循环神经网络模型的初步学习

December 20, 2020

以前只知道循环神经网络 (Recurrent Neural Network) 可以用于对序列建模，不了解其内部结构和计算上的细节。经过本学期各种课程的学习，虽然老师在相关课程上必不可少的讲到循环神经网络，但是我仍是一知半解，所以想借本课程的大作业对循环神经网络做一个比较基础但是详细的学习。本报告的学习内容主要来自 [1]。

首先，由于我对于神经网络与深度学习的不熟悉，导致我对老师一上来就给出的展开后的计算图 (unfolded computational graph) 产生了困惑。循环神经网络中的一层到底是由几个 Recurrent Unit 组成的？我不只一次发出这样的疑问。但是，显然按着循环神经网络的定义，就是循环就体现在 Recurrent Unit 与自己连接，从而达到在输入的不同位置之间共享参数的效果。书中也提到了循环神经网络相较于传统的全连接前馈网络，可以在几个时刻内共享相同的权重，从而不需要分别学习在不同位置上的每个规则。

可以看出这种共享参数的思想与卷积神经网络比较类似，而事实上，确实有在一维时间序列上进行卷积的神经网络，被称为时延神经网络 (time-delay neural network,TDNN)，可以让一个时间的部分时段共享参数，相较于展开的循环连接来说是比较浅层的。

循环神经网络的定义很泛泛，只要满足循环连接的网络似乎都可以称之为循环神经网络，中间的激活函数和梯度更新方式都可以使用各种在传统全连接神经网络中的方法。循环连接的方式也有很多种，其中有 hidden-to-hidden 也有 output-to-hidden 的。output-to-hidden 的循环神经网络虽然表示能力弱于 hidden-to-hidden 的，但是它可以使用前一个时刻目标的真实值作为计算当前时刻状态的输入对当前时刻的隐层进行求导 (teacher forcing)，而无需像 hidden-to-hidden 的循环神经网络先计算前一个时刻的状态，作为计算当前按状态的输入导致天生的顺序性 (这一点在老师在讲注意力机制时也提到过)。

在实际情况中，循环神经网络通常在 minibatches of sequences 上训练，而且每个 minibatch 中的 sequence 长度不同。但是，这里计算循环神经网络的梯度时只考虑了单个序列输入的情况。

下面是 hidden-to-hidden 的循环神经网络的前馈公式，和损失函数。这里假定使用双曲正切 (tanh) 激活函数，网络的输出是离散的，来进行多分类，如用于预测词或字符。在网络的输出层进行 Softmax，损失函数就用多分类常用的交叉熵函数。这里的模型是 n-to-n 的 RNN 即将一个输入序列映射到相同长度的输出序列。

一维 [!htb]

