深度学习之循环神经网络（RNN）

> 循环神经网络（Recurrent Neural Network，RNN）是一类具有短期记忆能力的神经网络，适合用于处理视频、语音、文本等与时序相关的问题。在循环神经网络中，神经元不但可以接收其他神经元的信息，还可以接收自身的信息，形成具有环路的网络结构。

循环神经网络的参数学习可以通过随时间反向传播算法来学习，即按照时间的逆序把误差一步步往前传递。而当输入序列比较长时，会产生梯度爆炸或梯度消失问题，这也叫做长期依赖问题。为了解决这个问题，门控机制被引入来改进循环神经网络，也就是长短期记忆网络（LSTM）和门控循环单元（GRU）。

好了，看了上面概括性的描述，心头一定有许多疑问冒出来：

1、为什么循环神经网络拥有记忆能力呢?

2、循环神经网络的具体结构是什么样的?

3、循环神经网络怎么用随时间反向传播算法来学习?

4、循环神经网络的长期依赖问题是怎么产生的?

5、针对不同的任务，循环神经网络有哪些不同的模式?

这篇文章整理以上几个问题的答案。

一、循环神经网络的记忆能力

前馈神经网络是一个静态网络，信息的传递是单向的，网络的输出只依赖于当前的输入，不具备记忆能力。

而循环神经网络通过使用带自反馈的神经元，使得网络的输出不仅和当前的输入有关，还和上一时刻的输出相关，于是在处理任意长度的时序数据时，就具有短期记忆能力。

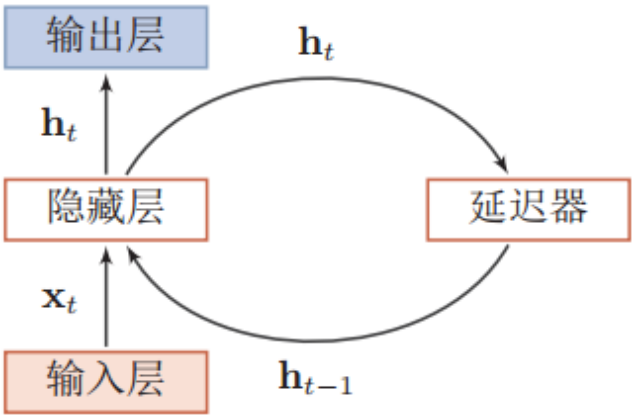
给定一个输入序列

$$\mathbf{x}_{1:T} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_t, \dots, \mathbf{x}_T)$$

，循环神经网络通过以下的公式来更新带反馈边的隐含层的活性值ht：

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

其中h0=0，f(·)是一个非线性函数，隐藏层的活性值ht又称为状态或隐状态。示例如下：

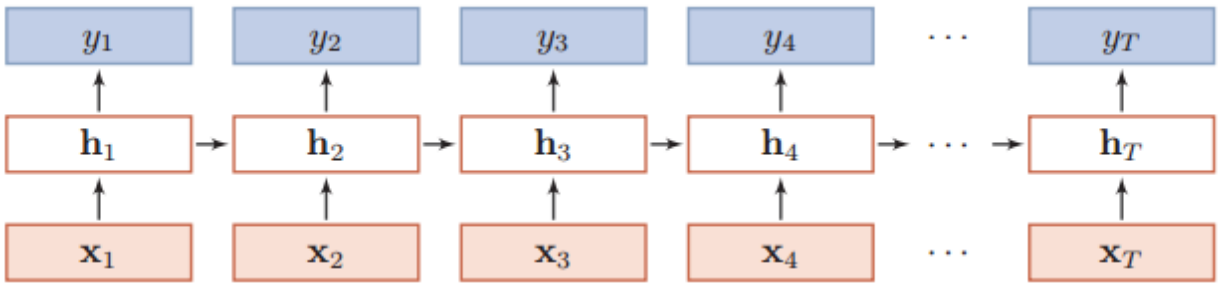


二、循环神经网络的结构

1、单向循环神经网络

先来搞清楚只有一个隐藏层的循环神经网络的结构。

直接上图！如下是一个按时间展开的循环神经网络图：



可以看到，连接不仅存在于相邻的层与层之间（比如输入层-隐藏层），还存在于时间维度上的隐藏层与隐藏层之间（反馈连接，h1到hT）。

用公式来描述隐状态的计算过程，假设在时刻t，网络的输入为xt，隐状态（即隐藏层神经元活性值）ht 不仅和当前时刻的输入xt相关，也和上一个时刻的隐状态ht-1相关，进而与全部过去的输入序列（x1, x2, ..., xt-1, xt）相关。

