循环神经网络 RNN

1. 循环神经网络`recurrent neural network:RNN`：用于处理序列数据

   $\mathbf{x}_i=\{\vec{\mathbf{x}}_i^{(1)},\vec{\mathbf{x}}_i^{(2)},\cdots,\vec{\mathbf{x}}_i^{(\tau_i)}\}$ 的神经网络，其中 $\mathbf{x}_i$ 表示第 i 个样本。$\mathbf{x}_i$ 是一个序列，序列的长度可以是固定的、也可以是变化的。

   - 固定的序列长度：对每个样本 $\mathbf{x}_i$，其序列长度都是常数 $\tau_i=\tau$。
   - 可变的序列长度：样本 $\mathbf{x}_i$ 的序列长度 τ_i 可能不等于样本 $\mathbf{x}_j$ 的序列长度 $\tau_j,i\neq j$。

2. 循环神经网络是一种共享参数的网络：参数在每个时间点上共享。

   传统的前馈神经网络在每个时间点上分配一个独立的参数，因此网络需要学习每个时间点上的权重。而循环神经网络在每个时间点上共享相同的权重。

3. 循环网络中使用参数共享的前提是：相同参数可以用于不同的时间步。即：前一个时间步和后一个时间步之间的关系与时刻 t 无关。

4. 就像几乎所有函数都可以被认为是前馈神经网络，几乎任何涉及循环的函数都可以被认为是循环神经网络。

一、RNN计算图

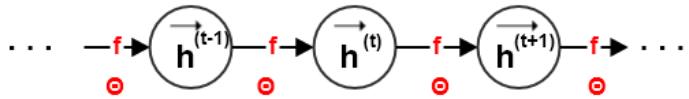
1.1 展开图

1. 考虑动态系统的经典形式：$\vec{\mathbf{h}}^{(t)}=f(\vec{\mathbf{h}}^{(t-1)};\Theta)$。其中：$\vec{\mathbf{h}}^{(t)}$ 称作系统的状态，Θ 为参数。

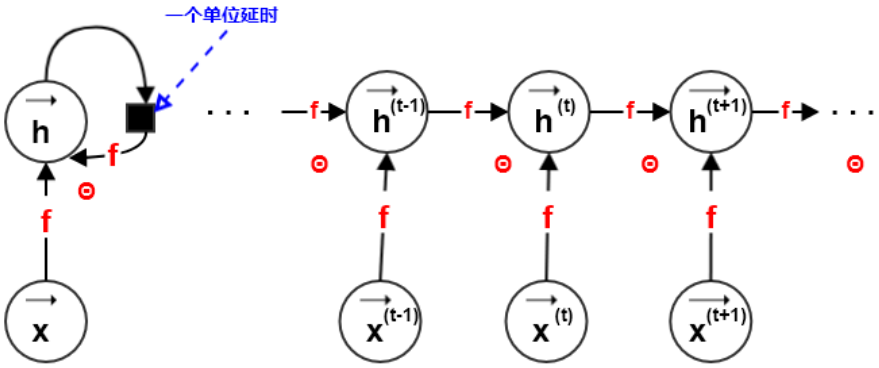
   对于有限的时间步 τ，应用 $\tau-1$ 次定义可以展开这个图：

   $$\vec{\mathbf{h}}^{(\tau)}=f(\vec{\mathbf{h}}^{(\tau-1)};\Theta)=\cdots=f(\cdots f(\vec{\mathbf{h}}^{(1)};\Theta)\cdots;\Theta)$$

   利用有向无环图来表述：



   假设 $\vec{\mathbf{x}}^{(t)}$ 为 t 时刻系统的外部驱动信号，则动态系统的状态修改为：$\vec{\mathbf{h}}^{(t)}=f(\vec{\mathbf{h}}^{(t-1)},\vec{\mathbf{x}}^{(t)};\Theta)$。



RNN：循环图　　RNN：展开图

2. 当训练`RNN`根据过去预测未来时，网络通常要将 $\vec{\mathbf{h}}^{(t)}$ 作为过去序列信息的一个有损的`representation`。

- 这个`representation`一般是有损的，因为它使用一个固定长度的向量 $\vec{\mathbf{h}}^{(t)}$ 来表达任意长的序列 $\{\vec{\mathbf{x}}^{(1)},\cdots,\vec{\mathbf{x}}^{(t-1)}\}$ 。
- 根据不同的训练准则，`representation` 可能会有选择地保留过去序列的某些部分。如`attention` 机制。

3. 网络的初始状态 $\vec{\mathbf{h}}^{(0)}$ 的设置有两种方式：

   - 固定为全零。这种方式比较简单实用。

     这种情况下，模型的反向梯度计算不需要考虑 $\vec{\mathbf{h}}^{(0)}$ ，因为 $\vec{\mathbf{h}}^{(0)}$ 全零导致对应参数的梯度贡献也为 0 。

   - 使用上一个样本的最后一个状态，即： $\vec{\mathbf{h}}_{i+1}^{(0)}=\vec{\mathbf{h}}_{i}^{(\tau_i)}$ 。

     这种场景通常是样本之间存在连续的关系（如：样本分别代表一篇小说中的每个句子），并且样本之间没有发生混洗的情况。此时，后一个样本的初始状态和前一个样本的最后状态可以认为保持连续性。

   另外注意：模型更新过程中

4. 展开图的两个主要优点：

   - 无论输入序列的长度 τ 如何，学得的模型始终具有相同的输入大小。因为模型在每个时间步上，其模型的输入 $\vec{\mathbf{x}}^{(t)}$ 都是相同大小的。
   - 每个时间步上都使用相同的转移函数 f ，因此需要学得的参数 Θ 也就在每个时间步上共享。

   这些优点直接导致了：

   - 使得学习在所有时间步、所有序列长度上操作的单个函数 f 成为可能。
   - 允许单个函数 f 泛化到没有见过的序列长度。
   - 学习模型所需的训练样本远少于非参数共享的模型（如前馈神经网络）。

1.2 网络模式

1. 基于图展开和参数共享的思想，可以设计不同模式的循环神经网络。根据输入序列的长度，`RNN` 网络模式可以划分为：输入序列长度为`0`、输入序列长度为`1` 、输入序列长度为 τ。

2. 设样本集合为 $\mathbb{X}=\{\mathbf{x}_1,\mathbf{x}_2,\cdots,\mathbf{x}_N\}$，其中每个样本为：

   - 对于输入序列长度为1的样本， $\mathbf{x}_i=\vec{\mathbf{x}}_i$ 。
   - 对于输入序列长度大于1的样本，$\mathbf{x}_i=\{\vec{\mathbf{x}}_i^{(1)},\vec{\mathbf{x}}_i^{(2)},\cdots,\vec{\mathbf{x}}_i^{(\tau_i)}\}$ ，其中 τ_i 为第 i 个样本的序列长度。

   设样本对应的真实标记集合为 $\mathbb{Y}=\{\mathbf{y}_1,\mathbf{y}_2,\cdots,\mathbf{y}_N\}$ ，其中每个样本的标记为：

   - 对于输出序列长度为1的样本， $\mathbf{y}_i=y_i$ 。对应的网络输出为：$\mathbf{o}_i=\vec{\mathbf{o}}_i$ 。
   - 对于输出序列长度大于1的样本， $\mathbf{y}_i=\{y_i^{(1)},y_i^{(2)},\cdots,y_i^{(\tau_i)}\}$，其中 τ_i 为第 i 个样本的序列长度。

     对应的网络输出为：$\mathbf{o}_i=\{\vec{\mathbf{o}}_i^{(1)},\vec{\mathbf{o}}_i^{(2)},\cdots,\vec{\mathbf{o}}_i^{(\tau_i)}\}$ 。

   设真实标记 y_i 为真实类别标签，网络的输出 $\vec{\mathbf{o}}_i$ 为预测为各类别的概率分布（经过 `softmax` 归一化的概率）。则该样本的损失函数为：

$$L_i=\sum_{t=1}^{\tau_i}L^{(t)}(y_i^{(t)},\vec{\mathbf{o}}_i^{(t)})$$

   其中 $L^{(t)}(\cdot)$ 为第 t 个时间步的损失函数。通常采用负的对数似然作为损失函数，则有：

$$L_i=-\sum_{t=1}^{\tau_i}\sum_{k=1}^{K}\mathbb{I}_{\left(k=y_i^{(t)}\right)}\log o_{i,k}^{(t)}$$