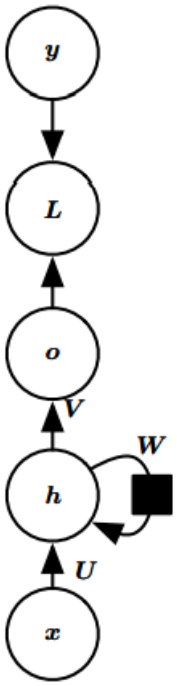

Figure 1: RNN 未进行展开的运算图

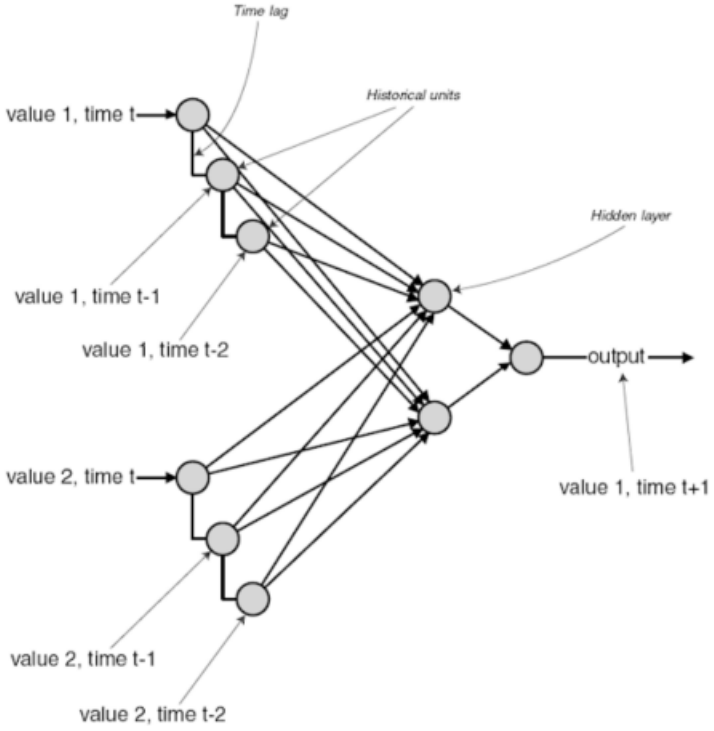

Figure 2: TDNN 的运算图

$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}, \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}), \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}, \\
\hat{\boldsymbol{y}}^{(t)} &= \mathrm{Softmax}(\boldsymbol{o}^{(t)}), \\
\mathrm{Loss}^{(t)}(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}) &= \sum_i \mathrm{Loss}(\boldsymbol{y}_i^{(t)}, \hat{\boldsymbol{y}}_i^{(t)}) \\
&= -\sum_i \boldsymbol{y}_i^{(t)} \log \hat{\boldsymbol{y}}_i^{(t)}, \\
\mathrm{Loss}(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)}) &= \sum_{t=1}^{\tau} \mathrm{Loss}^{(t)}(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)})^{(t)}
\end{aligned}
$$

其中的 bias 向量 $\boldsymbol{b}_{m\times 1}$ 和 $\boldsymbol{c}_{q\times 1}$ 连同权重矩阵 $\boldsymbol{U}_{m\times p}$、$\boldsymbol{V}_{q\times m}$ 和 $\boldsymbol{W}_{m\times m}$，分别对应于输入到隐藏、隐藏到输出和隐藏到隐藏的连接。其中 m 为模型维数，p 为输入维数，q 为输出维数。

假如还想全连接前馈神经网络那样使用单纯使用连锁规则进行反向传播会

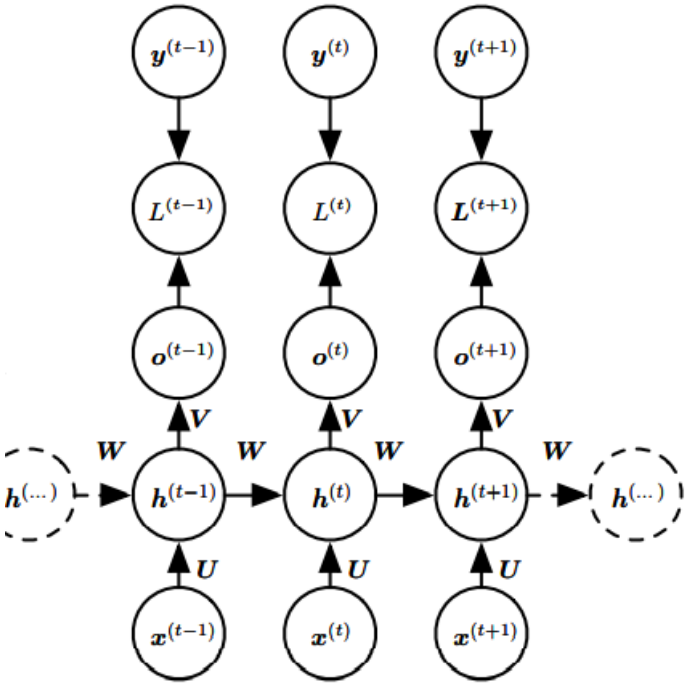

Figure 3: hidden-to-hidden 的 RNN 展开的运算图

得到下面的公式，很显然可以看出，对于 $\boldsymbol{c}_{q\times 1}$ 和 $\boldsymbol{V}_{q\times m}$ 的梯度计算 $\nabla_{\boldsymbol{c}}L$ 和 $\nabla_{\boldsymbol{V}}L$ 仍可以正常计算，而对于 $\boldsymbol{U}_{m\times p}$、$\boldsymbol{W}_{m\times m}$ 和 $\boldsymbol{b}_{m\times 1}$ 的梯度 $\nabla_{\boldsymbol{W}}L$、$\nabla_{\boldsymbol{U}}L$ 和 $\nabla_{\boldsymbol{b}}L$ 就像刚刚说的一样，涉及到全部中间状态 $\boldsymbol{h}_{(t)}$ 对损失函数的求导。但是对于 hidden-to-hidden 的循环神经网络 $\boldsymbol{h}_{(t)}$ 是由前一个时刻的状态 $\boldsymbol{h}_{(t-1)}$ 得到的。因此，想要得到 $\boldsymbol{h}_{(t-1)}$ 就需要递归地计算 $\boldsymbol{h}_{(t)}$。