$$\mathbf{z}_t = U\mathbf{h}_{t-1} + W\mathbf{x}_t + \mathbf{b},$$

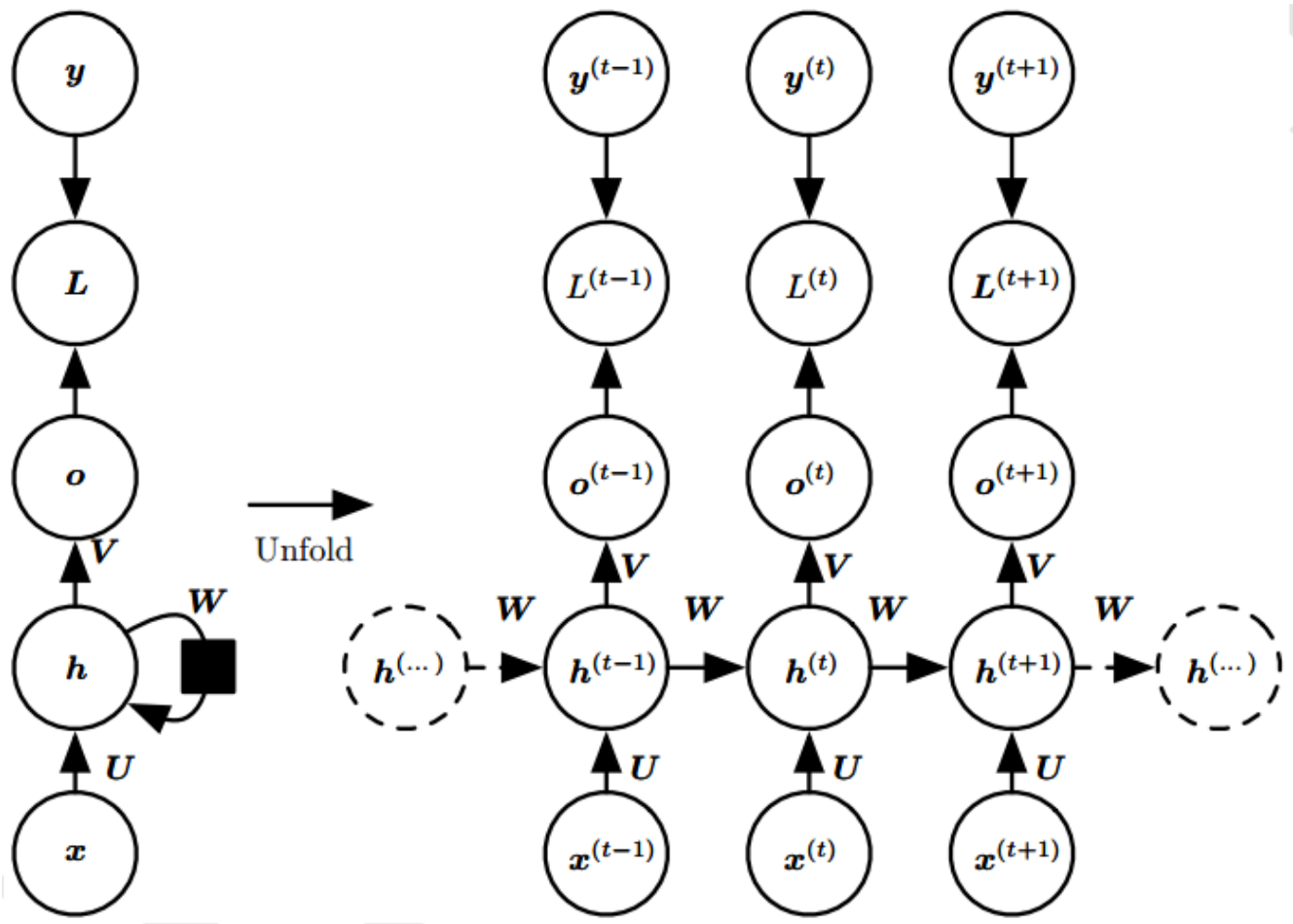
$$\mathbf{h}_t = f(\mathbf{z}_t),$$

其中zt是隐藏层的净输入；f(·)是非线性激活函数，通常为Sigmoid函数或Tanh函数；U是状态-状态权重矩阵，W是状态-输入权重矩阵，b为偏置。

这里要注意，在所有的时刻，我们使用相同参数（意思是U、W和b在每一时刻t都一样吗？）和相同的激活函数f(·)。

如果把每一时刻的状态看作是前馈神经网络的一层的话，那么循环神经网络可以看做是时间维度上**权值共享**的前馈神经网络。

有多个隐藏层的循环神经网络图如下：



2、双向循环神经网络

在某些任务中，当前时刻的输出不仅和过去的信息有关，还和后续时刻的信息有关。比如给定一个句子，即单词序列，每个单词的词性和上下文有关，因此可以增加一个按照时间的逆序来传递信息的网络层，增强网络的能力。