   其中 K 为类别的数量，$o_{i,k}^{(t)}$ 为 $\vec{\mathbf{o}}_i^{(t)}$ 的第 k 个分量， $\mathbb{I}(\cdot)$ 为示性函数：

$$\mathbb{I}(true)=1,\mathbb{I}(false)=0$$

如果将真实类别 $y_i^{(t)}$ 标记扩充为概率分布 $\vec{\mathbf{y}}_i^{(t)}=(0,\cdots,0,1,0,\cdots,0)$，其中真实的类别 $y_i^{(t)}$ 位置上其分量为 1，而其它位置上的分量为 0。则 $L^{(t)}(\cdot)$ 就是真实分布 $\vec{\mathbf{y}}_i^{(t)}$ 和预测分布 $\vec{\mathbf{o}}_i^{(t)}$ 的交叉熵：

$$L^{(t)}(y_i^{(t)},\vec{\mathbf{o}}_i^{(t)})=-\vec{\mathbf{y}}_i^{(t)}\cdot\log\vec{\mathbf{o}}_i^{(t)}$$

数据集的经验损失函数为：

$$\mathcal{L}=-\sum_{i=1}^{N}\sum_{t=1}^{\tau_i}\vec{\mathbf{y}}_i^{(t)}\cdot\log\vec{\mathbf{o}}_i^{(t)}$$

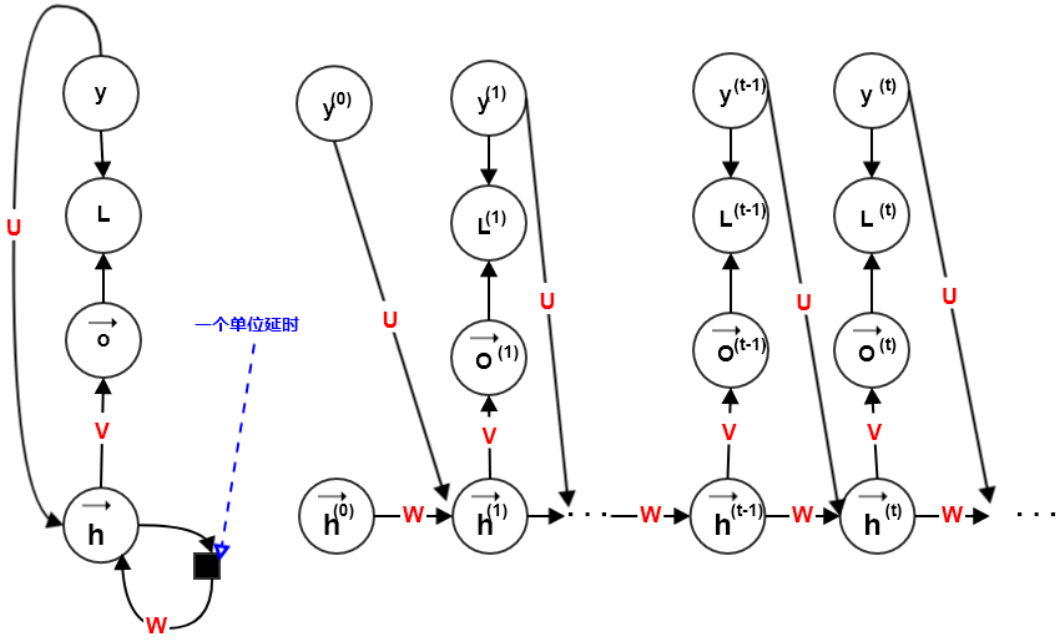
.

1.2.1 零长度输入序列

1. 输入序列长度为 0：此时网络没有外部输入，网络将当前时刻的输出作为下一个时刻的输入（需要提供一个初始的输出作为种子）。

   如文本生成算法：首先给定 $y^{(0)}$ 作为种子，然后通过 t 时刻为止的单词序列来预测 $t+1$ 时刻的单词；如果遇到某个输出为停止符，或者句子长度达到给定阈值则停止生成。

   在这个任务中，任何早期输出的单词都会对它后面的单词产生影响。



RNN：循环图　　RNN：展开图

2. 在零长度输入序列的 RNN 网络中，过去的输出序列 $\{y^{(1)}y^{(2)},\cdots,y^{(t-1)}\}$ 通过影响 $\vec{\mathbf{h}}^{(t)}$ 来影响当前的输出 $y^{(t)}$，从而解耦 $y^{(i)},i=1,2,\cdots,t-1$ 和 $y^{(t)}$。

3. 该模型的数学表示为：

$$o_k^{(t)}=p(y^{(t)}=k\mid y^{(0)},y^{(1)},\cdots,y^{(t-1)}),\quad k=1,2,\cdots,K$$

   其中 $o_k^{(t)}$ 表示模型第 t 步输出 $\vec{\mathbf{o}}_i^{(t)}$ 的第 k 个分量。

   单个样本的损失为：

$$L=-\sum_{t=1}^{\tau}\sum_{k=1}^{K}\mathbb{I}_{(k=y^{(t)})}\log o_k^{(t)}$$

   更新方程：

$$\vec{\mathbf{a}}^{(t)}=\vec{\mathbf{b}}+\mathbf{W}\vec{\mathbf{h}}^{(t-1)}+\mathbf{U}\vec{\mathbf{y}}^{(t-1)}$$
$$\vec{\mathbf{h}}^{(t)}=\tanh(\vec{\mathbf{a}}^{(t)})$$
$$\vec{\mathbf{o}}^{(t)}=\text{softmax}\left(\vec{\mathbf{c}}+\mathbf{V}\vec{\mathbf{h}}^{(t)}\right)$$

其中输出到隐状态的权重为 $\mathbf{U}$，隐状态到输出的权重为 $\mathbf{V}$ ，隐状态到隐状态的权重为 $\mathbf{W}$ ， $\vec{\mathbf{b}},\vec{\mathbf{c}}$ 为输入偏置向量和输出偏置向量。

1.2.2 单长度输入序列

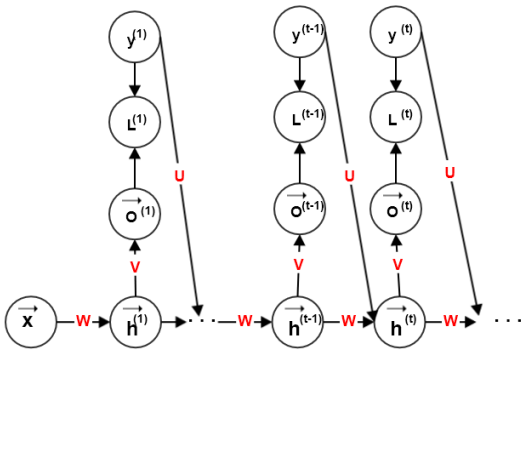
1. 输入序列长度为`1` ： 模型包含单个 $\vec{\mathbf{x}}$ 作为输入。此时有三种输入方式：输入 $\vec{\mathbf{x}}$ 作为每个时间步的输入、输入 $\vec{\mathbf{x}}$ 作为初始状态 $\vec{\mathbf{h}}^{(0)}$ 、以及这两种方式的结合。
   - 输入 $\vec{\mathbf{x}}$ 作为每个时间步的输入：
     - 模型的数学表示：$o_k^{(t)}=p(y^{(t)}=k\mid y^{(1)},\cdots,y^{(t-1)},\vec{\mathbf{x}}),\quad k=1,2,\cdots,K$
     - 单个样本的损失：$L=-\sum_{t=1}^{\tau}\sum_{k=1}^{K}\mathbb{I}_{(k=y^{(t)})}\log o_k^{(t)}$
     - 更新方程：

       $$\vec{\mathbf{a}}^{(t)}=\vec{\mathbf{b}}+\mathbf{W}\vec{\mathbf{h}}^{(t-1)}+\mathbf{U}\vec{\mathbf{y}}^{(t-1)}+\mathbf{R}\vec{\mathbf{x}}$$
       $$\vec{\mathbf{h}}^{(t)}=\tanh(\vec{\mathbf{a}}^{(t)})$$
       $$\vec{\mathbf{o}}^{(t)}=\text{softmax}\left(\vec{\mathbf{c}}+\mathbf{V}\vec{\mathbf{h}}^{(t)}\right)$$

