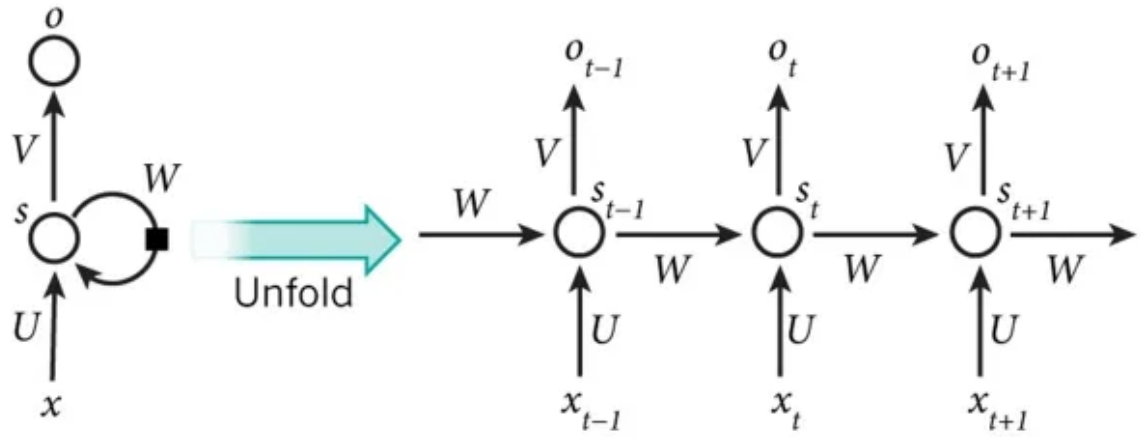
【超详细】一文带你了解RNN家族知识点

Python遇见机器学习 2020-09-04

公众号关注“Python遇见机器学习”
设为“星标”，第一时间知晓最新干货~



来源

NewBeeNLP 作者 余文毅

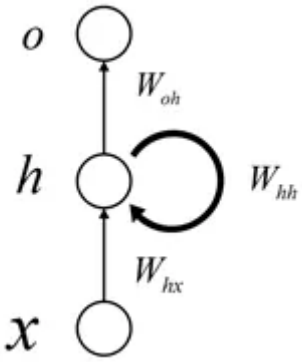
BERT虽好，不要忘记老朋友RNN呀🍻

写在前面

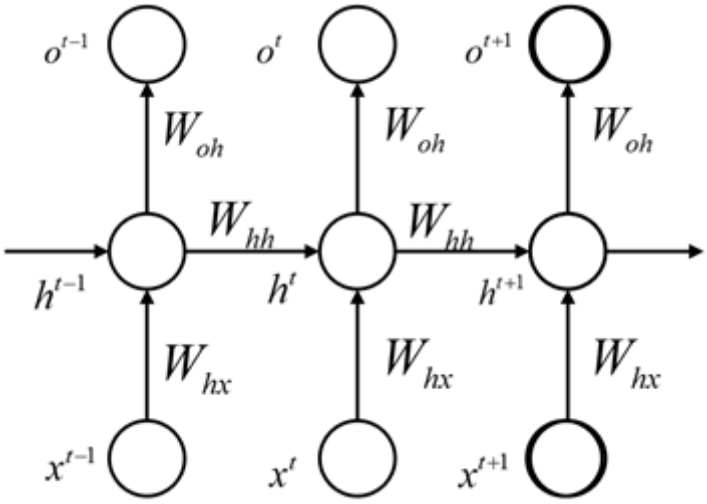
CNN（Convolution Neural Network） 和 RNN（Recurrent Neural Network）是当下 Deep Learning 应用领域中主流的两大结构。前篇文章中我们介绍了 CNN，本篇开始我们聊聊 RNN。RNN 跟 CNN 历史相似之处在于，都是上个世纪提出来的概念。但是由于当时计算量和数据量都比较匮乏，它们都被尘封，直到近几年开始大放异彩，可以说是超越时代的产物。区别在于，CNN 在2012年就开始大行其道，而 RNN 的流行却要到2015年以后了。本文会介绍 RNN 的相关概念，并具体介绍较常见的 RNN 架构。