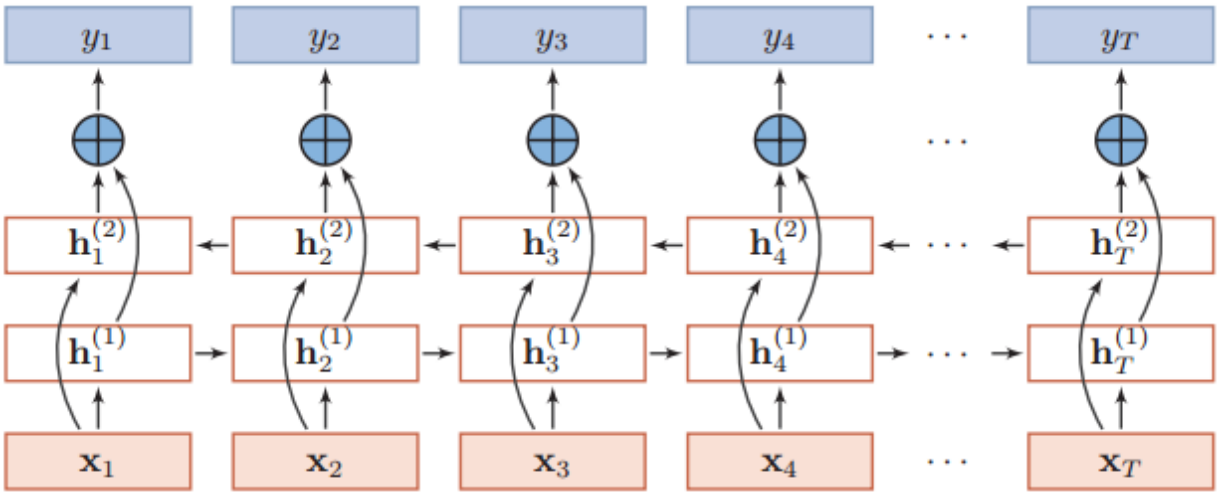
于是就有了双向循环神经网络（Bidirectional Recurrent Neural Network，Bi-RNN），它由两层循环神经网络组成，这两层网络都输入序列x，但是信息传递方向相反。

用公式来表示就是，假设第1层按时间顺序传递信息，第2层按时间逆序传递信息，这两层在时刻t的隐状态分别为ht(1)和ht(2)：

$$\mathbf{h}_t^{(1)} = f(U^{(1)}\mathbf{h}_{t-1}^{(1)} + W^{(1)}\mathbf{x}_t + \mathbf{b}^{(1)})$$
$$\mathbf{h}_t^{(2)} = f(U^{(2)}\mathbf{h}_{t+1}^{(2)} + W^{(2)}\mathbf{x}_t + \mathbf{b}^{(2)})$$
$$\mathbf{h}_t = \mathbf{h}_t^{(1)} \oplus \mathbf{h}_t^{(2)},$$

第三个式子表示把两个隐状态向量拼接起来。

双向循环神经网络按时间展开的图示如下:



三、循环神经网络的参数学习

循环神经网络的参数是状态-状态权重矩阵U，状态-输入权重矩阵W，和偏置b，可通过梯度下降法来进行学习。

以状态-状态权重矩阵U为例，推导梯度下降法进行参数学习的过程。

1、损失函数和梯度

以同步的序列到序列模式（每一时刻都有输入和输出，输入序列和输出序列长度相同）为例，运用随机梯度下降法，来计算整个序列上的损失函数。给定一个训练样本（x, y），其中x1:T = (x1, x2 ,..., xT)是长度为T的输入序列，y1:T = (y1, y2 ,..., yT)是长度为T的标签序列。于是时刻t的损失函数为：

$$\mathcal{L}_t = \mathcal{L}(y_t, g(\mathbf{h}_t))$$

其中g(ht)是第t时刻的输出，L是可微分的损失函数。那么整个序列上的损失函数为：

$$\mathcal{L} = \sum_{t=1}^{T} \mathcal{L}_t$$

于是计算整个序列上的损失函数对参数U的梯度为：

$$\frac{\partial \mathcal{L}}{\partial U} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}_t}{\partial U}$$

也就是每个时刻的损失对参数U的偏导数之和。

好，得到了损失函数和梯度的公式，那怎么计算梯度呢？

循环神经网络在时间维度上存在一个递归调用的非线性激活函数f(·)，因此计算参数梯度的方式和前馈神经网络不太相同。在循环神经网络中主要有两种计算梯度的方式：随时间反向传播（BPTT）算法和实时循环学习（RTRL）算法。这里整理一下随时间反向传播算法。

2、随时间反向传播算法

随时间反向传播算法和前馈神经网络中的误差反向传播算法比较类似，只不过是把循环神经网络看做是一个展开的多层前馈网络，每一层对应于循环神经网络中的每个时刻。

在展开的多层前馈网络中，所有层的参数是共享的，因此参数的真实梯度是所有前馈网络层的参数梯度之和。

接下来，首先计算第t时刻损失对参数U的偏导数，再计算整个序列的损失函数对参数U的梯度。

（1）计算第t时刻损失对参数U的偏导数

$$\frac{\partial \mathcal{L}_t}{\partial U}$$

是第t时刻损失对参数U的偏导数，也就是真实的参数梯度。由于第t时刻前已经有多层展开的前馈网络，而且各层的参数是共享的，所以需要把第t层和之前所有层的参数梯度都求出来，然后求和得到真实的参数梯度。

首先第t时刻的损失函数，是从净输入zt按照以下的公式一步步计算出来的：

$$z_t = Uh_{t-1} + Wx_t + b \quad \text{---净输入的计算公式}$$
$$h_t = f(z_t) \quad \text{--- } f(\bullet)\text{为激活函数}$$
$$\hat{y}_t = g(h_t) \quad \text{--- } g(\bullet)\text{为分类函数}$$
$$L_t = L(y_t, g(h_t)) \quad \text{--- } L(\bullet)\text{为损失函数}$$

于是，由隐藏层第k个时刻（1 ⩽ k ⩽ t，表示第t时刻所经过的所有时刻）的净输入zk=Uhk-1+Wxk+b可以得到，第t时刻的损失函数关于参数Uij的梯度为：

$$\begin{aligned}\frac{\partial \mathcal{L}_t}{\partial U_{ij}} &= \sum_{k=1}^{t} tr\left((\frac{\partial \mathcal{L}_t}{\partial \mathbf{z}_k})^{\mathrm{T}} \frac{\partial^{+} \mathbf{z}_k}{\partial U_{ij}} \right) \\ &= \sum_{k=1}^{t} \left(\frac{\partial^{+} \mathbf{z}_k}{\partial U_{ij}} \right)^{\mathrm{T}} \frac{\partial \mathcal{L}_t}{\partial \mathbf{z}_k},\end{aligned}$$

注意zk=Uhk-1+Wxk+b来计算zk对Uij的偏导数时，要保持hk-1不变（不然就要用链式法则，因为hk-1也是Uij的函数），这是和实时循环学习算法（RTRL）不同的地方，虽然我真没搞懂这是怎么做到