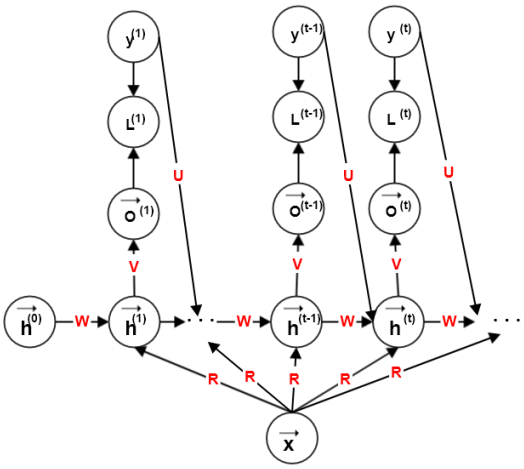
       其中输入到隐状态的权重为 $\mathbf{R}$，输出到隐状态的权重为 $\mathbf{U}$，隐状态到输出的权重为 $\mathbf{V}$ ，隐状态到隐状态的权重为 $\mathbf{W}$ ， $\vec{\mathbf{b}},\vec{\mathbf{c}}$ 为输入偏置向量和输出偏置向量。
   - 输入 $\vec{\mathbf{x}}$ 作为初始状态 $\vec{\mathbf{h}}^{(0)}$ ：
     - 模型的数学表示：$o_k^{(t)}=p(y^{(t)}=k\mid y^{(1)},\cdots,y^{(t-1)}),\quad k=1,2,\cdots,K$
     - 单个样本的损失：$L=-\sum_{t=1}^{\tau}\sum_{k=1}^{K}\mathbb{I}_{(k=y^{(t)})}\log o_k^{(t)}$
     - 更新方程：

       $$\vec{\mathbf{a}}^{(t)}=\vec{\mathbf{b}}+\mathbf{W}\vec{\mathbf{h}}^{(t-1)}+\mathbf{U}\vec{\mathbf{y}}^{(t-1)}$$
       $$\vec{\mathbf{h}}^{(t)}=\tanh(\vec{\mathbf{a}}^{(t)})$$
       $$\vec{\mathbf{o}}^{(t)}=\text{softmax}\left(\vec{\mathbf{c}}+\mathbf{V}\vec{\mathbf{h}}^{(t)}\right)$$

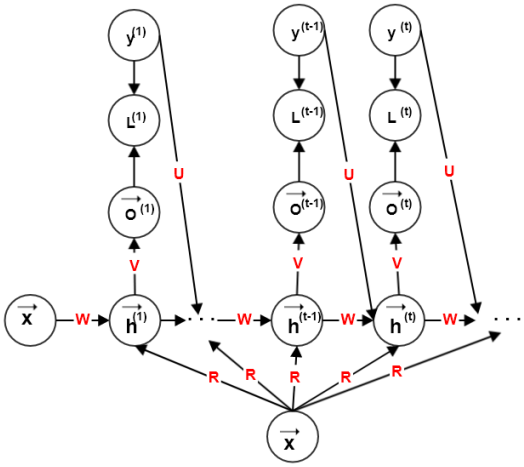
       .



x 作为初始状态



x 作为输入



x 同时作为初始状态和输入

2. 在图注任务中，单个图像作为模型输入，模型生成描述图像的单词序列。图像就是输入 $\vec{\mathbf{x}}$，它为每个时间步提供了一个输入。通过图像和 t 时刻为止的单词序列来预测 $t+1$ 时刻的单词。

   输出 $y^{(t)}$ 有两个作用：用作 $t+1$ 时刻的输入来预测 $y^{(t+1)}$；用于 t 时刻计算损失函数 $L^{(t)}$。

3. 当输入 $\vec{\mathbf{x}}$ 作为初始状态 $\vec{\mathbf{h}}^{(0)}$ 时，每个时间步也没有额外的输入。它与零输入`RNN`网络的区别在于：

   零输入`RNN`的初始输出 $y^{(0)}$ 是需要给定的，而这里的初始状态 $\vec{\mathbf{h}}^{(0)}$ 是给定的。

1.2.3 多长度输入序列

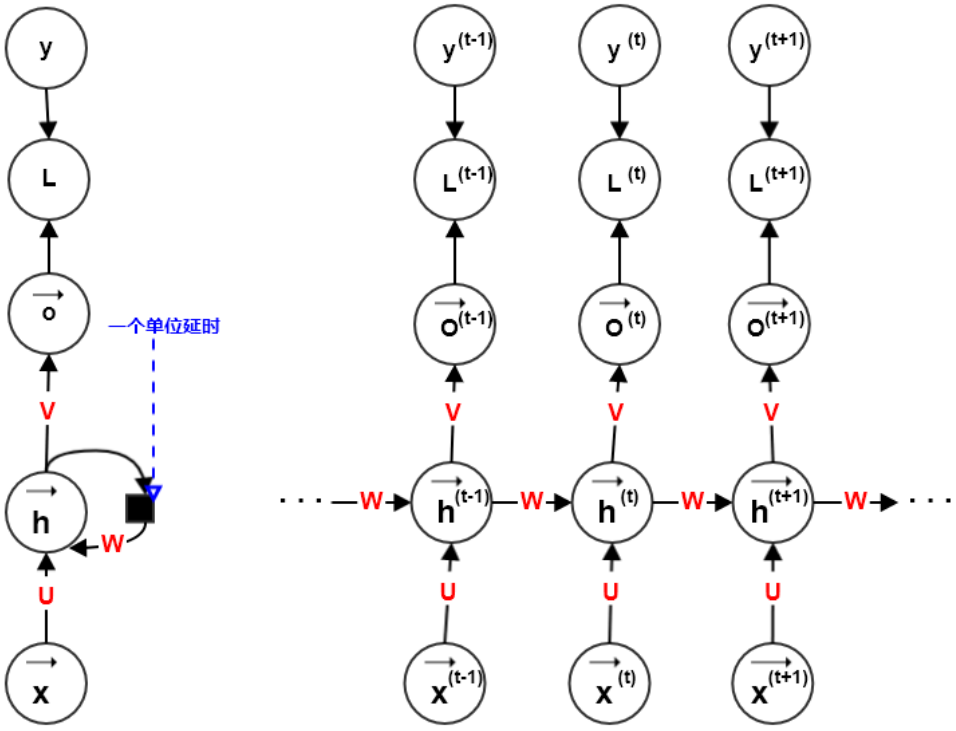
1. 多长度输入序列的`RNN`包含了`多输出&隐-隐连接RNN`、`多输出&输出-隐连接RNN`、`单输出&隐-隐连接RNN`等网络类型。

2. `多输出&隐-隐连接`循环网络：每个时间步都有输出，并且隐单元之间有循环连接。

   - 该网络将一个输入序列映射到相同长度的输出序列。
   - 模型的数学表示：$o_k^{(t)} = p(y^{(t)} = k \mid \vec{\mathbf{x}}^{(1)}, \cdots, \vec{\mathbf{x}}^{(t)}), \quad k = 1, 2, \cdots, K$
   - 单个样本的损失：$L = -\sum_{t=1}^{\tau} \sum_{k=1}^{K} \mathbb{I}_{(k=y^{(t)})} \log o_k^{(t)}$
   - 更新方程：

$$\vec{\mathbf{a}}^{(t)} = \vec{\mathbf{b}} + \mathbf{W}\vec{\mathbf{h}}^{(t-1)} + \mathbf{U}\vec{\mathbf{x}}^{(t)}$$
$$\vec{\mathbf{h}}^{(t)} = \tanh(\vec{\mathbf{a}}^{(t)})$$
$$\vec{\mathbf{o}}^{(t)} = \text{softmax}\left(\vec{\mathbf{c}} + \mathbf{V}\vec{\mathbf{h}}^{(t)}\right)$$

其中输入到隐状态的权重为 $\mathbf{U}$，隐状态到输出的权重为 $\mathbf{V}$，隐状态到隐状态的权重为 $\mathbf{W}$，$\vec{\mathbf{b}}, \vec{\mathbf{c}}$ 为输入偏置向量和输出偏置向量。

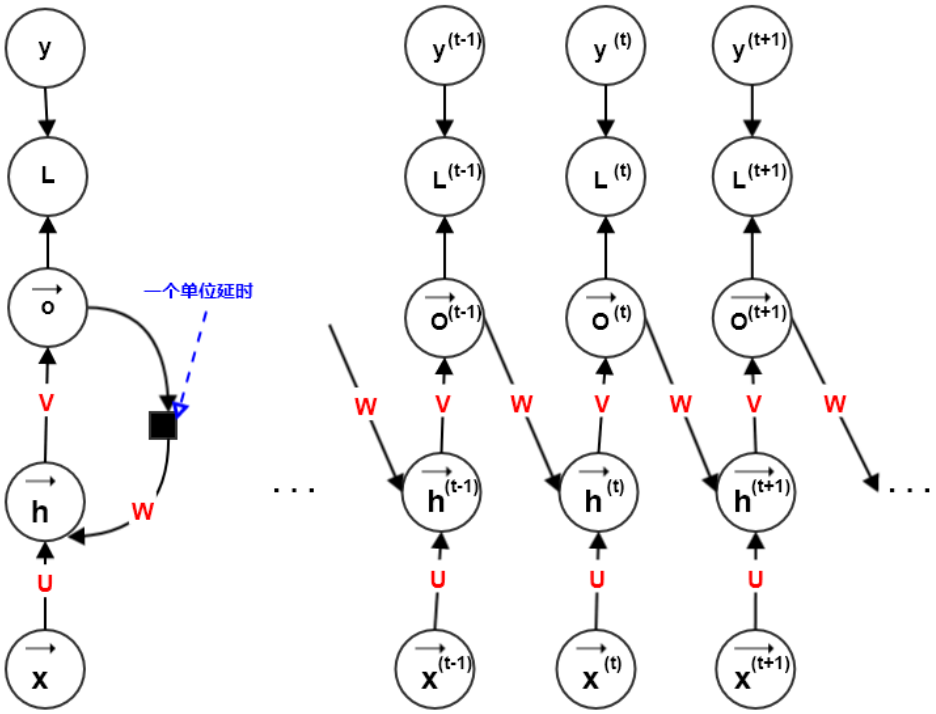


RNN：循环图　　RNN：展开图

3. 多输出&输出-隐连接循环网络：每个时间步都有输出，只有当前时刻的输出和下个时刻的隐单元之间有循环连接。
   - 该网络将一个输入序列映射到相同长度的输出序列。
   - 模型的数学表示：$o_k^{(t)} = p(y^{(t)} = k \mid \vec{\mathbf{x}}^{(1)}, \cdots, \vec{\mathbf{x}}^{(t)}), \quad k = 1, 2, \cdots, K$
   - 单个样本的损失：$L = -\sum_{t=1}^{\tau} \sum_{k=1}^{K} \mathbb{I}_{(k=y^{(t)})} \log o_k^{(t)}$
   - 更新方程：