原始 RNN

CNN 这种网络架构的特点之一就是网络的状态仅依赖于输入，而 RNN 的状态不仅依赖于输入，且与网络上一时刻的状态有关。因此，经常用于处理序列相关的问题。RNN 的基础结构如下



可以看出，它跟 CNN、DNN 这种 Feedforward Neural Network 结构上的区别就在于：Feedforward NN 的结构是 DAG（有向无环图），而 Recurrent NN 的结构中至少有一个环。我们假设h的状态转移发生在时间维度上，则上图可以展开成以下形式：



于是我们可以写出其具体表达式：

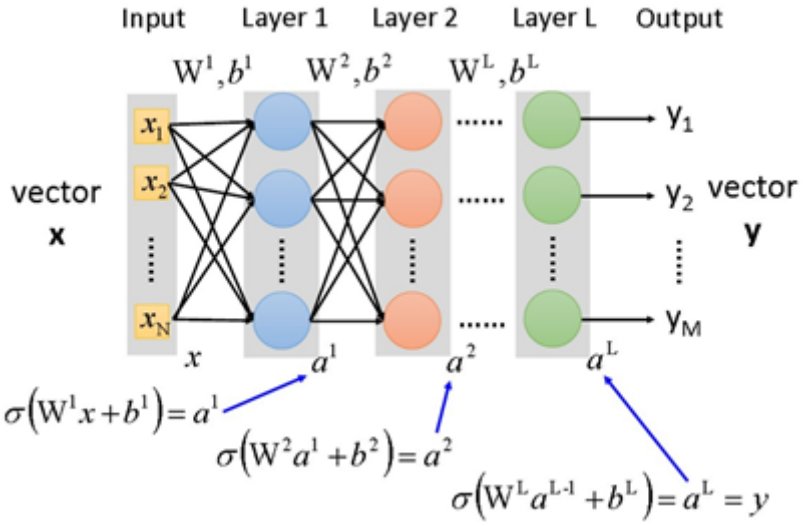
$$
\begin{aligned}
h_*^t &= W_{hx}x^t + W_{hh}h^{t-1} + b_h \\
h^t &= \sigma\left(h_*^t\right) \\
o_*^t &= W_{oh}h^t + b_o \\
o^t &= \theta\left(o_*^t\right)
\end{aligned}
$$

其中，x^t表示t时刻的输入，o^t表示t时刻的输出，h^t表示t时刻 Hidden Layer 的状态。

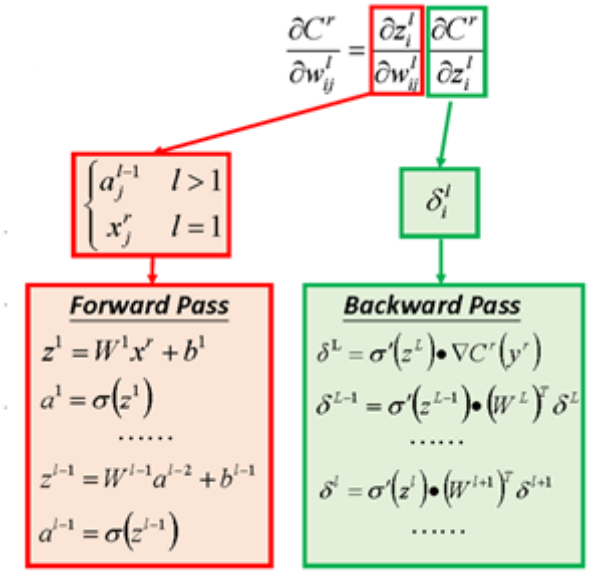
RNN与BPTT

RNN 的训练跟 CNN、DNN 本质一样，依然是 BP。但它的 BP 方法名字比较高级，叫做 BPTT（Back Propagation Through Time）。

