循环神经网络（RNN）简介

图说 明明想说 3天前

循环神经网络（Recurrent Neural Network，RNN）是指一个随着时间的推移，重复发生的结构（**“循环”的含义**）。RNN网络和其他网络最大的不同就在于RNN能够实现某种**“记忆功能”**，是进行序列分析时最好的选择。如同人类能够凭借自己过往的记忆更好地认识这个世界一样，RNN也实现了类似于人脑的这一机制，对所处理过的信息留存有一定的记忆。RNN在语音识别、看图说话、机器翻译等多个领域均有非常广泛的应用。

下面从循环神经网络的记忆能力、单向循环神经网络、参数学习、长期依赖问题、工作模式及应用领域几个方面阐述RNN。

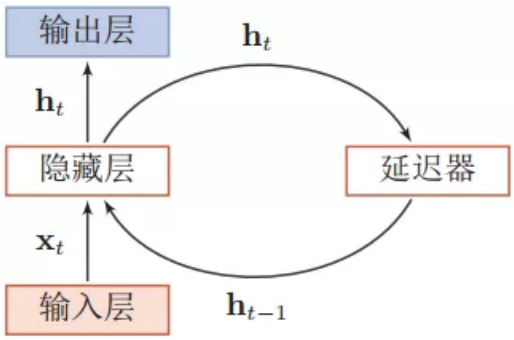
一、 循环神经网络的记忆能力

前馈神经网络（如CNN）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$\mathbf{x}_{1:T} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t, \ldots, \mathbf{x}_T)$，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：

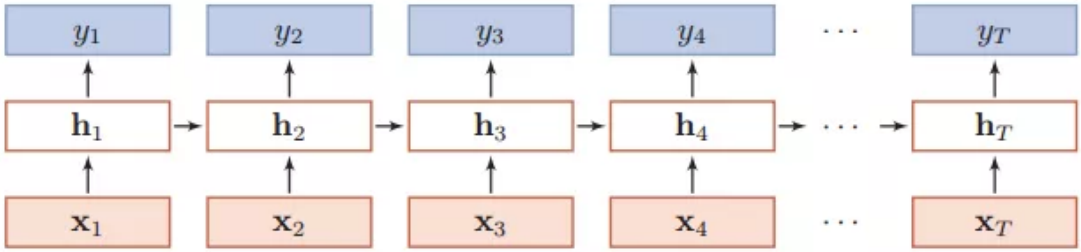
$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

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



二、（单向）循环神经网络

只有一个隐藏层的循环神经网络的结构按时间展开的循环神经网络图如下：



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，h_1到h_T）。

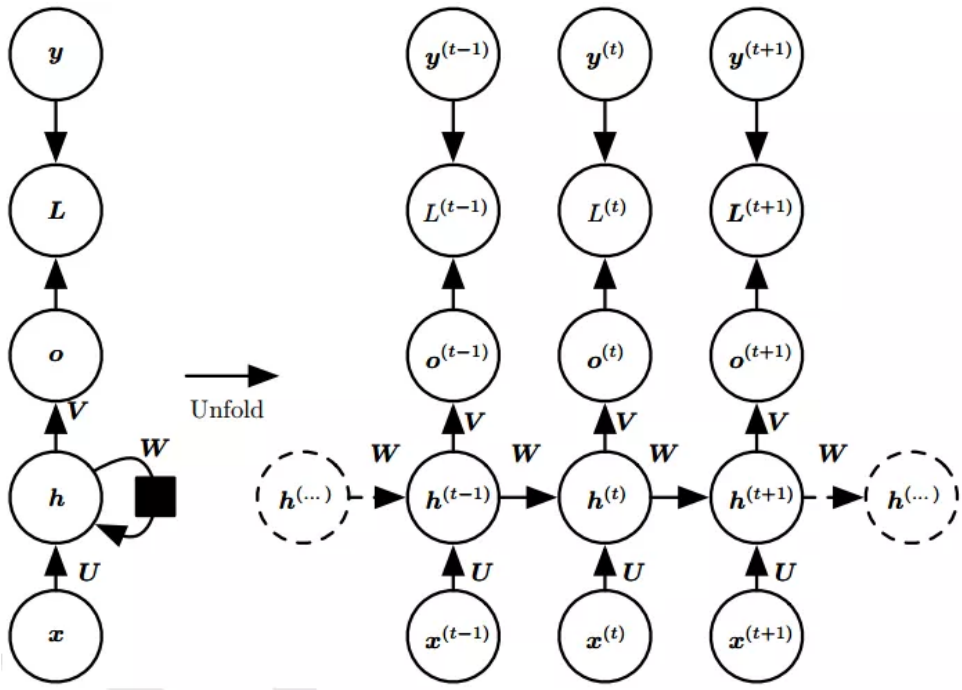
用公式来描述隐状态的计算过程，假设在时刻t，网络的输入为x_t，隐状态（即隐藏层神经元活性值）h_t 不仅和当前时刻的输入x_t相关，也和上一个时刻的隐状态h_{t-1}相关，进而与全部过去的输入序列（$x_1, x_2, ..., x_{t-1}, x_t$）相关。