$$\begin{aligned} \vec{\mathbf{a}}^{(t)} &= \vec{\mathbf{b}} + \mathbf{W}\vec{\mathbf{o}}^{(t-1)} + \mathbf{U}\vec{\mathbf{x}}^{(t)} \\ \vec{\mathbf{h}}^{(t)} &= \tanh(\vec{\mathbf{a}}^{(t)}) \\ \vec{\mathbf{o}}^{(t)} &= \text{softmax}\left(\vec{\mathbf{c}} + \mathbf{V}\vec{\mathbf{h}}^{(t)}\right) \end{aligned}$$

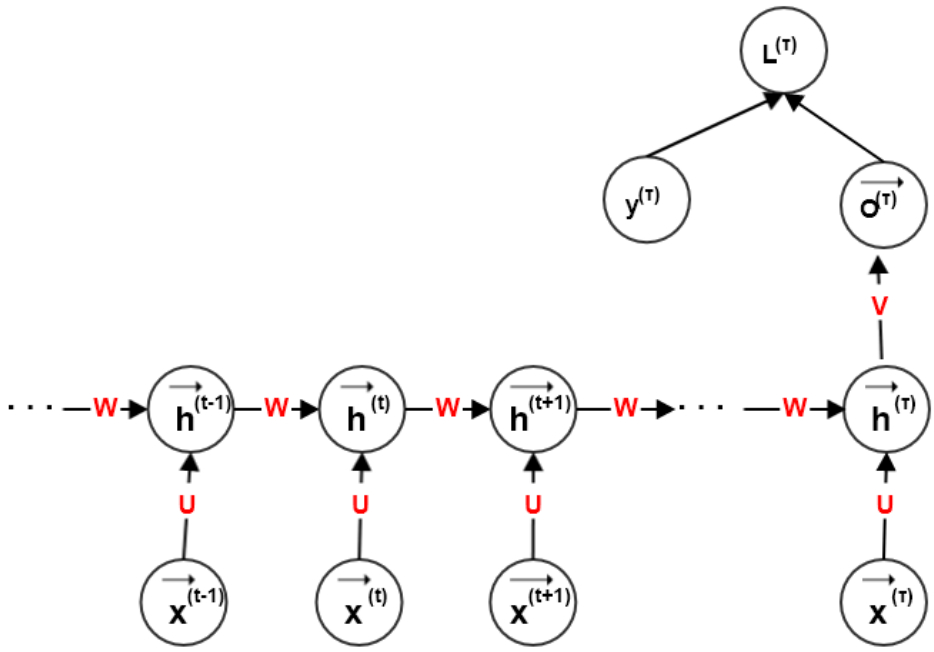
   其中输入到隐状态的权重为 $\mathbf{U}$，隐状态到输出的权重为 $\mathbf{V}$，输出到隐状态的权重为 $\mathbf{W}$，$\vec{\mathbf{b}}, \vec{\mathbf{c}}$ 为输入偏置向量和输出偏置向量。



4. 单输出&隐-隐连接 循环网络：隐单元之间存在循环连接，但是读取整个序列之后产生单个输出。
   - 单输出&隐-隐连接RNN 将一个输入序列映射到单个输出。
   - 模型的数学表示：$o_k^{(\tau)} = p(y^{(\tau)} = k \mid \vec{\mathbf{x}}^{(1)}, \cdots, \vec{\mathbf{x}}^{(\tau)}), \quad k = 1, 2, \cdots, K$
   - 单个样本的损失：$L = -\sum_{k=1}^{K} \mathbb{I}_{(k=y^{(\tau)})} \log o_k^{(\tau)}$
   - 更新方程：

$$\begin{aligned} \vec{\mathbf{a}}^{(t)} &= \vec{\mathbf{b}} + \mathbf{W}\vec{\mathbf{h}}^{(t-1)} + \mathbf{U}\vec{\mathbf{x}}^{(t)} \\ \vec{\mathbf{h}}^{(t)} &= \tanh(\vec{\mathbf{a}}^{(t)}) \\ \vec{\mathbf{o}}^{(\tau)} &= \text{softmax}\left(\vec{\mathbf{c}} + \mathbf{V}\vec{\mathbf{h}}^{(\tau)}\right) \end{aligned}$$

   其中输入到隐状态的权重为 $\mathbf{U}$，隐状态到输出的权重为 $\mathbf{V}$，隐状态到隐状态的权重为 $\mathbf{W}$，$\vec{\mathbf{b}}, \vec{\mathbf{c}}$ 为输入偏置向量和输出偏置向量。



RNN：展开图

5. 多输出&输出-隐连接循环网络比较于多输出&隐-隐连接循环网络，该网络的表达能力更小。
   - 多输出&隐-隐连接循环网络可以选择将其想要的关于过去的任何信息放入隐状态 $\vec{\mathbf{h}}$ 中，并且通过 $\vec{\mathbf{h}}$ 传播到未来。
   - 多输出&输出-隐连接循环网络中只有输出 $\vec{\mathbf{o}}$ 会被传播信息到未来。通常 $\vec{\mathbf{o}}$ 的维度远小于 $\vec{\mathbf{h}}$，并且缺乏过去的重要信息。
6. 多输出&输出-隐连接循环网络虽然表达能力不强，但是更容易训练：通过使用前一个时间步的真实标记 $y^{(t-1)}$ 来代替输出 $\vec{\mathbf{o}}^{(t-1)}$，使得每个时间步可以与其他时间步分离训练，从而允许训练期间更多的并行化。

1.3 输出序列长度

1. 对于输入序列长度为零或者为1的RNN模型，必须有某种办法来确定输出序列的长度。有三种方法来确定输出序列的长度：
   - 当输出是单词时，可以添加一个特殊的标记符。当输出遇到该标记符时，输出序列终止。

     此时需要改造训练集，对训练数据的每个输出序列末尾手工添加这个标记符。
   - 在模型中引入一个额外的二元输出单元，该输出单元用于指示：当前时间步是继续生成输出序列，还是停止生成。
     - 这种办法更普遍，适用于任何RNN。
     - 该二元输出单元通常使用sigmoid单元，被训练为最大化正确地预测到每个序列结束的对数似然。
   - 在模型中引入一个额外的输出单元，该输出单元预测输出序列的长度 τ 本身。
     - 这种方法需要在每个时间步的循环更新中增加一个额外输入，从而通知循环：是否已经到达输出序列的末尾。
     - 其原理是基于条件概率：$P(y^{(1)},y^{(2)},\cdots,y^{(\tau)})=P(\tau)P(y^{(1)},y^{(2)},\cdots,y^{(\tau)}\mid\tau)$。

二、训练算法

2.1 BPTT 算法

1. 以 多输出&隐-隐RNN 为例，设：输入到隐状态的权重为 $\mathbf{U}$，隐状态到输出的权重为 $\mathbf{V}$，隐状态到隐状态的权重为 $\mathbf{W}$，$\vec{\mathbf{b}},\vec{\mathbf{c}}$ 为输入偏置向量和输出偏置向量；激活函数为双曲正切激活函数 $\tanh$。

   设网络从特定的初始状态 $\vec{\mathbf{h}}^{(0)}$ 开始前向传播，从 $t=1$ 到 $t=\tau$ 的每个时间步，则有更新方程：

$$\vec{\mathbf{a}}^{(t)}=\vec{\mathbf{b}}+\mathbf{W}\vec{\mathbf{h}}^{(t-1)}+\mathbf{U}\vec{\mathbf{x}}^{(t)}$$
$$\vec{\mathbf{h}}^{(t)}=\tanh(\vec{\mathbf{a}}^{(t)})$$
$$\vec{\mathbf{o}}^{(t)}=\text{softmax}\left(\vec{\mathbf{c}}+\mathbf{V}\vec{\mathbf{h}}^{(t)}\right)$$