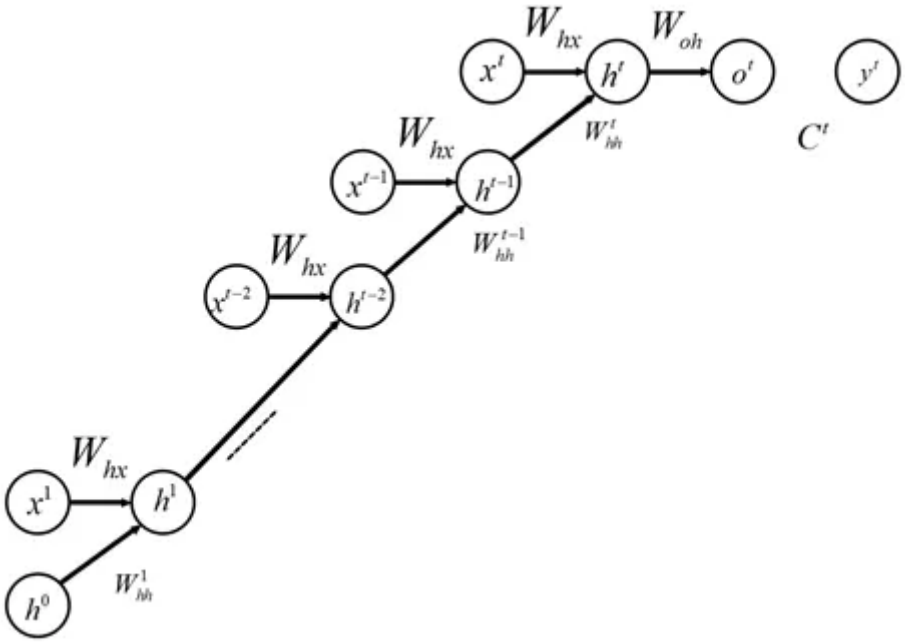
我们首先回顾一下 DNN 相关概念。DNN 的结构如下图



而 DNN 的 BP 中最重要的公式如下（不再展开讲）



有了以上 DNN 的结论，接下来我们将 RNN 沿着时间展开（UNFOLD），如下图



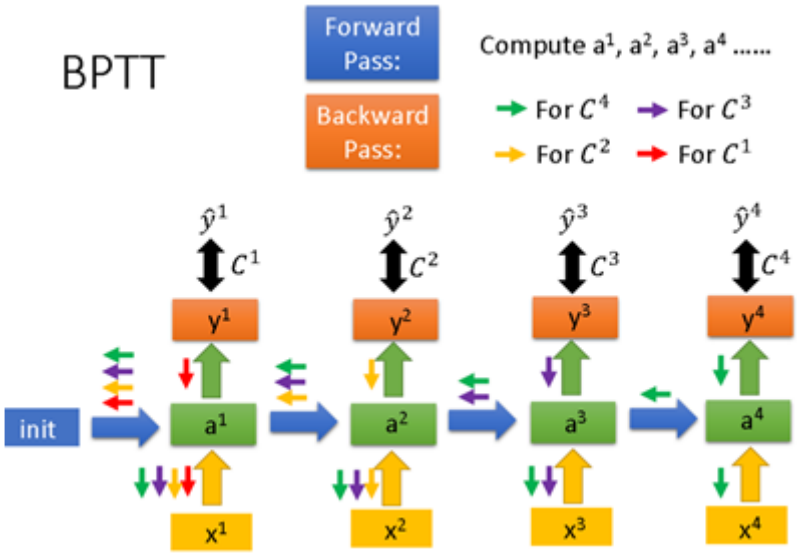
可以发现，UNFOLD 后的 RNN 其实跟 DNN 逻辑上是一样的，只是 DNN 中相邻层的连接在 RNN 中成了相邻时间的连接，所以公式也差不多。但有一点需要注意，对于某参数β，如果仅考虑第t时刻的 Loss C^t，其梯度是 UNFOLD 后整个 BP 过程中的累加梯度，如以下公式。其中β^t表示β在时刻t的状态。

$$\frac{\partial C^t}{\partial \beta} = \sum_k \frac{\partial C^t}{\partial \beta^k}$$