的。

于是分两步来求第t时刻的损失函数关于参数Uij的梯度：先计算每一层净输入值对参数的梯度，再计算损失函数对于每一层净输入值的梯度。

- **计算第k层的净输入值zk对参数Uij的梯度**。

$$\frac{\partial^{+}\mathbf{z}_k}{\partial U_{ij}} = \begin{bmatrix} 0 \\ \vdots \\ [\mathbf{h}_{k-1}]_j \\ \vdots \\ 0 \end{bmatrix} \triangleq \mathbb{I}_i([\mathbf{h}_{k-1}]_j)$$

- 计算第t时刻的损失对于第k时刻隐藏层的净输入zk的导数，也就是误差δt,k ，**就可以得到与前馈神经网络类似的误差反向传播公式**：

$$\begin{aligned}\delta_{t,k} &= \frac{\partial \mathcal{L}_t}{\partial \mathbf{z}_k} \\ &= \frac{\partial \mathbf{h}_k}{\partial \mathbf{z}_k}\frac{\partial \mathbf{z}_{k+1}}{\partial \mathbf{h}_k}\frac{\partial \mathcal{L}_t}{\partial \mathbf{z}_{k+1}} \\ &= \mathrm{diag}(f'(\mathbf{z}_k))U^{\mathrm{T}}\delta_{t,k+1}\end{aligned}$$

- **得到第t时刻的损失函数关于参数Uij的梯度：**

$$\frac{\partial \mathcal{L}_t}{}$$

$$\frac{\partial \mathcal{L}_t}{\partial U_{ij}} = \sum_{k=1} [\delta_{t,k}]_i [\mathbf{h}_{k-1}]_j$$

- **合并为矩阵形式：**

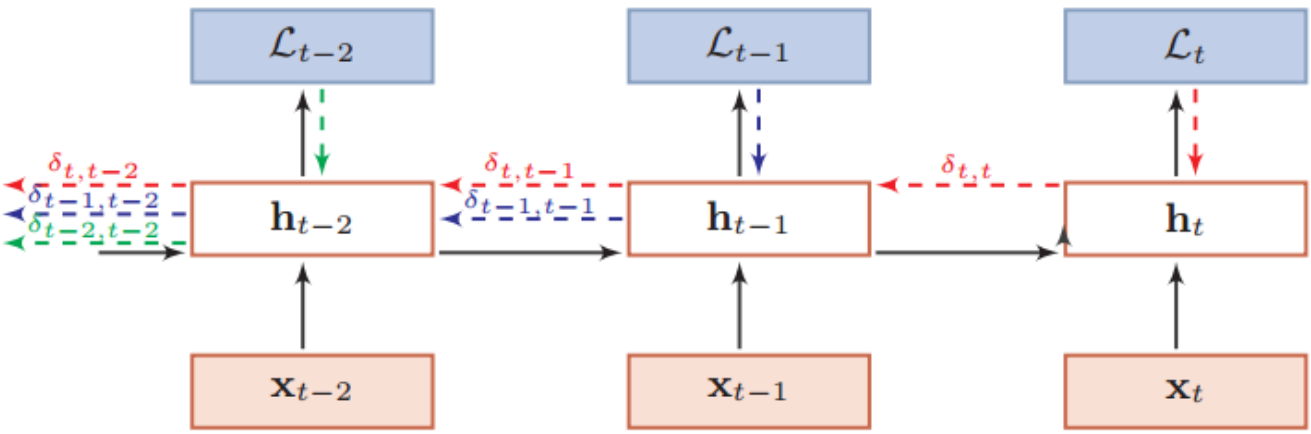
$$\frac{\partial \mathcal{L}_t}{\partial U} = \sum_{k=1}^{t} \delta_{t,k} \mathbf{h}_{k-1}^{\mathrm{T}}$$

经过以上三步，就得到了第t时刻的损失对参数U的梯度：

$\frac{\partial \mathcal{L}_t}{\partial U}$

。

上面公式太多，看完了模模糊糊明白是什么意思，结合图来理解更好。随时间的反向传播是关键，图示如下：



按照我们上面计算误差δt,k的公式：

$$\begin{aligned}\delta_{t,k} &= \frac{\partial \mathcal{L}_t}{\partial \mathbf{z}_k} \\ &= \frac{\partial \mathbf{h}_k}{\partial \mathbf{z}_k}\frac{\partial \mathbf{z}_{k+1}}{\partial \mathbf{h}_k}\frac{\partial \mathcal{L}_t}{\partial \mathbf{z}_{k+1}}\end{aligned}$$