   多输出&隐-隐RNN 的单个样本损失函数为：$L=-\sum_{t=1}^{\tau}\sum_{k=1}^{K}\mathbb{I}_{(k=y^{(t)})}\log o_k^{(t)}$。该损失函数的梯度计算代价较高：

   - 因为每个时间步只能一前一后的计算无法并行化，因此时间复杂度为 $O(\tau)$。
   - 前向传播中各个状态必须保存直到它们反向传播中被再次使用，因此空间复杂度也是 $O(\tau)$。
   - 采用 `tanh` 激活函数而不是 `ReLU` 激活函数的原因是为了缓解长期依赖。

2. `back-propagation through time:BPTT`：通过时间反向传播算法，其算法复杂度为 $O(\tau)$。

   由 `BPTT` 计算得到梯度，再结合任何通用的、基于梯度的技术就可以训练 `RNN`。

3. 计算图的节点包括参数 $\mathbf{U},\mathbf{V},\mathbf{W},\vec{\mathbf{b}},\vec{\mathbf{c}}$，以及以 t 为索引的节点序列 $\vec{\mathbf{x}}^{(t)},y^{(t)},\vec{\mathbf{h}}^{(t)},\vec{\mathbf{o}}^{(t)}$ 以及 $L^{(t)}$。

   - 根据 $L=\sum_{t=1}^{\tau}L^{(t)}$，则有：$\frac{\partial L}{\partial L^{(t)}}=1$。
   - 令节点 $\vec{\mathbf{s}}^{(t)}=\vec{\mathbf{c}}+\mathbf{V}\vec{\mathbf{h}}^{(t)}$，则有：$\vec{\mathbf{o}}^{(t)}=\text{softmax}(\vec{\mathbf{s}}^{(t)})$。则有：

$$L^{(t)}=-\sum_{k=1}^{K}\mathbb{I}_{(k=y^{(t)})}\log o_k^{(t)}=-s_{y^{(t)}}^{(t)}+\log\sum_{k=1}^{K}\exp(s_k^{(t)})$$

     其中 $s_k^{(t)}$ 表示 $\vec{\mathbf{s}}^{(t)}$ 的第 k 个分量。

     则有：

$$\left(\nabla_{\vec{\mathbf{s}}^{(t)}}L\right)_k=\frac{\partial L}{\partial s_k^{(t)}}=\frac{\partial L}{\partial L^{(t)}}\times\frac{\partial L^{(t)}}{\partial s_k^{(t)}}=1\times\frac{\partial L^{(t)}}{\partial s_k^{(t)}}$$
$$=-\mathbb{I}_{k=y^{(t)}}+\frac{\exp(s_k^{(t)})}{\sum_{k'=1}^{K}\exp(s_{k'}^{(t)})}=o_k^{(t)}-\mathbb{I}_{k=y^{(t)}}$$

     $\left(\nabla_{\vec{\mathbf{s}}^{(t)}}L\right)_k$ 表示梯度 $\left(\nabla_{\vec{\mathbf{s}}^{(t)}}L\right)$ 的第 k 个分量，$\mathbb{I}(\cdot)$ 为示性函数。写成向量形式为：

$$\nabla_{\vec{\mathbf{s}}^{(t)}}L=\vec{\mathbf{o}}^{(t)}-\vec{\mathbf{y}}^{(t)}$$

     其中 $\vec{\mathbf{y}}^{(t)}=(0,\cdots,0,1,0,\cdots,0)$ 为真实标签 $y^{(t)}$ 扩充得到的概率分布，其真实的类别 $y_i^{(t)}$ 位置上的分量为 1，而其它位置上的分量为 0。

   - 根据定义 $\vec{\mathbf{h}}^{(t+1)}=\tanh(\vec{\mathbf{b}}+\mathbf{W}\vec{\mathbf{h}}^{(t)}+\mathbf{U}\vec{\mathbf{x}}^{(t+1)})$，得到：

$$h_i^{(t+1)}=\tanh\left(b_i+\sum_j W_{i,j}h_j^{(t)}+\sum_j U_{i,j}x_j^{(t+1)}\right)$$

     根据导数：$d\frac{\tanh(x)}{dx}=1-\tanh^2(x)$，则有：

$$\frac{\partial h_i^{(t+1)}}{\partial h_j^{(t)}}=\left(1-(h_i^{(t+1)})^2\right)W_{i,j}$$

     设隐向量长度为 n，定义：

$$\frac{\partial \vec{\mathbf{h}}^{(t+1)}}{\partial \vec{\mathbf{h}}^{(t)}}=\begin{bmatrix}\frac{\partial h_1^{(t+1)}}{\partial h_1^{(t)}} & \cdots & \frac{\partial h_n^{(t+1)}}{\partial h_1^{(t)}}\\ \vdots & \ddots & \vdots\\ \frac{\partial h_1^{(t+1)}}{\partial h_N^{(t)}} & \cdots & \frac{\partial h_n^{(t+1)}}{\partial h_1^{(t)}}\end{bmatrix}$$

$$\text{diag}\left(1-(\vec{\mathbf{h}}^{(t+1)})^2\right)=\begin{bmatrix}1-(h_1^{(t+1)})^2 & 0 & \cdots & 0\\ 0 & 1-(h_2^{(t+1)})^2 & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \cdots & 1-(h_n^{(t+1)})^2\end{bmatrix}$$

则有：$\frac{\partial \vec{\mathbf{h}}^{(t+1)}}{\partial \vec{\mathbf{h}}^{(t)}}=\text{diag}\left(1-(\vec{\mathbf{h}}^{(t+1)})^2\right)\mathbf{W}$ 。

根据定义 $\vec{\mathbf{s}}^{(t)}=\vec{\mathbf{c}}+\mathbf{V}\vec{\mathbf{h}}^{(t)}$，即 $s_i^{(t)}=c_i+\sum_j V_{i,j}h_j^{(t)}$ ，则有：$\frac{\partial s_i^{(t)}}{\partial h_j^{(t)}}=V_{i,j}$ ，记作：

$$\frac{\partial \vec{\mathbf{s}}^{(t)}}{\partial \vec{\mathbf{h}}^{(t)}}=\mathbf{V}$$

因此得到隐单元的梯度：

- 当 $t=\tau$ 时，$\vec{\mathbf{h}}^{(\tau)}$ 只有一个后续结点 $\vec{\mathbf{o}}^{(\tau)}$ （从而只有一个后继节点 $\vec{\mathbf{s}}^{(\tau)}$ ），因此有：

$$\nabla_{\vec{\mathbf{h}}^{(\tau)}}L=\left(\frac{\partial \vec{\mathbf{s}}^{(t)}}{\partial \vec{\mathbf{h}}^{(t)}}\right)^T\nabla_{\vec{\mathbf{s}}^{(t)}}L=\mathbf{V}^T\nabla_{\vec{\mathbf{s}}^{(\tau)}}L$$

- 当 $t<\tau$ 时，$\vec{\mathbf{h}}^{(t)}$ 同时具有 $\vec{\mathbf{o}}^{(t)},\vec{\mathbf{h}}^{(t+1)}$ 两个后续节点，因此有：

$$\begin{aligned}\nabla_{\vec{\mathbf{h}}^{(t)}}L&=\left(\frac{\partial \vec{\mathbf{h}}^{(t+1)}}{\partial \vec{\mathbf{h}}^{(t)}}\right)^T\nabla_{\vec{\mathbf{h}}^{(t+1)}}L+\left(\frac{\partial \vec{\mathbf{s}}^{(t)}}{\partial \vec{\mathbf{h}}^{(t)}}\right)^T\nabla_{\vec{\mathbf{s}}^{(t)}}L\\&=\mathbf{W}^T(\nabla_{\vec{\mathbf{h}}^{(t+1)}}L)\text{diag}\left(1-(\vec{\mathbf{h}}^{(t+1)})^2\right)+\mathbf{V}^T\nabla_{\vec{\mathbf{s}}^{(t)}}L\end{aligned}$$

由于 $\nabla_{\vec{\mathbf{h}}^{(t)}}L$ 依赖于 $\nabla_{\vec{\mathbf{h}}^{(t+1)}}L$，因此求解隐单元的梯度时，从末尾开始反向计算。

4. 一旦获得了隐单元及输出单元的梯度，则可以获取参数节点的梯度。

注意：由于参数在多个时间步共享，因此在参数节点的微分操作时必须谨慎对待。

微分中的算子 $\nabla_{\mathbf{A}}f$ 在计算 $\mathbf{A}$ 对于 f 的贡献时，将计算图的所有边都考虑进去了。但是事实上：有一条边是 t 时间步的 $\mathbf{A}$，还有一条边是 $t+1$ 时间步的 $\mathbf{A}$ ，.... 。

为了消除歧义，使用虚拟变量 $\mathbf{A}^{(t)}$ 作为 $\mathbf{A}$ 的副本。用 $\nabla_{\mathbf{A}^{(t)}}f$ 表示参数 $\mathbf{A}$ 在时间步 t 对于梯度的贡献。将所有时间步上的梯度相加，即可得到 $\nabla_{\mathbf{A}}f$ 。