同理可得，如果考虑总误差$C = f(C^1, \dots C^{t-1}, C^t)$，则有，

$$\begin{aligned}\frac{\partial C}{\partial \beta} &= \sum_{j=1}^{t} \frac{\partial C^j}{\partial \beta} \frac{\partial C}{\partial C^j} \\ &= \sum_{j=1}^{t} \sum_{k=1}^{j} \frac{\partial C^j}{\partial \beta^k} f'\left(C^j\right)\end{aligned}$$

上述公式如果用更形象的方式来描述，可以参考下面这张李宏毅老师的 PPT



现在我们只考虑C^t，则观察 UNFOLD 后的 RNN 网络结构，参考着 DNN 的 BP 公式，可以直接写出原始 RNN 的 BPTT 公式如下：

$$\delta^{k-1} = \frac{\partial C^t}{\partial h_*^{k-1}} = \left(W_{hh}^T \delta^k\right) \odot \sigma'\left(h_*^{k-1}\right)$$

$$\frac{\partial C^t}{\partial w_{ij}} = \sum_{k=1}^{t} \frac{\partial h_{*i}^k}{\partial w_{ij}^k} \frac{\partial C^t}{\partial h_{*i}^k} = \sum_{k=1}^{t} h_j^{k-1} \delta_i^k$$

RNN 与 Gradient Vanish / Gradient Explode

上面 RNN 的 BPTT 公式跟 DNN 的 BP 非常相似，所以毫无疑问同样会面临 Gradient Vanish 和 Gradient Explode 的问题。这里主要有两点原因：

「1.激活函数」

对于上述公式中$h^t = \sigma(h_*^t)$，如果σ为 sigmoid 函数或者 tanh 函数，根据 δ 的递推式，当时间跨度较大时（对应于 DNN 中层数很深），δ 就会很小，从而使 BP 的梯度很小，产生 Gradient Vanish。解决方法也差不多，换一种 Activation Function，如 Relu 等

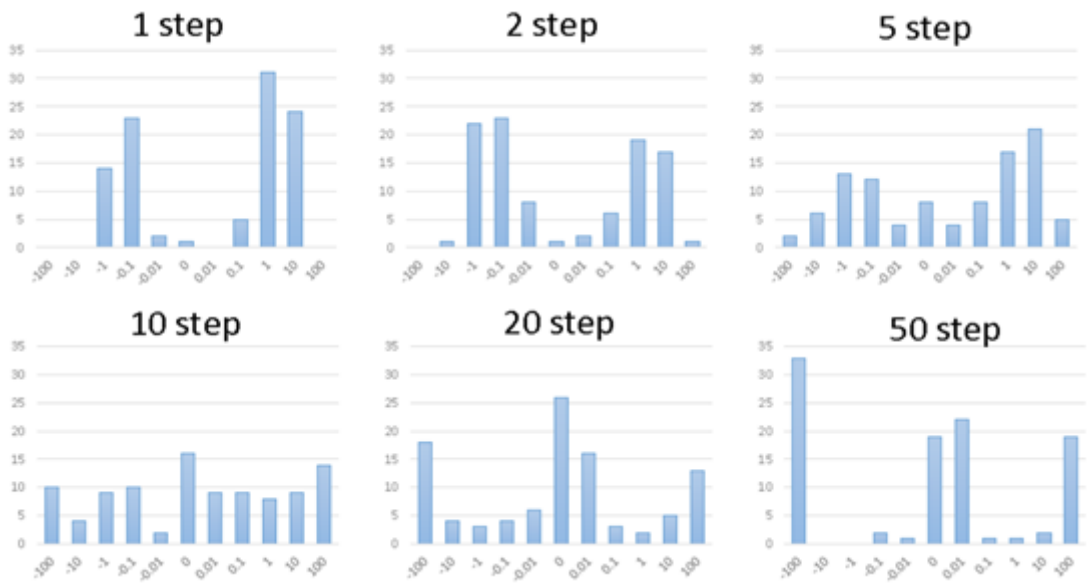
「2.参数W_{hh}」

跟 DNN 中每层的W^l，相对独立不同，RNN 中的W_{hh}在每个时刻其实指的是同一个参数，所以 δ 中会出现W_{hh} 的累乘。

当W_{hh}为对角阵时，我们就有两点结论：

- 若某对角线元素小于1，则其幂次会趋近于0，进而导致 Gradient Vanish
- 若某对角线元素大于1，则其幂次会趋近于无穷大，进而导致 Gradient Explode