我们来算一下δt,t-2，感受什么叫递归调用：

$$\begin{aligned}\delta_{t,t-2} &= \frac{\partial L_t}{\partial z_{t-2}} = (\frac{\partial h_{t-2}}{\partial z_{t-2}} \cdot \frac{\partial z_{t-1}}{\partial h_{t-2}}) \cdot \frac{\partial L_t}{\partial z_{t-1}} \\ &= (\frac{\partial h_{t-2}}{\partial z_{t-2}} \cdot \frac{\partial z_{t-1}}{\partial h_{t-2}}) \cdot (\frac{\partial h_{t-1}}{\partial z_{t-1}} \cdot \frac{\partial z_t}{\partial h_{t-1}}) \cdot \frac{\partial L_t}{\partial z_t} \\ &= \left[f'(z_{t-2}) \cdot U^T\right] \cdot \left[f'(z_{t-1}) \cdot U^T\right] \cdot \delta_{t,t}\end{aligned}$$

（2）计算整个序列的损失函数对参数U的梯度

得到第t时刻的损失函数对参数U的梯度之后，把所有的时刻T的梯度加起来，就得到了整个序列的损失函数对参数U的梯度。

$$\frac{\partial \mathcal{L}}{\partial U} = \sum_{t=1}^{T}\sum_{k=1}^{t} \delta_{t,k}\mathbf{h}_{k-1}^{\mathrm{T}}$$

同样，按照上面的算法，可以得到整个序列的损失函数对于参数W和偏置b的梯度：

$$\frac{\partial \mathcal{L}}{\partial W} = \sum_{t=1}^{T}\sum_{k=1}^{t}\delta_{t,k}\mathbf{x}_k^{\mathrm{T}},$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}} = \sum_{t=1}^{T}\sum_{k=1}^{t}\delta_{t,k}.$$

（3）随时间反向传播算法的计算复杂度

在随时间反向传播算法（BPTT）中，参数的梯度需要在一个完整的“前向”计算和“反向”计算后才能得到，并进行参数更新，因此需要保存所有时刻的中间梯度，空间复杂度较高。而实时循环学习（RTRL）算法在第t时刻，可以实时计算损失关于参数的梯度，不需要梯度回传，空间复杂度低。

四、循环神经网络的长期依赖问题

1、长期依赖问题

我们可以用随时间反向传播算法中的误差δt,k的公式来理解长期依赖问题，先看循环神经网络中的梯度消失和梯度爆炸问题是如何产生的。将误差δt,k的公式展开为：

$$\delta_{t,k} = \prod_{i=k}^{t-1}\Big(\mathrm{diag}(f'(\mathbf{z}_i))U^{\mathrm{T}}\Big)\delta_{t,t}$$

$$\frac{\partial \mathcal{L}_t}{\partial U} = \sum_{k=1}^{t}\delta_{t,k}\mathbf{h}_{k-1}^{\mathrm{T}}$$

如果定义

$$\gamma \cong \| \operatorname{diag}(f'(\mathbf{z}_i))U^{\mathrm{T}} \|$$

则有

$$\delta_{t,k} = \gamma^{t-k}\delta_{t,t}$$

可以看到，当γ>1，t-k→∞时，γt-k→∞，造成梯度爆炸问题；相反，γ<1，t-k→∞时，γt-k→0，会出现梯度消失问题。

而循环神经网络中经常使用的激活函数为Sigmoid函数和Tanh函数，其导数值都小于1，再加上权重矩阵U的值也不会太大，因此如果时间间隔t-k过大，就会导致误差δt,k趋于0，出现梯度消失问题。

虽然循环神经网络理论上可以建立长时间间隔的状态之间的依赖关系，但是由于梯度爆炸或梯度消失问题，实际上可能只能学习到短期的依赖关系。

因此长期依赖问题就是指，如果t时刻的输出yt依赖于t-k时刻的输入xt-k，当间隔k比较大时，由于梯度爆炸或梯度消失问题，循环神经网络难以建立这种长距离的依赖关系。长期依赖问题主要是由于梯度消失产生的。