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

其中z_t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，使用相同参数U，W，b和相同的激活函数f(•)。

有多个隐藏层的循环神经网络图如下：



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。下面以状态-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$x_{1:T} = (x_1, x_2, ..., x_T)$是长度为T的输入序列，$y_{1:T} = (y_1, y_2, ..., y_T)$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$g(h_t)$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。

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。这里介绍随时间反向传播算法。

2、随时间反向传播算法（BPTT）

由于RNN模型与时间序列有关，因此不能直接使用BP（back propagation）算法。针对RNN问题的特殊情况，提出了BPTT算法。BPTT的全称是“随时间反向传播算法”（back propagation through time）。这个方法的基础仍然是常规的链式求导法则。

接下来，首先计算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t时刻损失对参数U的偏导数

$\frac{\partial \mathcal{L}_t}{\partial U}$是t时刻损失对参数U的偏导数。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。

第t时刻的损失函数，是从净输入z_t按照以下的公式一步步计算出来的：

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

于是，由隐藏层第k个时刻（$1 \le k \le t$，表示第t时刻所经过的所有时刻）的净输入$z_k=Uh_{k-1}+Wx_k+b$可以得到，第t时刻的损失函数关于参数U_{ij}的梯度为：

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

于是分两步来求第t时刻的损失函数关于参数U_{ij}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z_k对参数U_{ij}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z_k的导数，也就是误差$\delta_{t,k}$，**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U_{ij}的梯度：**

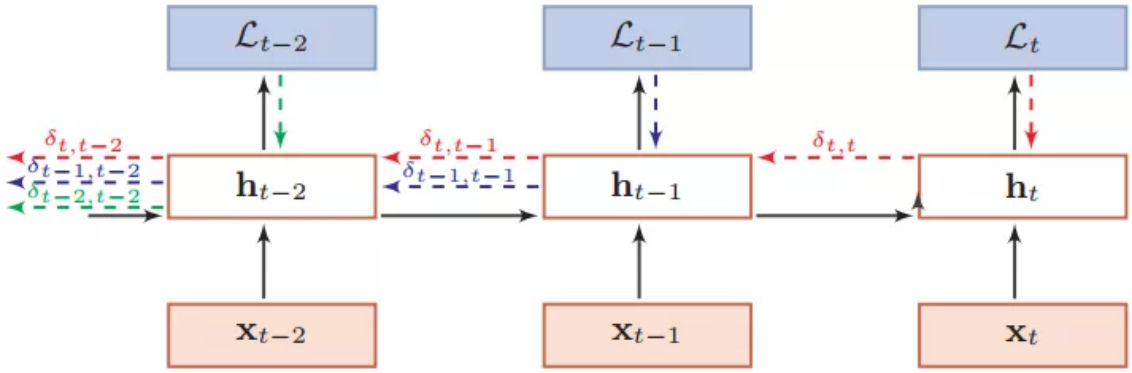
$$
\frac{\partial \mathcal{L}_t}{\partial U_{ij}} = \sum_{k=1}^{t} [\delta_{t,k}]_i [\mathbf{h}_{k-1}]_j
$$

- **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，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$\delta_{t,k}$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$\delta_{t,t-2}$，感受什么叫递归调用：

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

（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$\delta_{t,k}$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$\delta_{t,k}$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

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

则有

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

可以看到，当$\gamma>1$，$t-k\to\infty$时，$\gamma^{t-k}\to\infty$，造成梯度爆炸问题；相反，$\gamma<1$，$t-k\to\infty$时，$\gamma^{t-k}\to 0$，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$\delta_{t,k}$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y_t依赖于t-k时刻的输入x_{t-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

缓解循环神经网络的梯度爆炸和梯度消失问题可以避免长期依赖问题，从下面的公式来看，尽量让$\gamma \approx 1$。

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

（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L_1正则化和L_2正则化来限制参数的取值范围，从而使得$\gamma \leq 1$。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$f'(z_i)=1$，即

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

其中$g(\bullet)$是一个非线性函数。$\gamma=1$，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：

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

h_t与h_{t-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$g(\bullet)$，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h_t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h_t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。

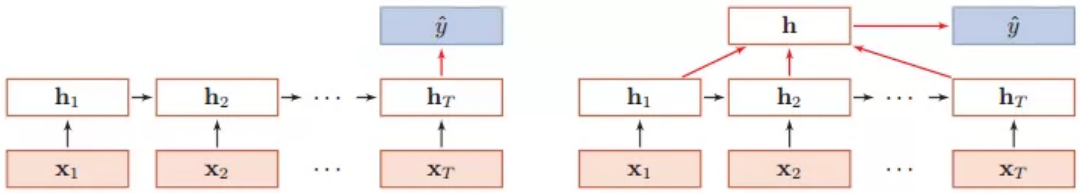
假设有一个样本$x_{1:T} = (x_1, x_2, ..., x_T)$为一个长度为T的序列，输出为一个类别$y \in \{1, 2, ..., C\}$。将样本x按不同的时刻输入到循环神经网络中去，可以得到不同时刻的隐状态$h_1, h_2, ..., h_T$，然后将h_T看做整个序列的最终表示，输入给分类器$g(\bullet)$做分类。