当然，W_{hh}不一定是对角阵，如果其为非对角阵，我们就用实验的方式说明。首先我们对W_{hh}的数值进行随机初始化。随后观察其累乘后数值分布随着幂次的变化趋势如下图。可以看出在多次累乘后，数值的分布有明显的趋势：要么趋近于0，要么趋近于绝对值很大的值。而这两种情况，就很可能会分别造成 Gradient Vanish 和 Gradient Explode



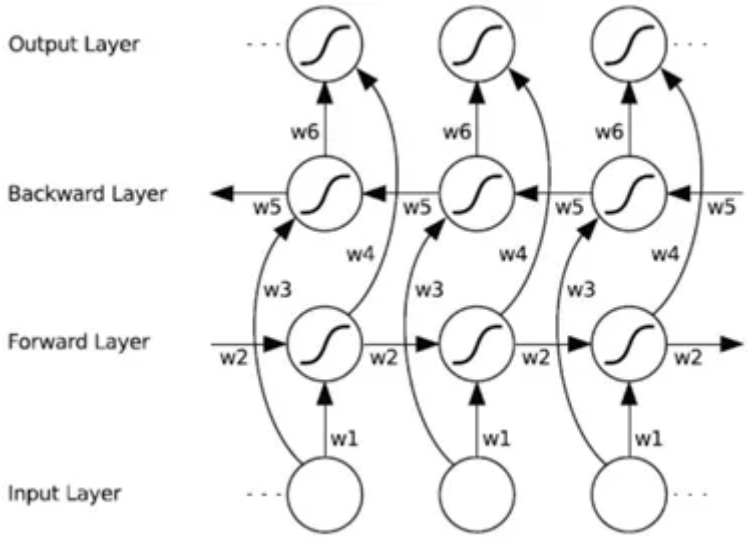
而解决 Gradient Vanish 和 Gradient Explode 的方法则有：

- 对于 Gradient Vanish，传统的方法也有效，比如换 Activation Function 等；不过一个更好的架构能更显著的缓解这个问题，比如下面会介绍的 LSTM、GRU
- 对于 Gradient Explode，一般处理方法就是将梯度限制在一定范围内，即 Gradient Clipping。可以是通过阈值，也可以做动态的放缩

BRNN

BRNN（Bi-directional RNN）由 Schuster 在"Bidirectional recurrent neural networks, 1997"中提出，是单向 RNN 的一种扩展形式。普通 RNN 只关注上文，而 BRNN 则同时关注上下文，能够利用更多的信息进行预测。

