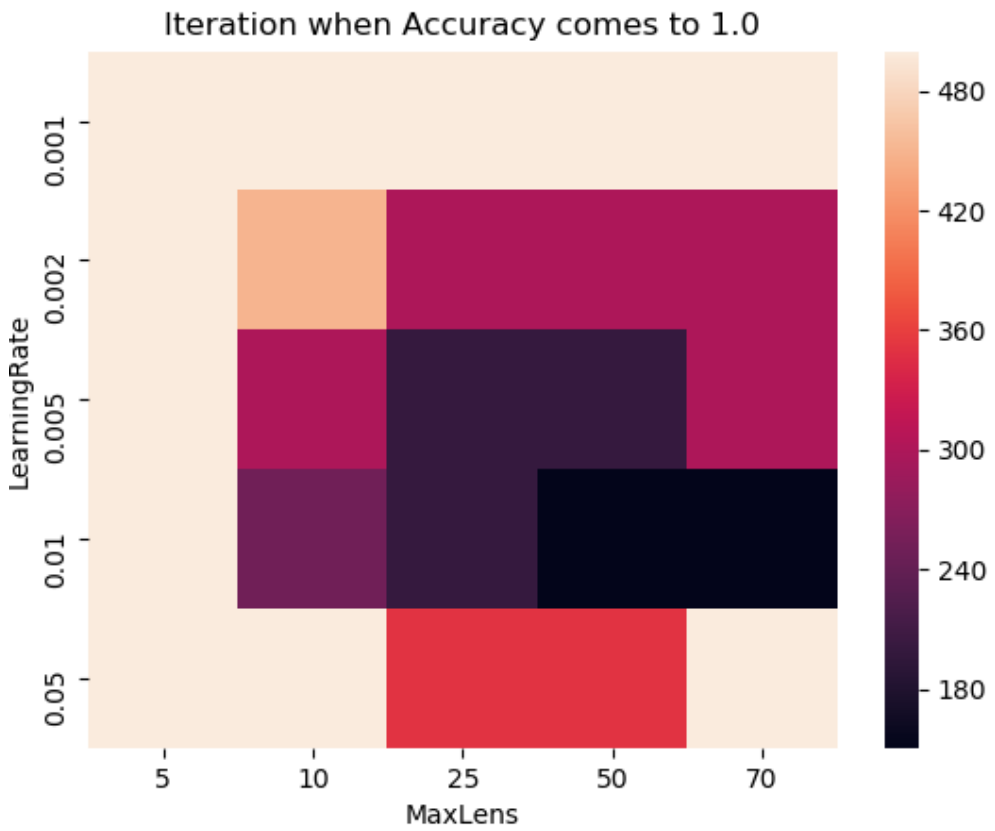
对于神经网络而言，学习率的取值比较重要，如果偏大则模型会难以收敛，如果偏小则拖慢收敛速度。此处可以从图六中看出：对于较大的学习率（LR>0.01），其中 LR=0.05 时 Loss 与 Accuracy 折线剧烈震荡且难以收敛，此时的模型参数也在最优点附近来回震荡，难以进入更优的参数区间，而 LR=0.1/0.5 时，直接发生了梯度爆炸，Loss=Nan；对于较小的学习率而言（LR<0.005），其 Loss 折线的下降趋势与 Accuracy 折现的上升趋势要明显慢于最优曲线，但最终 Loss/Accuracy 结果与其余折线类似，完成了收敛，Accuracy 达到 1.0。

从经验角度出发，学习率需要在训练初始阶段较大来保证较快的收敛速度，并在训练后期收敛到最优点附近时下降以避免震荡的情况。代码这里同样也采用了 Pytorch 的 Scheduler 机制（torch.optim.lr_scheduler.MultiStepLR），对学习率进行动态调整（灰色折线），可看出动态调整下的灰色折线最早达到 1.0 的准确率，Loss 的下降也十分快速。学习率过大时，发生的梯度爆炸可能意味着：在深层模型的反向传播中随层数适度增大学习率可能可以缓解梯度消失问题。

