

# Assignment 2

Pattern Recognition and Machine Learning

FUDAN UNIVERSITY @ 2020 SPRING

2020 年 5 月 6 日

## 1 问题描述

在这次作业中，需要设计一个基于 RNN 的神经网络来模拟加法器。首先，学习编写基于 Tensorflow 2.0(TF) 或 PyTorch(PT) 的源代码，完成其余的代码。然后，修改数字的长度，分析模型性能，并对模型进行改进。

## 2 RNN

循环神经网络 (Recurrent neural network: RNN) 是神经网络的一种，通过使用带自反馈的神经元，能够处理任意长度的时序数据，可以应用到很多不同类型的机器学习任务。

给定一个输入序列  $X_{1:T} = X_1, X_2, \dots, X_i, \dots, X_T$ ，循环神经网络通过下面公式更新带反馈边的隐藏层的活性值  $h_t$ ：

$$h_t = f(h_{t-1}, x_t) \quad (1)$$

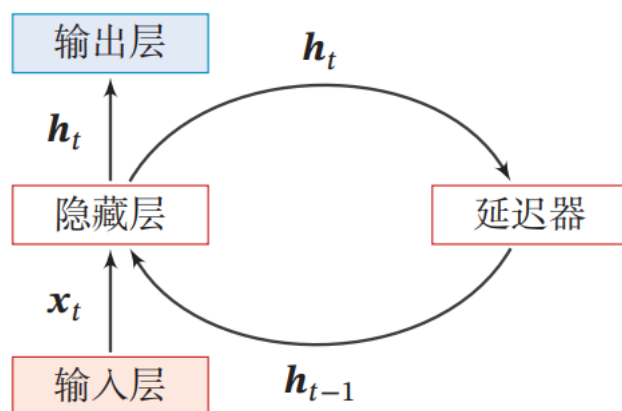


图 1: 循环神经网络

我们使用 RNN 模型实现一个简单的加法器。以二进制加法器为例，任何一个整数都可以用一个二进制串来表示，给定两个二进制串，我们希望生成表示其和的二进制串。一个二进制串可以看成是一个序列，这可以用 RNN 来搭建模型。

$$\begin{array}{r} \text{11111111} \quad =-1 \\ + \quad \text{11111110} \quad =-2 \\ \hline = \quad \text{11101} \end{array}$$

图 2: 二进制加法器

二进制加法器从左向右开始计算，通过两个运算数对应位上二进制数来得到新的二进制数。但考虑到运算溢出的问题，图 2 上彩色方框中的 1 表示的是运算溢出后的“携带位”，需要将其传递给下一位的运算。由于 RNN 具有短期记忆能力，相当于存储装置，上一个时刻的隐含特征保存这个“携带位”信息，这样当前位置的运算就可以捕获到前面运算溢出得到的“携带位”。

### 3 模型

该模型共有 4 层: 一个嵌入层 (embedding layer)，两个隐藏层 (hidden layer) 和一个全连接层 (fully-connected layer)。

```
class myPTRNNModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.embed_layer = nn.Embedding(length, 32) #embedding layer
        self.rnn = nn.RNN(64, 64, 2, batch_first=True) #hidden layer
        self.dense = nn.Linear(64, length) #fully-connected layer

    def forward(self, num1, num2):
        num1=self.embed_layer(num1)
        num2=self.embed_layer(num2)
        input = torch.cat((num1, num2), 2)
```

```
output,h_n=self.rnn(input)
logits=self.dense(output)
return logits
```

---

## 3.1 初始化

### 3.1.1 嵌入层

**nn.Embedding(num\_embeddings, embedding\_dim)**

- num\_embeddings (int) - 嵌入字典的大小
- embedding\_dim (int) - 每个嵌入向量的大小

这是一个矩阵类，里面初始化了一个随机矩阵。把进行加法运算的数看作是一个字符串序列，矩阵的长是序列的大小，宽是用来表示序列中每个元素的属性向量，向量的维度根据想要表示的元素的复杂度而定。类实例化之后可以根据序列中元素的下标来查找元素对应的向量。