结构上， BRNN 由两个方向相反的 RNN 构成，这两个 RNN 连接着同一个输出层。这就达到了上述的同时关注上下文的目的。其具体结构图如下



BRNN 与普通 RNN 本质一样，仅在训练的步骤等细节上略有差别，这里不再详解描述。有兴趣的同学可以参考原文。

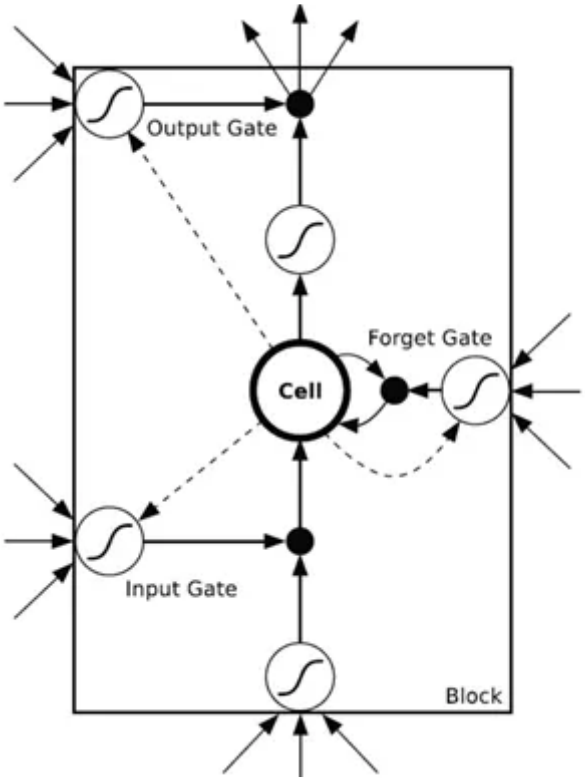
LSTM

为了解决 Gradient Vanish 的问题，Hochreiter&Schmidhuber 在论文“Long short-term memory, 1997”中提出了 LSTM（Long Short-Term Memory）。原始的 LSTM 只有 Input Gate、Output Gate。而咱们现在常说的 LSTM 还有 Forget Gate，是由 Gers 在"Learning to Forget: Continual Prediction with LSTM, 2000"中提出的改进版本。后来，在"LSTM Recurrent Networks Learn Simple Context Free and Context Sensitive Languages, 2001"中 Gers 又加入了 Peephole Connection 的概念。同时，现在常用的深度学习框架 Tensorflow、Pytorch 等在实现 LSTM 上也有一些细微的区别。以上所说的虽然本质都是 LSTM，但结构上还是有所区别，在使用时需要注意一下。

下文介绍的 LSTM 是"Traditional LSTM with Forget Gates"版本。

Traditional LSTM with Forget Gates

LSTM 其实就是将 RNN 中 Hidden Layer 的一个神经元，用一个更加复杂的结构替换，称为 Memory Block。单个 Memory Block 的结构如下（图中的虚线为 Peephole Connection，忽略即可）



先对其中结构进行简要介绍：

- Input Gate, Output Gate, Forget Gate：这三个 Gate 本质上就是权值，形象点则类似电路中用于控制电流的开关。当值为1，表示开关闭合，流量无损耗流过；当值为0，表示开关打开，完全阻塞流量；当值介于(0,1)，则表示流量通过的程度。而这种$[0,1]$的取值，其实就是通过 Sigmoid 函数实现的
- Cell：Cell 表示当前 Memory Block 的状态，对应于原始 RNN 中的 Hidden Layer 的神经元
- Activation Function：图中多处出现了 Activation Function（小圆圈+ sigmoid 曲线的图案），对这些 Activation Function 的选择有一个通用的标准。一般，对 Input Gate, Output Gate, Forget Gate，使用的 Activation Function 是 sigmoid 函数；对于 Input 和 Cell， Activation Function 使用 tanh 函数

其具体公式如下：

$$
\begin{aligned}
i^t &= \text{sigmoid}(W_{ix}x_t + W_{ih}h_{t-1} + b_i) \\
f^t &= \text{sigmoid}(W_{fx}x_t + W_{fh}h_{t-1} + b_f) \\
g^t &= \tanh(W_{gx}x_t + W_{ch}h_{t-1} + b_g) \\
o^t &= \text{sigmoid}(W_{ox}x_t + W_{oh}h_{t-1} + b_o) \\
c^t &= f^t \odot c^{t-1} + i^t \odot g^t \\
h^t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

其中，i^t, o^t, f^t分别表示 Input Gate，Output Gate，Forget Gate；g^t表示 Input；h^t表示 Output；C^t表示 Cell 在时刻t的状态

LSTM 与 Gradient Vanish

上面说到，LSTM 是为了解决 RNN 的 Gradient Vanish 的问题所提出的。关于 RNN 为什么会出现 Gradient Vanish，上面已经介绍的比较清楚了，本质原因就是因为矩阵高次幂导致的。下面简要解释一下为什么 LSTM 能有效避免 Gradient Vanish。

对于 LSTM，有如下公式

$$c^t = f^t \odot c^{t-1} + i^t \odot g^t$$

模仿 RNN，我们对 LSTM 计算$\delta^{k-1} = \partial C^t / \partial c^{k-1}$， 有

$$\begin{aligned}\delta^{k-1} &= \frac{\partial C^t}{\partial c^{k-1}} \\ &= \frac{\partial C^t}{\partial c^k}\frac{\partial c^k}{\partial c^{k-1}} \\ &= \delta^k \frac{\partial c^k}{\partial c^{k-1}} \\ &= \delta^k \left(f^t + \dots\right)\end{aligned}$$