$$
\begin{aligned}
\nabla_{\boldsymbol{c}}L &= \sum_t \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{c}}\right)^{\top} \nabla_{\boldsymbol{o}^{(t)}} L = \sum_t \nabla_{\boldsymbol{o}^{(t)}} L, \\
\nabla_{\boldsymbol{b}}L &= \sum_t \left(\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{b}^{(t)}}\right)^{\top} \nabla_{\boldsymbol{h}^{(t)}} L, \\
\nabla_{\boldsymbol{V}}L &= \sum_t \sum_i \left(\frac{\partial L}{\partial o_i^{(t)}}\right) \nabla_{\boldsymbol{V}} o_i^{(t)} = \sum_t (\nabla_{\boldsymbol{o}^{(t)}} L) \boldsymbol{h}^{(t)^{\top}}, \\
\nabla_{\boldsymbol{W}}L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{\boldsymbol{W}^{(t)}} h_i^{(t)}, \\
\nabla_{\boldsymbol{U}}L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{\boldsymbol{U}^{(t)}} h_i^{(t)},
\end{aligned}
$$

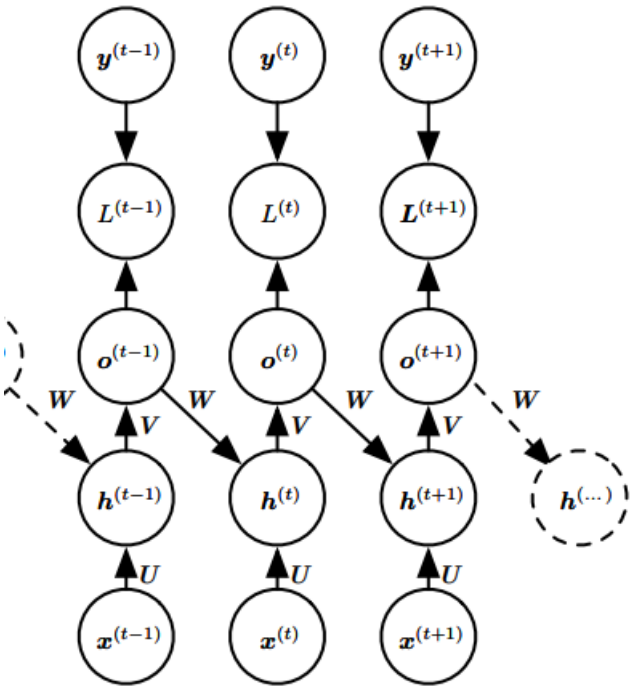

Figure 4: output-to-hidden 的 RNN 展开的运算图

接下来重点就是在于如何求得每个时刻的中间状态的导数 $\nabla_{\boldsymbol{h}^{(t)}} L$。这时可以从展开的计算图入手，一步一步通过连锁规则求解。其实与普通的梯度计算只多了一个展开，通过获得首项和递推公式从后向前计算 $\nabla_{\boldsymbol{h}^{(t)}} L$，就可以很清晰地得到所有的中间状态导数，从而完成参数的导数计算。

首先，对于序列的中间状态，可以看出，最后一个中间状态关于损失函数的导数 $\nabla_{\boldsymbol{h}^{(\tau)}} L$ 只与输出有关，而其他中间状态关于损失函数的导数 $\nabla_{\boldsymbol{h}^{(t-1)}} L$ 不仅与输出有关，还与下一时刻的中间状态关于损失函数的导数有关。那么首先计算最后一个状态。对于交叉熵函数的求导很容易得出。

$$\frac{\partial L}{\partial \hat{\boldsymbol{y}}_i^{(t)}} = -\frac{\boldsymbol{y}_i^{(t)}}{\hat{\boldsymbol{y}}_i^{(t)}}$$