因此，若要优化 RNN 在加法任务中的训练速度，则应当在学习率取值较合适（该情况下≈0.01）或采取动态学习率调整策略情况下（代码中已实现）进行训练。

3.2.3 相关性

为了更好地说明 3.1 与 3.2 的观点，项目对于学习率与 Maxlen 关系制作了热点图（由于每个情况只统计了一次，普遍性有待提升）：



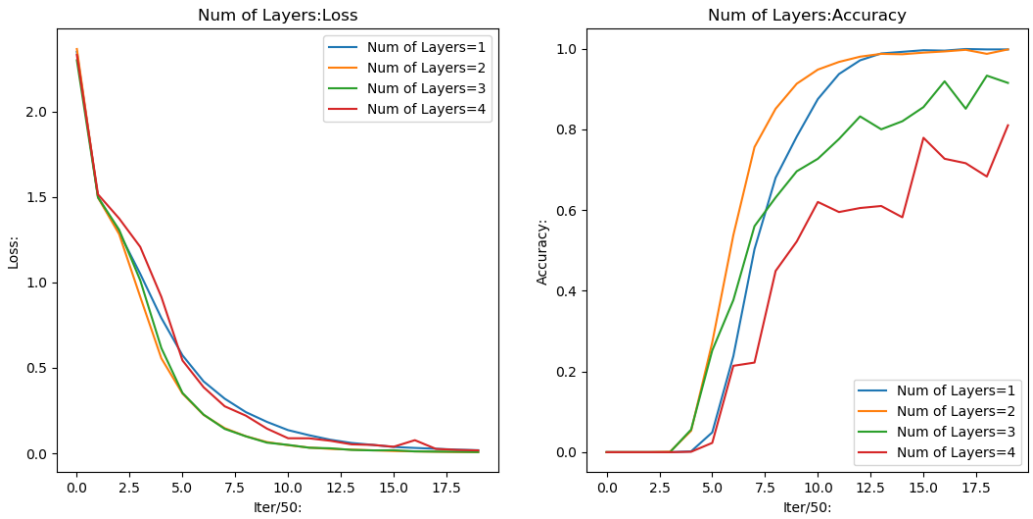
其中颜色的深度表示准确率达到 1.0 的最早迭代次数，颜色越深表示对应条件下迭代速度快，更早达到准确率 1.0；颜色越浅则迭代速度慢，更晚或难以达到准确率 1.0。若在 500 次迭代内没有达到准确率 1.0，则按 500 次迭代计算。
可以看出对于同 Maxlen 有两侧颜色浅中间颜色深。
可以看出对于同学习率有从左至右，颜色基本逐渐变深（0.05/70 是一个例外，可能是因为学习率正好不匹配数据集，难以收敛等）。

可以看出对于学习率在 0.01，Maxlen 在 50-70 左右颜色最深迭代最快。

3.3 RNN 在不同层数下的性能分析

3.3.1 实验结果

针对于 RNN 模型（学习率=0.001，Maxlen=5，隐藏单元初始化为高斯随机采样，非线性激活函数为 ReLU），选择不同的 RNN 层数（1，2，3，4）来进行实验，以下图七为不同 RNN 层数下的 RNN 模型结果（下列结果均为五次平均值）：

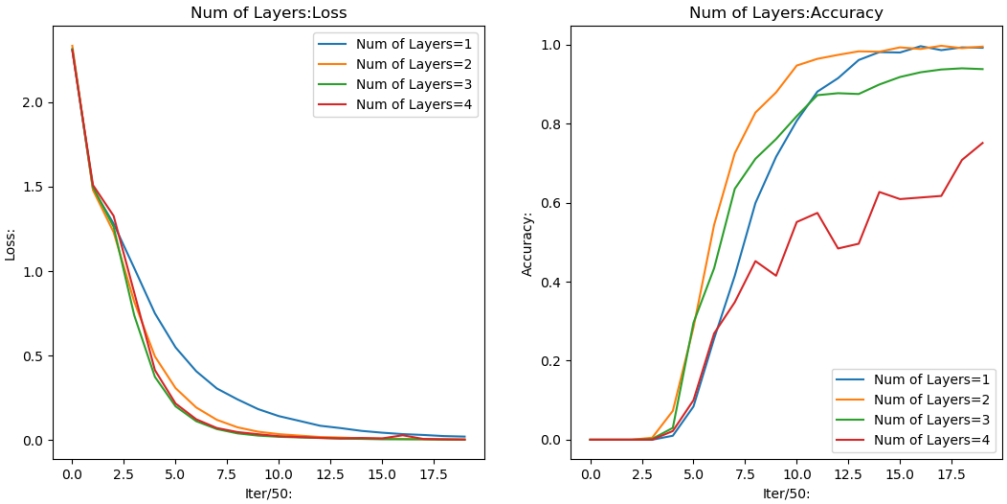


图七：不同 RNN 层数下的 RNN 模型 Loss 和 Accuracy 结果

3.3.2 分析

图七中结果说明：当模型 RNN 层数越大时，其训练时迭代速度更慢，在 loss 下降趋势和 Accuracy 的上升趋势更加缓慢，该结果非常可能是因为模型深度以及随之而来的梯度消失所导致的。