### 3.1.2 隐藏层

**nn.RNN(input\_size,hidden\_size,num\_layers)**

- input\_size - 输入序列的大小
- hidden\_size - 隐层的特征数量
- num\_layers - RNN 的层数

对输入序列中每个元素，RNN 每层的计算公式为

$$h_t = \tanh(w_{ih}x_t + b_{ih} + w_{hh}h_{t-1} + b_{hh}) \quad (2)$$

$h_t$  是时刻  $t$  的隐状态。 $x_t$  是上一层时刻  $t$  的隐状态，或者是第一层在时刻  $t$  的输入。

### 3.1.3 全连接层

**nn.Linear(in\_features, out\_features)**

- in\_features - 每个输入样本的大小
- out\_features - 每个输出样本的大小

输入数据做线性变换： $y = Ax + b$

## 3.2 Forward

forward 定义了每次执行的计算步骤，在所有的子类中都需要重写这个函数。

1. 输入为长度为  $num\_embeddings = 10$  的两个字符串序列 num1 和 num2，经过嵌入层的变换，输出两个大小为  $embedding\_dim = 32$  的嵌入向量。然后对输出的两个嵌入向量进行拼接，得到大小为  $2 \times embedding\_dim = 64$  的向量 input。
2. 将 input 作为隐藏层的输入，得到 RNN 最后一层的输出 output 和最后一个时刻隐状态  $h_n$ 。
3. 将大小为  $in\_features = 64$  的向量 output，经过全连接层的线性变换，得到大小为  $out\_features = 10$  的向量 logits（运算和的字符串序列）。

## 4 模型性能

### 4.1 输入序列长度

在第 3 节中定义的 RNN 模型，在加数位数分别为 10, 20, 50, 100，训练次数为  $step = 1000$ ，学习率  $lr = 0.001$  的情况下，在测试集上均能达到准确率  $acc = 1.00$ 。在训练集上的收敛情况如图 3 所示，随着加数位数的增加，模型的收敛速度变慢。分析如下，

1. 由于各个加数的字符串序列长度不同，我们采用 padding 方法，将每个字符串序列通过补 0 扩充到一样的长度，这样会导致模型对样本的表示通过了非常多无用的字符，这样得到的信息就会有误差。
2. 当嵌入向量的大小  $embedding\_dim$  一定时，输入序列的长度越大，通过嵌入层之后，得到的嵌入向量可能无法完全捕捉到原序列的有效信息。

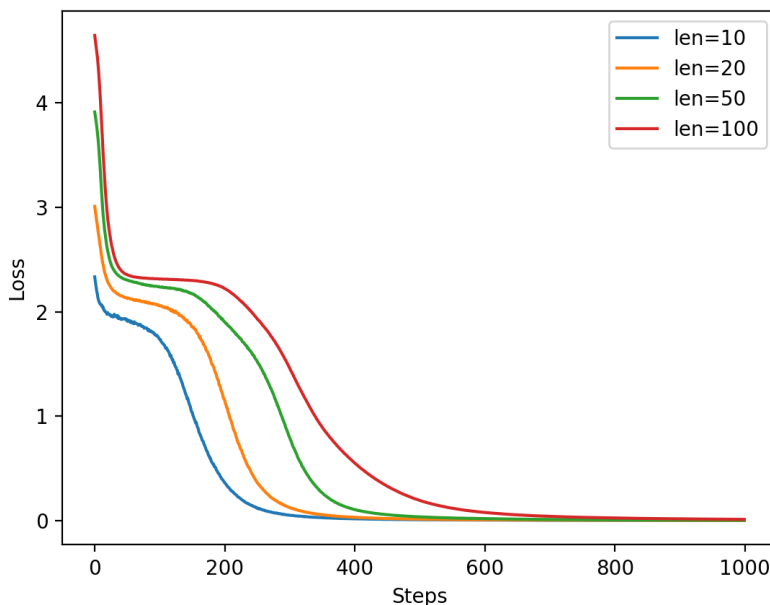


图 3: 输入序列长度不同的 RNN 模型 loss 曲线

## 4.2 学习率

如图 4所示, 在输入序列长度分别为 10,20,50 情况下, 学习率  $lr = 0.001$  时, 模型收敛速度最慢,  $lr = 0.01$  和  $lr = 0.1$  时, 模型收敛速度明显变快。