而 Softmax 函数的导数计算后进一步代入可得

$$\frac{\partial L}{\partial \boldsymbol{o}_i^{(t)}} = -\boldsymbol{y}_i^{(t)}(1 - \hat{\boldsymbol{y}}_i^{(t)})$$

其中 $\boldsymbol{y}_i^{(t)}$ 为 0-1 二值，并且其中只有正确的那个为 1。接下来，输出向量 $\boldsymbol{o}^{(t)}$ 关于中间状态 $\boldsymbol{h}^{(t)}$ 的导数为

$$\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h}^{(t)}} = \boldsymbol{V}^T$$

通过以上公式即可求得最后的中间状态的导数 $\nabla_{\boldsymbol{h}^{(\tau)}} L$ 为

$$\begin{aligned}\nabla_{\boldsymbol{h}^{(\tau)}} L &= \frac{\partial L}{\partial \boldsymbol{o}^{(\tau)}} \frac{\partial \boldsymbol{o}^{(\tau)}}{\partial \boldsymbol{h}^{(\tau)}} \\ &= \boldsymbol{V}^\top \nabla_{\boldsymbol{o}^{(\tau)}} L\end{aligned}$$

接下来计算导数 $\nabla_{\boldsymbol{h}^{(t-1)}} L$，首先根据展开后的计算图有两个反向传播的来源

$$\nabla_{\boldsymbol{h}^{(t-1)}} L = \Big(\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}\Big)^\top (\nabla_{\boldsymbol{h}^{(t)}} L) + \Big(\frac{\partial \boldsymbol{o}^{(t-1)}}{\partial \boldsymbol{h}^{(t-1)}}\Big)^\top (\nabla_{\boldsymbol{o}^{(t-1)}} L)$$

其中等式右端第一项是从 t-1 时刻的 Loss 函数得来的可以直接由上面的公式计算，而等式右端的第二项则是通过下一时刻的状态导数通过连锁规则得来。现在只需得到 $\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}$。

$$\begin{aligned}\boldsymbol{h}_i^{(t)} &= \tanh\Big(\boldsymbol{b}_i + \sum_{j=1}^{m} w_{ij} \boldsymbol{h}_j^{(t-1)} + \sum_{j=1}^{p} u_{ip} \boldsymbol{x}_p^t\Big), \\ \frac{\partial \boldsymbol{h}_i^{(t)}}{\partial \boldsymbol{h}_i^{(t-1)}} &= \Big[1 - \tanh^2\Big(\boldsymbol{b}_i + \sum_{j=1}^{m} w_{ij} \boldsymbol{h}_j^{(t-1)} + \sum_{j=1}^{p} u_{ip} \boldsymbol{x}_p^t\Big)\Big] \times w_{ij} \\ &= \Big[1 - (\boldsymbol{h}_i^{(t)})^2\Big] \times w_{ij} \\ \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}_i^{(t-1)}} &= \sum_{i=1}^{m} \frac{\partial \boldsymbol{h}_i^{(t)}}{\partial \boldsymbol{h}_i^{(t-1)}}, \\ \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} &= \Big(\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}_1^{(t-1)}} \cdots \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}_m^{(t-1)}}\Big)^T \\ &= \boldsymbol{W}^T \operatorname{diag}\Big(1 - (\boldsymbol{h}^{(t)})^2\Big)\end{aligned}$$

其中 $\operatorname{diag}\Big(1 - (\boldsymbol{h}^{(t+1)})^2\Big)$ 表示包含元素 $1 - (h_i^{(t+1)})^2$ 的对角矩阵。因此有

$$\begin{aligned}\nabla_{\boldsymbol{h}^{(t-1)}} L &= \Big(\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}\Big)^\top (\nabla_{\boldsymbol{h}^{(t)}} L) + \Big(\frac{\partial \boldsymbol{o}^{(t-1)}}{\partial \boldsymbol{h}^{(t-1)}}\Big)^\top (\nabla_{\boldsymbol{o}^{(t-1)}} L) \\ &= \boldsymbol{W}^\top (\nabla_{\boldsymbol{h}^{(t)}} L) \operatorname{diag}\Big(1 - (\boldsymbol{h}^{(t)})^2\Big) + \boldsymbol{V}^\top (\nabla_{\boldsymbol{o}^{(t-1)}} L),\end{aligned}$$