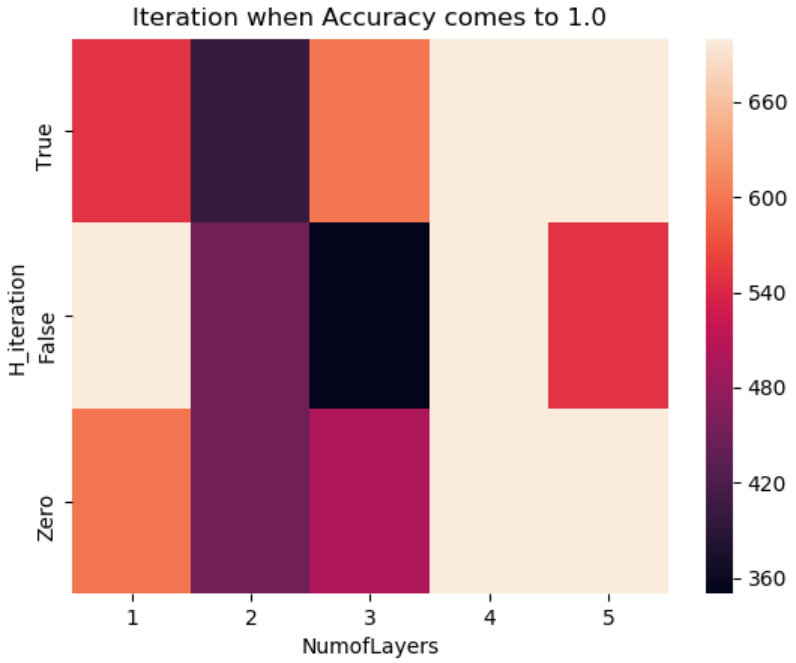
在 2.3.2 中猜测了不同层数下 RNN 受隐藏单元（隐状态）影响不一，且层数越大，受隐藏单元影响也越大。验证该猜测的好方法是取消每次迭代的隐藏单元更新并将全零张量作为隐藏单元输入，比较图七与后者结果的区别即可说明迭代过程中的隐藏单元的影响。下图为全零张量输入（下列结果均为五次平均值）：



可见猜测并不显著正确，或者说猜测中所提及的影响并没有在层数为 1 至 4 时十分明显。

3.3.3 相关性分析

为了更好地说明猜测中提及的影响是否正确，项目对于 RNN 层数与隐藏单元的迭代方法制作了关系热点图（由于每个情况只统计了一次，普遍性有待提升）：



其中颜色的深度表示准确率达到 1.0 的最早迭代次数，颜色越深表示对应条件下迭代速度快，更早达到准确率 1.0；颜色越浅则迭代速度慢，更晚或难以达到准确率 1.0。若在 500 次迭代内没有达到准确率 1.0，则按 500 次迭代计算。

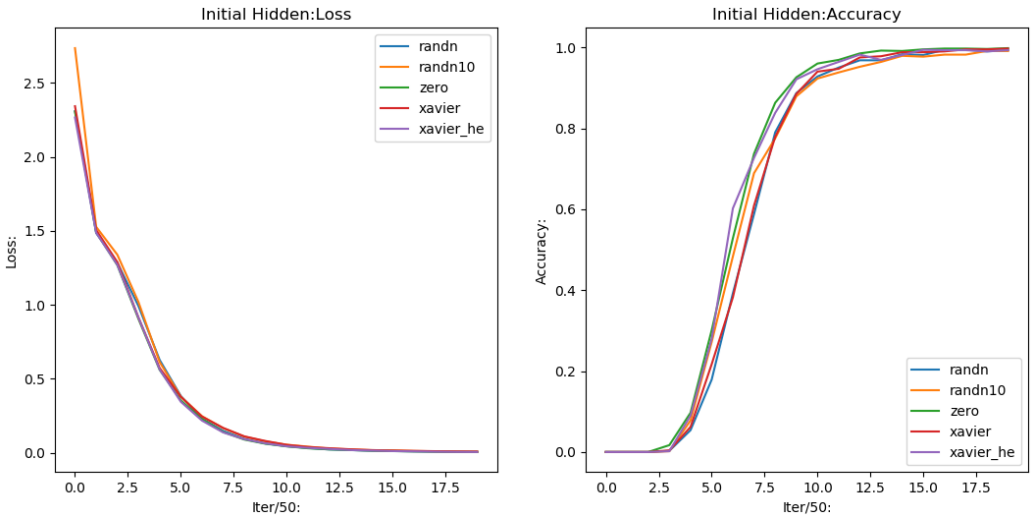
可以从图中看出第一行与第三行的变化情况一致，即在迭代时隐藏单元加入时（H_iteration = True）相对于迭代时用全零张量替换隐藏单元时（H_iteration = Zero）随层数上升的变化相差不多。