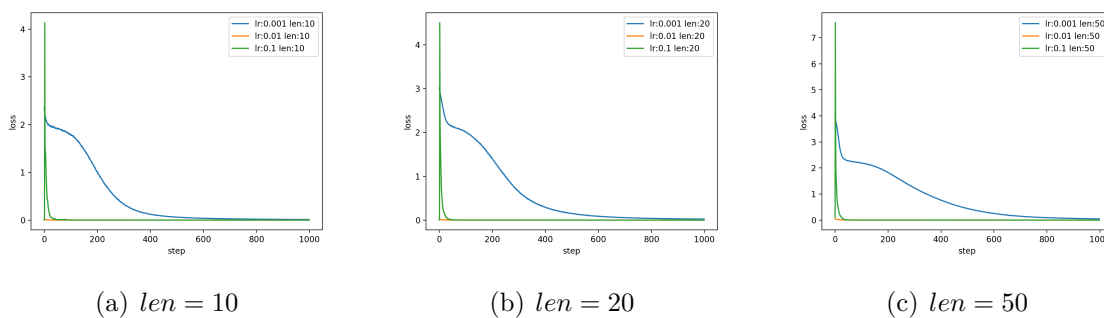
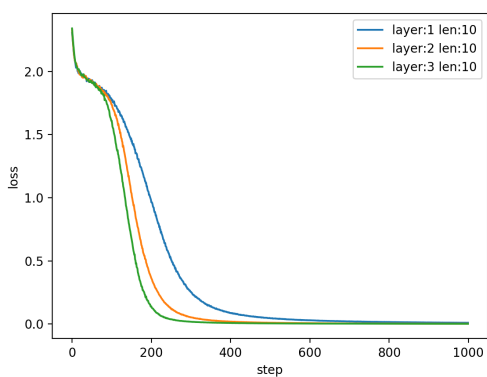


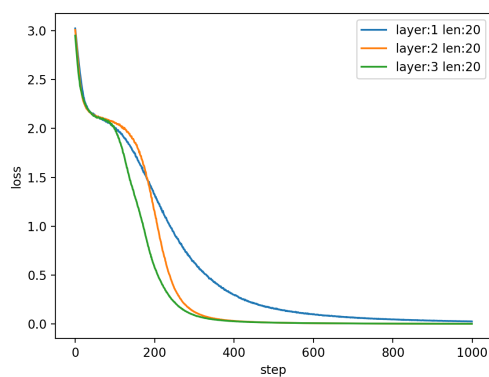
图 4: 学习率不同的 RNN 模型 loss 曲线

## 4.3 模型层数

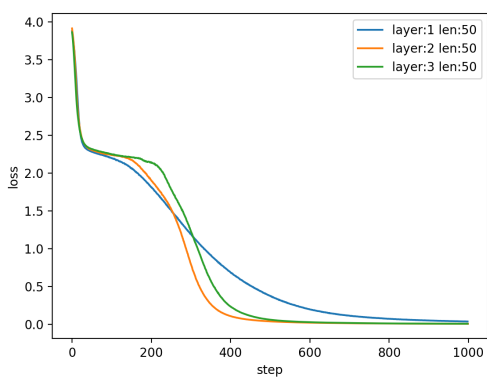
随着网络层数的加深, 网络的表达能力以及抽象能力也会随着提高, 但网络并非越深越好。在输入序列长度较小  $len = 10/20$  时, 模型层数越多, 模型收敛速度越快; 在输入



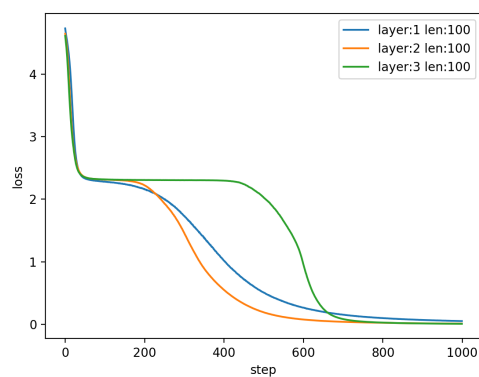
(a)  $len = 10$



(b)  $len = 20$



(c)  $len = 50$



(d)  $len = 100$

图 5: 模型层数不同的 RNN 模型 loss 曲线

序列长度较大  $len = 50/100$  时，模型层数  $layer = 2$  时，模型收敛速度最快（图 5）。原因可能是不断加深神经网络会带来一些负面问题，如，