5. 根据定义 $\vec{\mathbf{s}}^{(t)}=\vec{\mathbf{c}}^{(t)}+\mathbf{V}\vec{\mathbf{h}}^{(t)}$ ，即 $s_i^{(t)}=c_i+\sum_j V_{i,j}^{(t)}h_j^{(t)}$ 。则有：

$$\frac{\partial \vec{\mathbf{s}}^{(t)}}{\partial \vec{\mathbf{c}}^{(t)}}=\mathbf{I},\quad \frac{\partial s_i^{(t)}}{\partial V_{i,j}^{(t)}}=h_j^{(t)}$$

- 考虑到 $\vec{\mathbf{c}}$ 对于每个输出 $\vec{\mathbf{o}}^{(1)},\cdots,\vec{\mathbf{o}}^{(\tau)}$ 都有贡献，因此有：

$$\nabla_{\vec{\mathbf{c}}}L=\sum_{t=1}^{t=\tau}\left(\frac{\partial \vec{\mathbf{s}}^{(t)}}{\partial \vec{\mathbf{c}}^{(t)}}\right)^T\nabla_{\vec{\mathbf{s}}^{(t)}}L=\sum_{t=1}^{t=\tau}\nabla_{\vec{\mathbf{s}}^{(t)}}L$$

- 记：

$$\nabla_{V_{k,:}^{(t)}}s_i^{(t)}=\begin{cases}\vec{\mathbf{h}}^{(t)}, & i=k\\ \vec{\mathbf{0}}, & i\neq k\end{cases}$$

考虑到 $\mathbf{V}$ 对于每个输出 $\vec{\mathbf{o}}^{(1)},\cdots,\vec{\mathbf{o}}^{(\tau)}$ 都有贡献，因此有：

$$\nabla_{V_{i,:}} L=\sum_{t=1}^{t=\tau}\left(\frac{\partial L}{\partial s_i^{(t)}}\right)\nabla_{V_{i,:}^{(t)}} s_i^{(t)}=\sum_{t=1}^{t=\tau}(\nabla_{\vec{\mathbf{s}}^{(t)}} L)_i\vec{\mathbf{h}}^{(t)}$$

其中 $(\nabla_{\vec{\mathbf{s}}^{(t)}} L)_i$ 表示 $\nabla_{\vec{\mathbf{s}}^{(t)}} L$ 的第 i 个分量。

6. 根据定义 $\vec{\mathbf{h}}^{(t)}=\tanh(\vec{\mathbf{b}}^{(t)}+\mathbf{W}\vec{\mathbf{h}}^{(t-1)}+\mathbf{U}\vec{\mathbf{x}}^{(t)})$，即：

$$h_i^{(t)}=\tanh\left(b_i+\sum_j W_{i,j}^{(t)}h_j^{(t-1)}+\sum_j U_{i,j}x_j^{(t)}\right)$$

则有：

$$\frac{\partial\vec{\mathbf{h}}^{(t)}}{\partial\vec{\mathbf{b}}^{(t)}}=\operatorname{diag}\left(1-(\vec{\mathbf{h}}^{(t)})^2\right),\quad\frac{\partial h_i^{(t)}}{\partial W_{i,j}^{(t)}}=(1-h_i^{(t)2})h_j^{(t-1)},\quad\frac{\partial h_i^{(t)}}{\partial U_{i,j}^{(t)}}=(1-h_i^{(t)2})x_j^{(t)}$$

- 考虑到 $\vec{\mathbf{b}}$ 对于每个隐向量 $\vec{\mathbf{h}}^{(1)},\cdots,\vec{\mathbf{h}}^{(\tau)}$ 都有贡献，因此有：

$$\nabla_{\vec{\mathbf{b}}} L=\sum_{t=1}^{t=\tau}\left(\frac{\partial\vec{\mathbf{h}}^{(t)}}{\partial\vec{\mathbf{b}}^{(t)}}\right)^T\nabla_{\vec{\mathbf{h}}^{(t)}} L=\sum_{t=1}^{t=\tau}\operatorname{diag}\left(1-(\vec{\mathbf{h}}^{(t)})^2\right)\nabla_{\vec{\mathbf{h}}^{(t)}} L$$

- 记：

$$\nabla_{W_{k,:}^{(t)}} h_i^{(t)}=\begin{cases}(1-h_i^{(t)2})\vec{\mathbf{h}}^{(t-1)}, & i=k\\ \vec{\mathbf{0}}, & i\neq k\end{cases}$$

考虑到每个 $\mathbf{W}^{(t)}$ 都对 L 有贡献，则：

$$\nabla_{W_{i,:}} L=\sum_{t=1}^{t=\tau}\left(\frac{\partial L}{\partial h_i^{(t)}}\right)\nabla_{W_{i,:}^{(t)}} h_i^{(t)}=\sum_{t=1}^{t=\tau}(\nabla_{\vec{\mathbf{h}}^{(t)}} L)_i\left(1-h_i^{(t)2}\right)\vec{\mathbf{h}}^{(t-1)}$$

其中 $(\nabla_{\vec{\mathbf{h}}^{(t)}} L)_i$ 表示 $\nabla_{\vec{\mathbf{h}}^{(t)}} L$ 的第 i 个分量。

- 记：

$$\nabla_{U_{k,:}^{(t)}} h_i^{(t)}=\begin{cases}(1-h_i^{(t)2})\vec{\mathbf{x}}^{(t)}, & i=k\\ \vec{\mathbf{0}}, & i\neq k\end{cases}$$

考虑到每个 $\mathbf{U}^{(t)}$ 都对 L 有贡献，则：

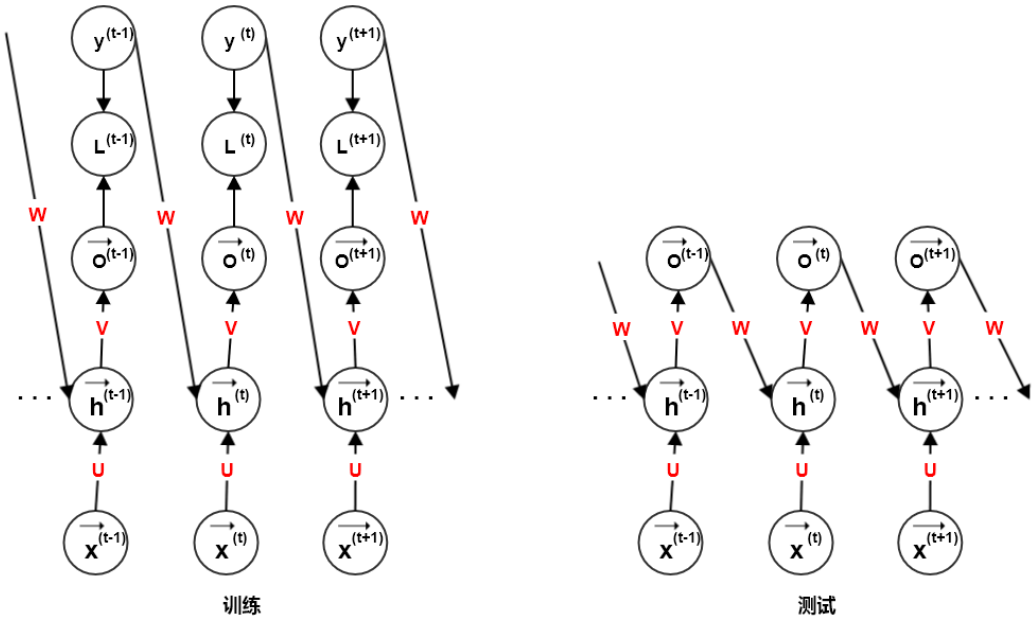
$$\nabla_{U_{i,:}} L=\sum_{t=1}^{t=\tau}\left(\frac{\partial L}{\partial h_i^{(t)}}\right)\nabla_{U_{i,:}^{(t)}} h_i^{(t)}=\sum_{t=1}^{t=\tau}(\nabla_{\vec{\mathbf{h}}^{(t)}} L)_i\left(1-h_i^{(t)2}\right)\vec{\mathbf{x}}^{(t)}$$

其中 $(\nabla_{\vec{\mathbf{h}}^{(t)}} L)_i$ 表示 $\nabla_{\vec{\mathbf{h}}^{(t)}} L$ 的第 i 个分量。

7. 因为任何参数都不是训练数据 $\vec{\mathbf{x}}^{(t)}$ 的父节点，因此不需要计算 $\nabla_{\vec{\mathbf{x}}^{(t)}} L$ 。