因此从结果看来，隐变量迭代的方式对于模型层数而言并没有显著的相关性，这可能是因为模型本身较浅所导致的。

3.4 RNN 在不同隐藏单元初始化下的性能分析

3.4.1 实验结果

针对于 RNN 模型（学习率=0.001，Maxlen=5，非线性激活函数为 ReLU），选择不同的隐藏单元初始化（高斯随机分布初始化，10 倍的高斯随机分布初始化，全零初始化以及广受推崇的 Xavier 初始化（tanh 版与 ReLU 版））来进行实验，以下图八为不同隐藏单元初始化下的 RNN 模型结果（下列结果均为五次平均值）：



图八：不同隐藏单元初始化下的 RNN 模型 Loss 和 Accuracy 结果

3.4.2 分析

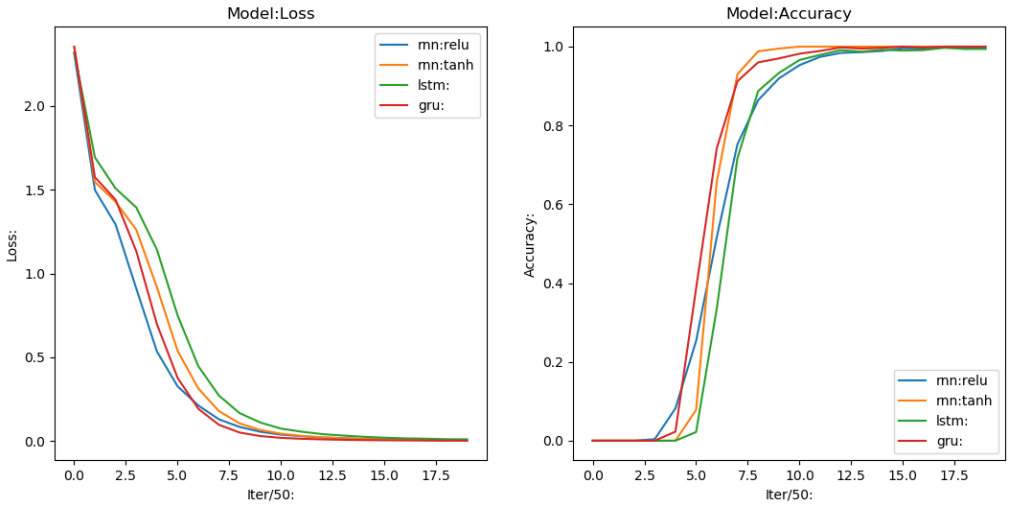
可以从图八中得出：对于不同的隐藏单元初始化方法，在 loss 的优化上没有过多的区别，而在 Accuracy 的优化上，初始化设为全零张量效果可能会更好，甚至部分优于 xavier 初始化，其中 xavier_he 为适用于 ReLU 的 xavier 初始化方法。这可能是因为网络层数不够深，或者任务本身的样本空间与类别空间的分布（密度）没有什么差别，xavier 的作用不是非常明显。

因此，若要优化 RNN 在加法任务中的训练速度，则应当选择全零张量作为隐藏单元的初始化进行训练。

3.5 RNN 在不同非线性激活函数的性能分析以及和 LSTM/GRU 模型的对比

3.5.1 实验结果

针对于 RNN 模型（学习率=0.001，Maxlen=5，隐藏单元初始化为高斯随机采样），选择不同非线性激活函数来进行实验，以下图九为不同非线性激活函数下的 RNN 以及 LSTM 和 GRU 模型结果（下列结果均为五次平均值）：



图九：不同非线性激活函数的 RNN 模型、LSTM 与 GRU 的 Loss 和 Accuracy 结果

3.5.2 分析

能够从图九中看出：在 loss 下降方面，rnn/ReLU 在前期迭代更快，而在后期 GRU 最早收敛。在 Accuracy 上升方面，虽然 GRU 在前期的上升迅速，但后期 rnn/tanh 最早收敛。由于加法任务本身不需要对于历史信息过多的提取，因此理论上有：对于长程依赖问题的缓解不能够让模型对加法任务有实质的性能提升，甚至由于 LSTM 有关于记忆的更复杂处理，可能会导致迭代更慢一点。但 GRU 由于本身能够对记忆进行有效清理，在加法任务中能够更好的选择短期记忆，无论是在 Loss 下降还是 Accuracy 上升都有较好的效果。
由于网络的层数不够深，没有凸显 ReLU 和 tanh 在梯度消失问题上的区别。

因此，若要优化 RNN 在加法任务中的训练速度，则应当在当前情况下选择 RNN/tanh 或 GRU 进行训练。

4 RNN 程序执行

python souce.py

代码中原来的 data.py 最大生成数字长度为 19，在 data.py 中加入了新的 gen_data 函数，现在能够生成 90 位以上的数据