1. 梯度消失，梯度爆炸问题。梯度问题产生的根本原因就是源于网络结构太深。
2. 过拟合问题。随着网络层数的加深，参数变多，神经网络的拟合能力变得很强，这也就意味着其表达出来的函数会更复杂，而如果对于简单问题如加法器，采用过于复杂的函数，容易导致过拟合。

#### 4.4 LSTM/GRU

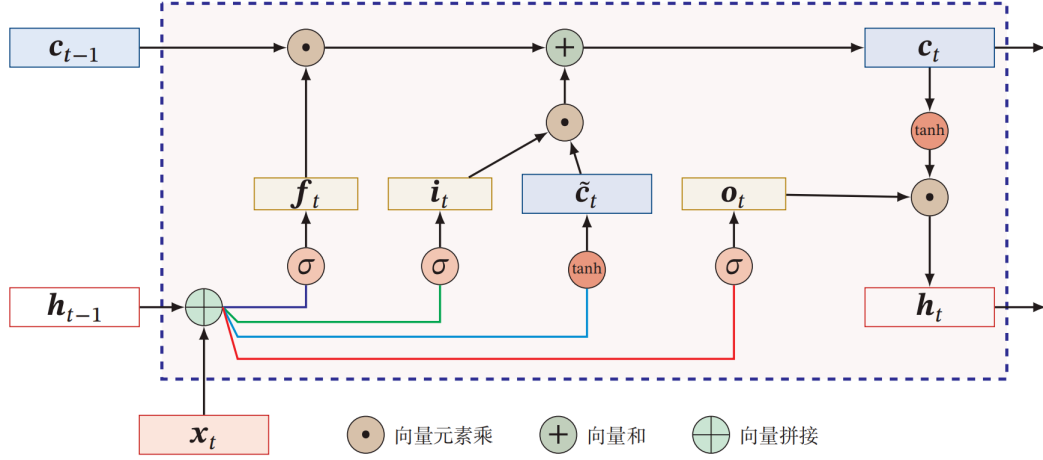


图 6: LSTM 网络的循环单元结构

长短期记忆网络 (Long Short-Term Memory Network, LSTM, 图 6) 是循环神经网络的一个变体, 可以有效地解决简单循环神经网络的梯度爆炸或消失问题。LSTM 网络主要改进在引入新的内部状态和门控机制。

门控循环单元 (Gated Recurrent Unit, GRU, 图 7) 网络是一种比 LSTM 网络更加简单的循环神经网络。GRU 网络引入门控机制来控制信息更新的方式。和 LSTM 不同, GRU 不引入额外的记忆单元。

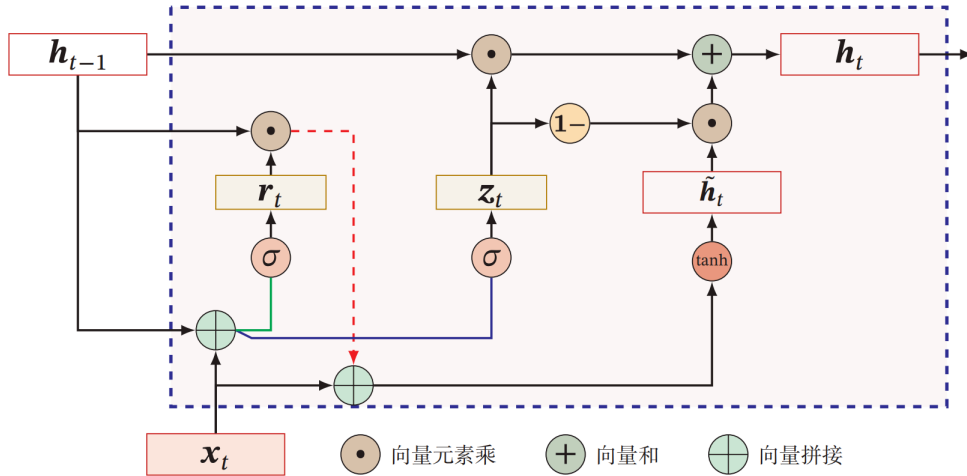
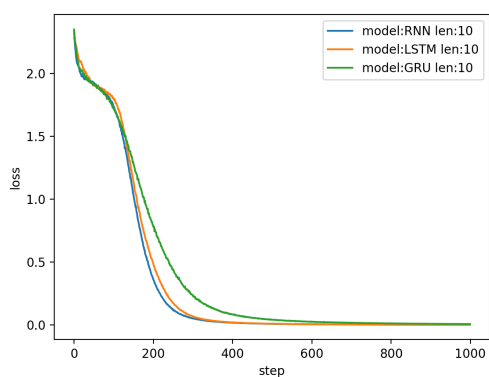