至此可以给出各个权值的用于反向传播的导数

$$\nabla_{\boldsymbol{b}} L = \sum_t \Big(\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{b}^{(t)}}\Big)^\top \nabla_{\boldsymbol{h}^{(t)}} L = \sum_t \mathrm{diag}\Big(1 - \big(\boldsymbol{h}^{(t)}\big)^2\Big) \nabla_{\boldsymbol{h}^{(t)}} L,$$

$$\begin{aligned}\nabla_{\boldsymbol{W}} L &= \sum_t \sum_i \Big(\frac{\partial L}{\partial h_i^{(t)}}\Big) \nabla_{\boldsymbol{W}^{(t)}} h_i^{(t)} \\ &= \sum_t \mathrm{diag}\Big(1 - \big(\boldsymbol{h}^{(t)}\big)^2\Big) (\nabla_{\boldsymbol{h}^{(t)}} L) \boldsymbol{h}^{(t-1)^\top},\end{aligned}$$

$$\begin{aligned}\nabla_{\boldsymbol{U}} L &= \sum_t \sum_i \Big(\frac{\partial L}{\partial h_i^{(t)}}\Big) \nabla_{\boldsymbol{U}^{(t)}} h_i^{(t)} \\ &= \sum_t \mathrm{diag}\Big(1 - \big(\boldsymbol{h}^{(t)}\big)^2\Big) (\nabla_{\boldsymbol{h}^{(t)}} L) \boldsymbol{x}^{(t)^\top},\end{aligned}$$

从而可以给出 n-to-n 的 hidden-to-hidden 的 RNN 的权重梯度计算的算法框架。

算法 1 计算所有参数梯度

输入: 要求梯度的参数集合 T
展开的运算图 G
要被求导的参数 z
输出: 所有要求的参数的梯度

```
让 G' 为 G 中只是 z 的祖先的节点构成的子图
[初始化]grad_table，参数对自身求导为 1
grad_table[z] ← 1
for 在 T 中的每个参数 V do
    build_grad(V, G, G', grad_table)
end for
```

可以看出，在实现循环神经网络时，一定保存每个时刻的中间状态来进行参数的梯度计算，当得到所有损失关于中间状态的导数时，就可以得到不同时刻参数关于中间状态的梯度并通过连锁规则和所有的时刻梯度相加，得到本次反向传播所需的参数梯度。事实在实现中，通常会对参数的梯度的累加深度进行截断 (truncated)。上面计算梯度的操作都可以比较简单的完成，其中对角矩阵与向量相乘，在代码中也应该可以直接体现为两个矩阵按位置相乘得到 (在 NiuTensor 里应该体现为张量的点乘操作 XTensor Multiply(const XTensor &a, const XTensor &b, DTYPE alpha = 0.0, int leadingDim = 0))。

本来本次大作业准备完成简单的单层 RNN 的实现，但是由于考试原因无法完成，只能给出一个学习 RNN 反向传播的过程。还请老师谅解，如果老师觉得工作量不够，不知能否等考试结束后再给时间完成实现。

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

算法 2 build_grad

输入: 需要将自身梯度加入梯度记录表 grad_table 和运算图 G 的参数 V
需要进行修改的运算图 G
只与 V 的梯度计算有关的运算子图 G'
梯度记录表 grad_table
输出: 参数 V 的梯度

if V 已经在梯度记录表 grad_table 中 **then**
  return grad_table[V]
end if
$i \leftarrow 1$
for 每个在计算子图 G' 中使用了 V 作为输入的节点 C **do**
  $op \leftarrow$ 节点 C 执行的运算操作
  $D \leftarrow \text{build\_grad}(C, G, G', \text{grad\_table})$
  $g^{(i)} \leftarrow$ 根据节点 C 的 op、梯度和输入计算其关于 V 的梯度
  $i \leftarrow i + 1$
end for
$g \leftarrow \sum_i g^{(i)}$
grad_table[V] $\leftarrow g$
将 g 和相应操作插入运算图 G
return g