公式里其余的项不重要，这里就用省略号代替了。可以看出当$f^t = 1$时，就算其余项很小，梯度仍然可以很好地传导到上一个时刻，此时即使层数较深也不会发生 Gradient Vanish 的问题；当$f^t = 0$时，即上一时刻的信号不影响到当前时刻，则此项也会为0；f^t在这里控制着梯度传导到上一时刻的衰减程度，与它 Forget Gate 的功能一致。

LSTM 与 BPTT

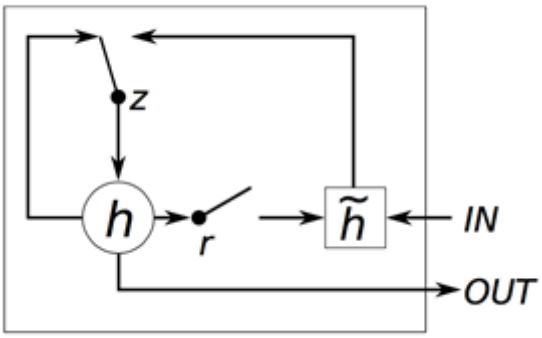
最初 LSTM 被提出时，其训练的方式为“Truncated BPTT”。大致的意思为，只有 Cell 的状态会 BP 多次，而其他部分的梯度会被截断，不 BP 到上一个时刻的 Memory Block。当然，这种方法现在也不使用了，所以仅此一提。

在"Framewise phoneme classification with bidirectional LSTM and other neural network architectures, 2005"中，作者提出了 Full Gradient BPTT 来训练 LSTM，也就是标准的 BPTT。这也是如今具有自动求导功能的开源框架们使用的方法。关于 LSTM 的 Full Gradient BPTT，我并没有推导过具体公式，有兴趣的同学可以参考 RNN 中 UNFOLD 的思想来试一试，这里也不再赘述了。

GRU

GRU（Gated Recurrent Unit）是由 K.Cho 在"Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation, 2014"中提出的。它是 LSTM 的简化版本，但在大多数任务中其表现与 LSTM 不相伯仲，因此也成为了常用的 RNN 算法之一。

GRU 的具体结构与对应的公式如下：



$$
\begin{aligned}
r^t &= \text{sigmoid}\left(W_{rx}x^t + W_{rh}h^{t-1} + b_r\right) \\
z^t &= \text{sigmoid}\left(W_{zx}x^t + W_{zh}h^{t-1} + b_z\right) \\
n^t &= \tanh\left(W_{nx}x^t + W_{nh}\left(r^t \odot h_{t-1}\right) + b_n\right) \\
h^t &= \left(1 - z^t\right) \odot n^t + z^t \odot h^{t-1}
\end{aligned}
$$

其中，r, z分别被称为 Reset Gate 和 Update Gate。可以看出，GRU 与 LSTM 有一定的相似性，而区别主要在于：

1. LSTM 有三个 Gate，而 GRU 仅两个
2. GRU 没有 LSTM 中的 Cell，而是直接计算输出
3. GRU 中的 Update Gate 类似于 LSTM 中 Input Gate 和 Forget Gate 的融合；而观察它们结构中与上一时刻相连的 Gate，就能看出 LSTM 中的 Forget Gate 其实分裂成了 GRU 中的 Update Gate 和 Reset Gate

很多实验都表明 GRU 跟 LSTM 的效果差不多，而 GRU 有更少的参数，因此相对容易训练且过拟合的问题要轻一点，在训练数据较少时可以试试。

尾巴

除了文中提到的几种架构，RNN 还有其它一些变化。但总体而言 RNN 架构的演进暂时要逊色于 CNN，暂时常用的主要是 LSTM 和 GRU。同样，也是由于 RNN 可讲的比 CNN 少些，本次就只用一篇文章来介绍 RNN，内容上进行了压缩。但是这不代表 RNN 简单，相反不论是理论还是应用上，使用 RNN 的难度都要比 CNN 大不少。