图 7: GRU 网络的循环单元结构

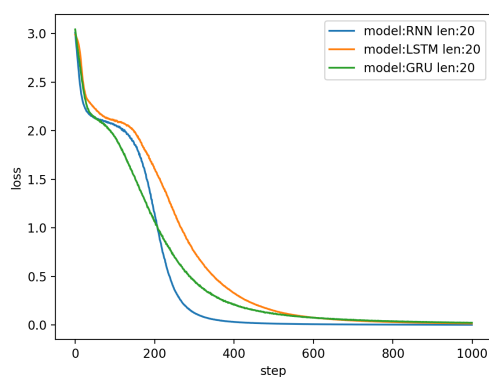
在输入序列长度  $len = 10$  时, 模型收敛速度:  $RNN > LSTM > GRU$ ; 在输入序列长度  $len = 20/50/100$  时, 模型收敛速度:  $RNN > GRU > LSTM$ 。且随着输入序列长度

的增大, LSTM 和 GRU 的收敛速度差异越来越大, 而 RNN 和 GRU 的收敛速度差异越来越小 (图 8)。分析原因如下,

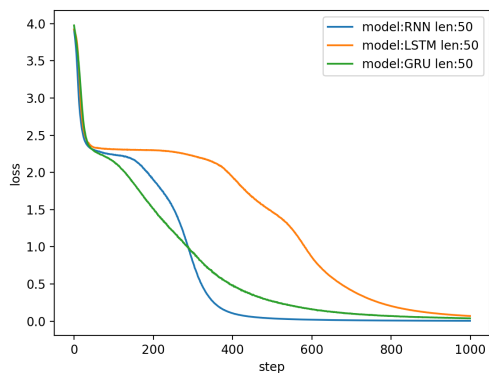
1. LSTM 及其变种 GRU 通过有选择地加入新的信息, 并有选择地遗忘之前累积的信息, 来改善 RNN 的长程依赖问题。但由于加法器当前位的输出, 仅与两个加数当前位, 以及上一时刻的进位 (“携带位”) 有关, 与更早时刻的 “携带位” 无关, 不存在长程依赖问题, 所以 RNN 模型在  $len$  不同的情况下, 始终能保持较好的性能。
2. 对于 LSTM 与 GRU 而言, 由于 GRU 参数更少, 收敛速度更快, 因此其实际花费时间要少很多, 这可以大大加速了我们的迭代过程。



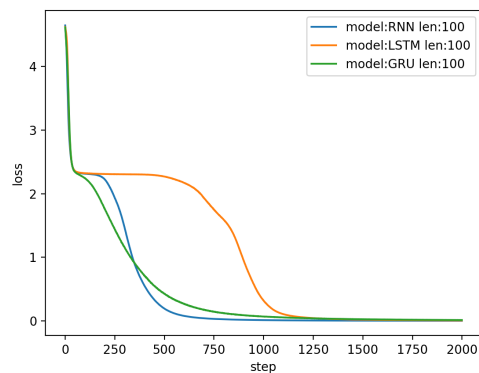
(a)  $len = 10$



(b)  $len = 20$



(c)  $len = 50$



(d)  $len = 100$

图 8: RNN/LSTM/GRU 模型 loss 曲线



## 5 总结

通过上述实验可知，要实现一个基于 RNN 的加法器，选择 RNN 作为隐藏层，当模型层数为 2，学习率为 0.01 时候，能在不同的输入序列长度下保持较好的训练效果。

## 6 代码执行方式

---

```
python source.py
```

---