2.2 Teacher forcing 算法

1. `多输出&输出-隐连接RNN`模型可以使用 `teacher forcing` 算法进行训练。
   - 模型的数学表示：$o_k^{(t)}=p(y^{(t)}=k\mid y^{(t-1)},\vec{\mathbf{x}}^{(t)}),\quad k=1,2,\cdots,K$
   - 单个样本的损失：$L=-\sum_{t=1}^{\tau}\sum_{k=1}^{K}\mathbb{I}_{(k=y^{(t)})}\log o_k^{(t)}$
   - 训练时：在时刻 $t+1$ 接受真实类别分布 $\vec{\mathbf{y}}^{(t)}$ 作为输入，而不必等待 t 时刻的模型输出分布 $\vec{\mathbf{o}}^{(t)}$。
   - 推断时：真实的标记通常是未知的，因此必须用模型的输出分布 $\vec{\mathbf{o}}^{(t)}$ 。



2. `teacher forcing` 训练的本质原因是：当前隐状态与早期隐状态没有直接连接。虽然有间接连接，但是由于 $y^{(t)}$ 已知，因此这种连接被切断。
    - 如果模型的隐状态依赖于早期时间步的隐状态，则需要采用 `BPTT` 算法。
    - 某些模型训练时，需要同时使用 `teacher forcing` 和 `BPTT` 算法。

三、长期依赖

3.1 长期依赖

1. 长期依赖的问题是深度学习中的一个主要挑战，其产生的根本问题是：经过许多阶段传播之后，梯度趋向于消失或者爆炸。
    - 长期依赖的问题中，梯度消失占大部分情况，而梯度爆炸占少数情况。但是梯度爆炸一旦发生，就优化过程影响巨大。
    - `RNN` 涉及到许多相同函数的多次复合作用，每个时间步一次。这种复合作用可以导致极端的非线性行为。因此在 `RNN` 中，长期依赖问题表现得尤为突出。
2. 考虑一个没有非线性、没有偏置非常简单的循环结构：$\vec{\mathbf{h}}^{(t)}=\mathbf{W}\vec{\mathbf{h}}^{(t-1)}$ 。则有：

$$\vec{\mathbf{h}}^{(t)}=\mathbf{W}^t\vec{\mathbf{h}}^{(0)}$$

$$\frac{\partial\vec{\mathbf{h}}^{(t)}}{\partial\vec{\mathbf{h}}^{(t-1)}}=\mathbf{W},\quad \frac{\partial\vec{\mathbf{h}}^{(t)}}{\partial\vec{\mathbf{h}}^{(0)}}=\mathbf{W}^t$$

$$\nabla_{\vec{\mathbf{h}}^{(0)}}L=\frac{\partial\vec{\mathbf{h}}^{(t)}}{\partial\vec{\mathbf{h}}^{(0)}}\nabla_{\vec{\mathbf{h}}^{(t)}}L=\mathbf{W}^t\nabla_{\vec{\mathbf{h}}^{(t)}}L$$

设 $\mathbf{W}$ 可以正交分解时： $\mathbf{W}=\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T$ 。其中 $\mathbf{Q}$ 为正交矩阵，$\mathbf{\Lambda}$ 为特征值组成的三角阵。则：

$$\vec{\mathbf{h}}^{(t)}=\mathbf{Q}\mathbf{\Lambda}^t\mathbf{Q}^T\vec{\mathbf{h}}^{(0)}$$

$$\nabla_{\vec{\mathbf{h}}^{(0)}}L=\frac{\partial\vec{\mathbf{h}}^{(t)}}{\partial\vec{\mathbf{h}}^{(0)}}\nabla_{\vec{\mathbf{h}}^{(t)}}L=\mathbf{Q}\mathbf{\Lambda}^t\mathbf{Q}^T\nabla_{\vec{\mathbf{h}}^{(t)}}L$$

    - 前向传播：
        - 对于特征值的幅度不到 1 的特征值对应的 $\vec{\mathbf{h}}^{(0)}$ 的部分将随着 t 衰减到 0 。
        - 对于特征值的幅度大于 1 的特征值对应的 $\vec{\mathbf{h}}^{(0)}$ 的部分将随着 t 指数级增长。

- 反向传播：

- 对于特征值幅度不到1的梯度的部分将随着 t 衰减到 0 。
- 对于特征值幅度大于1的梯度的部分将随着 t 指数级增长 。

4. 若考虑非线性和偏置，即： $\vec{\mathbf{h}}^{(t+1)}=\tanh(\vec{\mathbf{b}}+\mathbf{W}\vec{\mathbf{h}}^{(t)}+\mathbf{U}\vec{\mathbf{x}}^{(t+1)})$ ，有：

$$\frac{\partial\vec{\mathbf{h}}^{(t+1)}}{\partial\vec{\mathbf{h}}^{(t)}}=\text{diag}\left(1-(\vec{\mathbf{h}}^{(t+1)})^2\right)\mathbf{W}$$

   - 前向传播：

     由于每一级的 $\vec{\mathbf{h}}$ 的幅度被 $\tanh(\cdot)$ 函数限制在 `(-1,1)` 之间，因此前向传播并不会指数级增长。

     这也是为什么 `RNN` 使用 `tanh` 激活函数，而不使用 `relu` 的原因。

   - 反向传播：

     由于隐状态的幅度被 $\tanh(\cdot)$ 函数限制在 `(-1,1)` 之间，因此 $\text{diag}\left(1-(\vec{\mathbf{h}}^{(t+1)})^2\right)\mathbf{W}$ 对 $\mathbf{W}$ 进行了一定程度上的缩小。 $\vec{\mathbf{h}}^{(t+1)}$ 越大，结果越小。

     - 如果 $\mathbf{W}$ 的特征值经过这样的缩小之后，在每个时刻都远小于1（因为每个时刻缩小的比例会变化），则该梯度部分将衰减到 0 。
     - 如果 $\mathbf{W}$ 的特征值经过这样的缩小之后，在每个时刻都远大于1，则该梯度部分将指数级增长。
     - 如果 $\mathbf{W}$ 的特征值经过这样的缩小之后，在不同的时刻有时候小于1有时候大于1（因为每个时刻缩小的比例会变化），则该梯度部分将比较平稳。

5. 对于非循环神经网络，长期依赖的情况稍好。

   - 对于标量权重 w，假设每个时刻使用不同的权重 $w^{(t)}$ 。假设 $w^{(t)}$ 是独立同分布的随机变量，均值为 0、方差为 v ，则 $\prod_t w^{(t)}$ 的方差为 $O(v^n)$ 。
   - 非常深的前馈神经网络通过精心设计可以避免梯度消失和梯度爆炸问题。

3.2 多时间尺度

1. 缓解长期依赖的一个策略是：设计多个时间尺度的模型：在细粒度的时间尺度上处理近期信息、在粗粒度时间尺度上处理远期的信息。

2. 得到粗粒度时间尺度的一种方法是跳跃连接：增加从远期的隐变量到当前隐变量的直接连接。

   - 普通的`RNN`中循环从时刻 t 隐变量连接到了时刻 $t+1$ 隐变量，跳跃连接会增加一条从时刻 t 到时刻 $t+d$ 隐变量的连接。

     注意：是增加而不是替代。

   - 引入了 d 延时的循环连接可以减轻梯度消失的问题。

     现在梯度指数降低的速度与 $\frac{\tau}{d}$ 相关，而不是与 τ 相关。这允许算法捕捉到更长时间的依赖性。但是这种做法无法缓解梯度指数级爆炸的问题。

3. 得到粗粒度时间尺度的另一种方法是删除连接：主动删除时间跨度为 1 的连接，并用更长的连接替换。

   删除连接与跳跃连接的区别：

   - 删除连接不会增加计算图中的连接，而跳跃连接会增加计算图中的连接。
   - 删除连接强迫单元在长时间尺度上工作；而跳跃连接可以选择在长时间尺度上工作，也可以在短时间尺度上工作。

3.3 渗漏单元

1. 缓解梯度爆炸和梯度消失的一个方案是：尽可能的使得梯度接近1。这可以通过线性自连接单元来实现。

如：（其中 $h^{(t)}$ 为隐单元， $x^{(t)}$ 为输入）

$$h^{(t)} = \alpha h^{(t-1)} + (1-\alpha)x^{(t)}$$

- 当 α 接近1 时， $h^{(t)}$ 能记住过去很长一段时间的输入信息
- 当 α 接近 0 时，$h^{(t)}$ 只能记住附近的一小段输入信息。

拥有类似行为的隐单元称作渗漏单元。

2. 渗漏单元与跳跃连接的区别：
   - d 时间步的跳跃连接：可以确保隐单元总能够被 d 个时间步之前的输入值所影响。
   - 参数为 α 的渗漏单元：通过调整 α 值，可以更灵活的确保隐单元访问到过去不同时间步的输入值。
3. 渗漏单元和跳跃连接的 α, d 参数有两种设置方式：
   - 手动设置为常数。如：初始化时从某些分布采样它们的值。
   - 让它们成为可训练的变量，从训练中学习出来。
4. 可以使得不同的循环单元在不同时间尺度上工作：
   - 手动设置不同的循环单元具有不同的 α, d 参数。
   - 虽然不同的循环单元具有相同的 α, d 参数，但是在梯度下降的参数更新中，显式使得不同循环单元的参数采用不同的更新频率。