2、改进方案

可以通过缓解循环神经网络的梯度爆炸和梯度消失问题来避免长期依赖问题，从下面的公式来看，尽量让γ≈1。

$$\gamma \cong \| \operatorname{diag}(f'(\mathbf{z}_i))U^{\mathrm{T}} \|$$

（1）梯度爆炸

可以通过权重衰减和梯度截断来避免梯度爆炸问题。权重衰减是通过给参数增加L1正则化和L2正则化来限制参数的取值范围，从而使得γ≤1。而梯度截断则是当梯度的模大于一定阈值时，就将它截断为一个比较小的数。

（2）梯度消失

可以改变模型，比如让U=I，同时使得f′(zi)=1，即

$$\mathbf{h}_t = \mathbf{h}_{t-1} + g(\mathbf{x}_t; \theta)$$

其中g(·)是一个非线性函数。γ=1，这就不存在梯度消失和梯度爆炸问题了，但是这种非线性激活的方法，会降低模型的表示能力。可以改为：

$$\mathbf{h}_t = \mathbf{h}_{t-1} + g(\mathbf{x}_t, \mathbf{h}_{t-1}; \theta)$$

ht与ht-1之间既有线性关系，也有非线性关系，在一定程度上可以化解梯度消失问题。

3、记忆容量问题

在上面梯度消失的解决办法中，是通过引入了一个函数g(·)，使得ht与ht-1之间既有线性关系，也有非线性关系。这会带来记忆容量问题，也就是随着ht不断存储新的输入信息，会变得越来越大，也就是发生饱和现象。而隐状态ht可以存储的信息是有限的，随着记忆单元存储的内容越来越多，其丢失的信息也越来越多。

为了解决容量问题，可以用两种方法。一是增加一些额外的存储单元，即外部记忆单元；二是进行选择性遗忘和选择性更新，即长短期记忆网络（LSTM）中的门控机制。

五、循环神经网络的模式

循环神经网络在不同类型的机器学习任务中有不同的模式：序列到类别模式、同步的序列到序列模式、异步的序列到序列模式。

1、序列到类别的模式

序列到类别模式主要用于序列数据的分类问题：输入为序列（T个数据），输出为类别（一个数据）。典型的就是文本分类任务，输入数据为单词的序列（构成一篇文档），输出为该文本的类别。

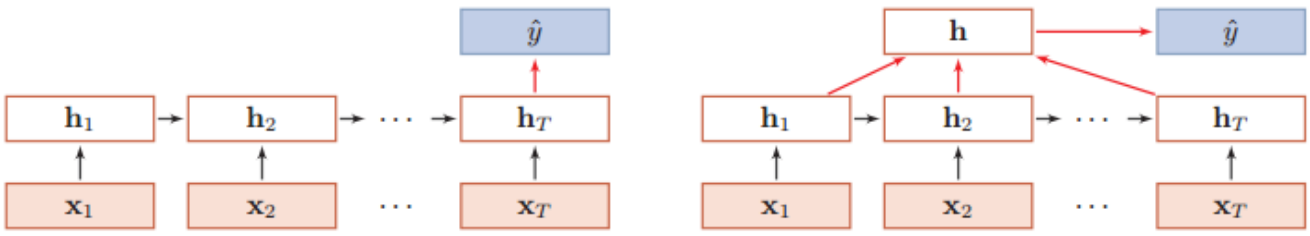
假设有一个样本x1:T = (x1, x2, ..., xT)为一个长度为T的序列，输出为一个类别y∈{1, 2, ..., C}。将样本x按不同的时刻输入到循环神经网络中去，可以得到不同时刻的隐状态h1, h2, ..., hT，然后将hT看做整个序列的最终表示，输入给分类器g(·)做分类。