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

当然除了采用最后时刻的隐状态h_T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：



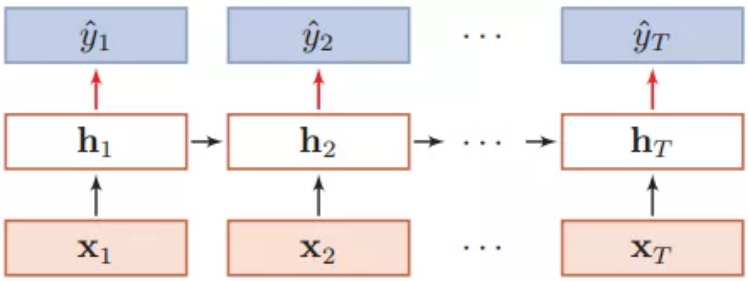
(a) 正常模式　　(b) 按时间进行平均采样模式

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$x_{1:T} = (x_1, x_2, ..., x_T)$为一个长度为T的序列，输出序列为$y_{1:T} = (y_1, y_2, ..., y_T)$。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$h_1, h_2, ..., h_T$，然后把每个时刻的隐状态输入给分类器$g(\bullet)$，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

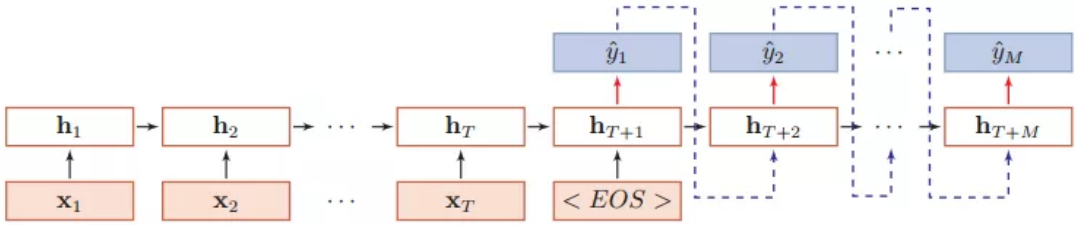
异步的序列到序列模式也称为编码器-解码器（Encoder-Decoder）模型，即输入序列和输出序列不需要有严格的对应关系，也不用保持相同的长度。比如机器翻译中，输入为源语言的单词序列，输出为目标语言的单词序列。

在异步的序列到序列模式中，输入为一个长度为T的序列：$x_{1:T} = (x_1, x_2, ..., x_T)$，输出一个长度为M的序列：$y_{1:M} = (y_1, y_2, ..., y_M)$，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h^T，然后在另一个循环神经网络（解码器）中得到输出序列$\hat{y}_{1:M}$。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。

$$
\begin{aligned}
h_t &= f_1(h_{t-1}, x_t) && \forall t \in [1, T] \\
h_{T+t} &= f_2(h_{T+t-1}, \hat{y}_{t-1}) && \forall t \in [1, M] \\
\hat{y}_t &= g(h_{T+t}) && \forall t \in [1, M]
\end{aligned}
$$

其中$f_1(\bullet)$和$f_2(\bullet)$分别表示用作编码器和解码器的循环神经网络，$g(\bullet)$为分类器。编码器和解码器的工作过程如下图所示：



六、 RNN的应用领域

RNN的应用非常多，下面是比较常见的应用领域。

1） 语言建模和文本生成: 给出一个词语序列，试着预测下一个词语的可能性。这在翻译任务中是很有用的，因为最有可能的句子将是可能性最高的单词组成的句子。

2） 机器翻译: 将文本内容从一种语言翻译成其他语言使用了一种或几种形式的 RNN。所有日常使用的实用系统都用了某种高级版本的 RNN。

3） 语音识别: 基于输入的声波预测语音片段，从而确定词语。

4） 生成图像描述: RNN 一个非常广泛的应用是理解图像中发生了什么，从而做出合理的描述。这是 CNN 和 RNN 相结合的作用。CNN 做图像分割，RNN 用分割后的数据重建描述。这种应用虽然基本，但可能性是无穷的。

5） 视频标记: 可以通过一帧一帧地标记视频进行视频搜索。

七、总结

循环神经网络最主要的特色是具有“记忆”功能，所以特别适合序列的场景。“记忆”功能的代价是处理的数据比较多，既有当前的数据，也有之前的数据，所以处理复杂度相对比较高。

参考：

MEMORY NETWORKS

End-To-End Memory Networks

Ask Me Anything: Dynamic Memory Networks for Natural Language Processing

https://baijiahao.baidu.com/s?id=1606695096009275683&wfr=spider&for=pc

https://www.cnblogs.com/Luv-GEM/p/10703906.html

喜欢此内容的人还喜欢

神经网络初试（一）

我是陈小白

一站式机器学习云研发平台

老胡的储物柜

扯扯ATT&CK和机器学习

落水轩