$$\hat{y} = g(\mathbf{h}_T)$$

当然除了采用最后时刻的隐状态hT作为序列的表示之外，还可以对整个序列的所有状态进行平均，用平均隐状态来作为整个序列的表示。

$$\hat{y} = g(\frac{1}{T}\sum_{t=1}^{T}\mathbf{h}_t)$$

这两种序列到类别模式的图示如下：



(a) 正常模式　　(b) 按时间进行平均采样模式

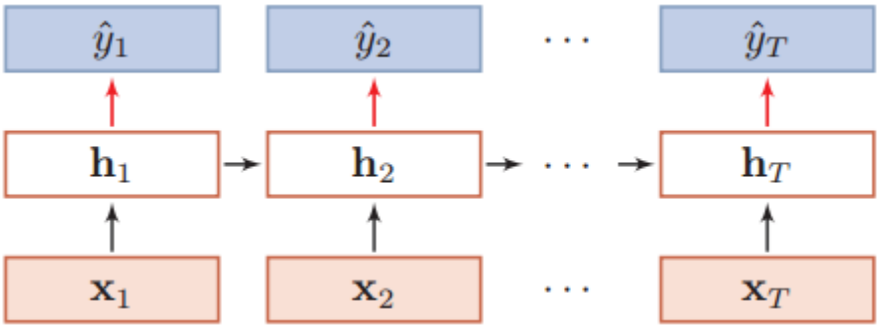
2、同步的序列到序列模式

同步的序列到序列模式主要用于序列标注任务，即每一时刻都有输入和输出，输入序列和输出序列的长度相同。比如词性标注（Pos Tagging），每个单词都需要标注它的词性。命名实体识别（Name

Entity Recognition，NER）也可以看做是序列标注问题，与词性标注的做法类似，特点在于对于命名实体，输出它的命名实体标签来代替词性。

假设有一个样本x1:T = (x1, x2, ..., xT)为一个长度为T的序列，输出序列为y1:T = (y1, y2, ..., yT)。将样本x按不同的时刻输入到循环神经网络中去，可以得到不同时刻的隐状态h1, h2, ..., hT，然后把每个时刻的隐状态输入给分类器g(·)，得到当前时刻的标签。

$$\hat{y}_t = g(\mathbf{h}_t), \qquad \forall t \in [1, T]$$



3、异步的序列到序列模式

异步的序列到序列模式也称为编码器-解码器（Encoder-Decoder）模型，即输入序列和输出序列不需要有严格的对应关系，也不用保持相同的长度。比如机器翻译中，输入为源语言的单词序列，输出为目标语言的单词序列。

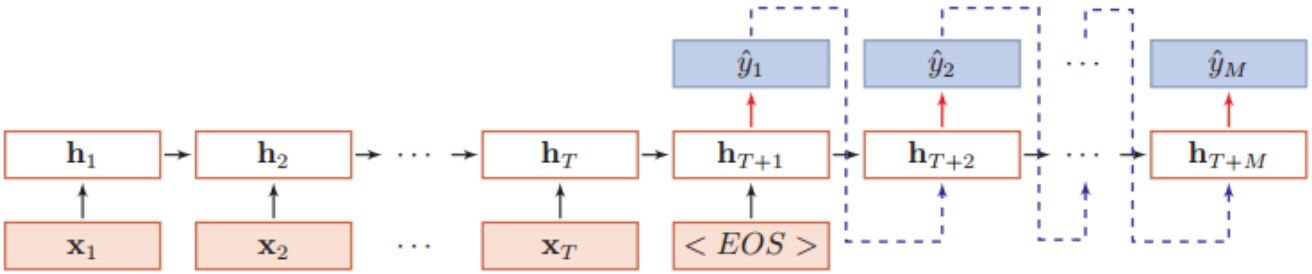
在异步的序列到序列模式中，输入为一个长度为T的序列：x1:T = (x1, x2, ..., xT)，输出一个长度为M的序列：y1:M = (y1, y2, ..., yM)，通过先编码后解码的方式实现。

先将样本x按不同时刻输入到一个循环神经网络（编码器）中，得到其编码hT，然后在另一个循环神经网络（解码器）中得到输出序列ý1:M。为了建立输出序列之间的依赖关系，在解码器中通常使用非线性的自回归模型。

$$h_t = f_1(h_{t-1}, x_t) \qquad \forall t \in [1, T]$$

$$h_{T+t} = f_2(h_{T+t-1}, \hat{y}_{t-1}) \qquad \forall t \in [1, M]$$
$$\hat{y}_t = g(h_{T+t}) \qquad \forall t \in [1, M]$$

其中f1(·)和f2(·)分别表示用作编码器和解码器的循环神经网络，g(·)为分类器。编码器和解码器的工作过程如下图所示：



参考资料：

邱锡鹏：《神经网络与深度学习》