3.4 梯度截断

1. 对于长期依赖问题中的梯度爆炸，最常用的解决方案是梯度截断。

   梯度截断有两种方案，设梯度 $\vec{\mathbf{g}} = (g_1, g_2, \cdots, g_n)^T$ ：

   - 在更新参数之前，逐元素的截断参数梯度，其中 v 为 g_i 的上界：

   $$g_i = \begin{cases} g_i & ,if\ g_i <= v \\ \text{sign}(g_i) \times v & ,else \end{cases}$$

   - 在更新参数之前，截断梯度的范数，其中 v 是梯度范数的上界：

   $$\vec{\mathbf{g}} = \begin{cases} \vec{\mathbf{g}} & ,if\ ||\vec{\mathbf{g}}|| <= v \\ \frac{\vec{\mathbf{g}} \times v}{||\vec{\mathbf{g}}||} & ,else \end{cases}$$

   第二种方案可以确保截断后的梯度仍然是在正确的梯度方向上。但是实践表明：两种方式的效果相近。因为逐元素的梯度截断时，梯度更新的方向不仅不再是真实梯度方向，甚至也不是`mini-batch`的梯度方向。但是它仍然是一个使得目标值下降的方向。

2. 当梯度大小超过了阈值时，即使采用简单的随机步骤，效果往往也非常好。即：随机采用大小为 v 的向量来作为梯度。因为这样通常会使得梯度离开数值不稳定的状态。

3. 如果在`mini-batch`梯度下降中应用了梯度范数截断，则真实梯度的方向不再等于所有`mini-batch`梯度的平均。

   对于一个`mini-batch`，梯度范数截断不会改变它的梯度方向。对于许多个`mini-batch`，使用梯度范数截断之后，它们的平均值并不等同于真实梯度的范数截断。

   因此使用范数截断的`mini-batch`梯度下降，引入了额外的梯度误差。这种误差有助于随机梯度下降算法逃离局部极小值。

3.5 引导信息流的正则化

1. 梯度截断有助于解决梯度爆炸，但是无助于解决梯度消失。为解决梯度消失，有两种思路：
   - 让路径的梯度乘积接近1 ，如`LSTM`及其他门控机制。
   - 正则化或者约束参数，从而引导信息流。
2. 正则化引导信息流：希望梯度向量 $\nabla_{\vec{\mathbf{h}}^{(t)}} L$ 在反向传播时能维持其幅度。即：希望 $\nabla_{\vec{\mathbf{h}}^{(t-1)}} L$ 与 $\nabla_{\vec{\mathbf{h}}^{(t)}} L$ 尽可能一样大。

考虑到

$$\nabla_{\vec{\mathbf{h}}^{(t-1)}} L = \left(\frac{\partial \vec{\mathbf{h}}^{(t)}}{\partial \vec{\mathbf{h}}^{(t-1)}}\right)^T \nabla_{\vec{\mathbf{h}}^{(t)}} L$$

`Pascanu et al.` 给出了以下正则项：

$$\Omega = \sum_t \left(\frac{||\left(\frac{\partial \vec{\mathbf{h}}^{(t)}}{\partial \vec{\mathbf{h}}^{(t-1)}}\right)^T \nabla_{\vec{\mathbf{h}}^{(t)}} L||}{||\nabla_{\vec{\mathbf{h}}^{(t)}} L||} - 1 \right)^2$$

- 计算这个正则项可能比较困难。`Pascanu et al.` 提出：可以将反向传播梯度 $\nabla_{\vec{\mathbf{h}}^{(t)}} L$ 考虑作为恒值来近似。
- 实验表明：如果与梯度截断相结合，该正则项可以显著增加 `RNN` 可以学习的依赖跨度。
- 该方法的一个主要缺点是：在处理数据冗余的任务时，如语言模型，它并不像 `LSTM` 一样有效 。

四、常见 RNN 变种

4.1 双向 RNN

1. 前面介绍的 `RNN` 网络隐含了一个假设：时刻 t 的状态只能由过去的输入序列 $\{\vec{\mathbf{x}}^{(1)}, \vec{\mathbf{x}}^{(2)}, \cdots, \vec{\mathbf{x}}^{(t-1)}\}$，以及当前的输入 $\vec{\mathbf{x}}^{(t)}$ 来决定。

   实际应用中，网络输出 $\vec{\mathbf{o}}^{(t)}$ 可能依赖于整个输入序列。如：语音识别任务中，当前语音对应的单词不仅取决于前面的单词，也取决于后面的单词。因为词与词之间存在语义依赖。

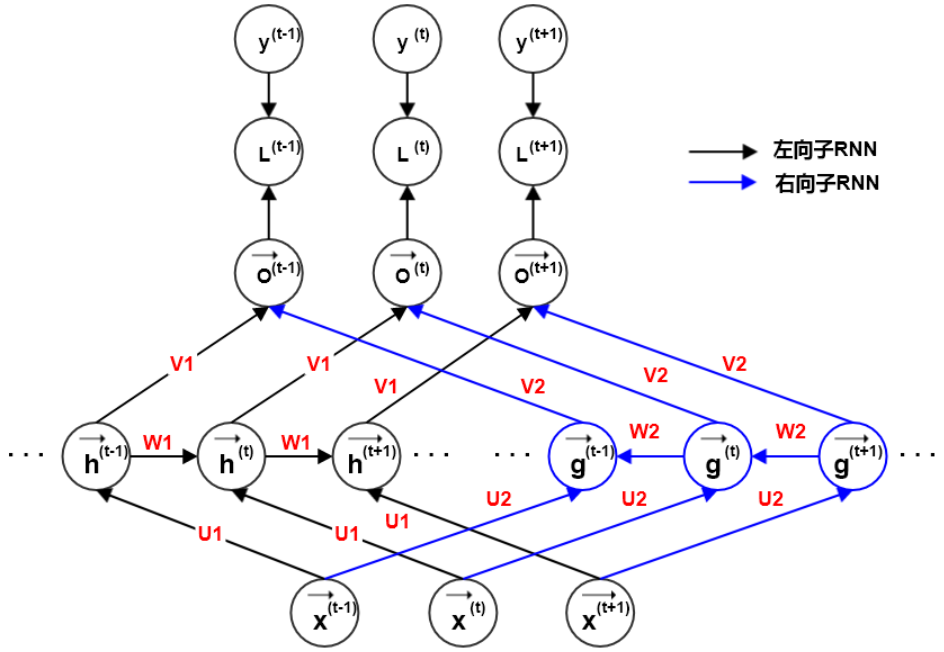
   双向 `RNN` 就是为了解决这种双向依赖问题，它在需要双向信息的应用中非常成功。如：手写识别、语音识别等。

2. 典型的双向 `RNN` 具有两条子 `RNN`：

   - $\vec{\mathbf{h}}^{(t)}$ 代表通过时间向未来移动的子 `RNN` 的状态，向右传播信息；$\vec{\mathbf{g}}^{(t)}$ 代表通过时间向过去移动的子 `RNN` 的状态，向左传播信息。
   - t 时刻的输出 $\vec{\mathbf{o}}^{(t)}$ 同时依赖于过去、未来、以及时刻 t 的输入 $\vec{\mathbf{x}}^{(t)}$ 。
   - 模型的数学表示：$o_k^{(t)} = p(y^{(t)} = k \mid \vec{\mathbf{x}}^{(1)}, \cdots, \vec{\mathbf{x}}^{(t)}), \quad k = 1, 2, \cdots, K$
   - 单个样本的损失：$L = -\sum_{t=1}^{\tau} \sum_{k=1}^{K} \mathbb{I}_{(k=y^{(t)})} \log o_k^{(t)}$
   - 更新方程：

   $$\begin{aligned}
   \vec{\mathbf{a}}_1^{(t)} &= \vec{\mathbf{b}}_1 + \mathbf{W}_1 \vec{\mathbf{h}}^{(t-1)} + \mathbf{U}_1 \vec{\mathbf{x}}^{(t)} \\
   \vec{\mathbf{a}}_2^{(t)} &= \vec{\mathbf{b}}_2 + \mathbf{W}_2 \vec{\mathbf{g}}^{(t+1)} + \mathbf{U}_2 \vec{\mathbf{x}}^{(t)} \\
   \vec{\mathbf{h}}^{(t)} &= \tanh(\vec{\mathbf{a}}_1^{(t)}), \quad \vec{\mathbf{g}}^{(t)} = \tanh(\vec{\mathbf{a}}_2^{(t)}) \\
   \vec{\mathbf{o}}^{(t)} &= \text{softmax}\left(\vec{\mathbf{c}} + \mathbf{V}_1 \vec{\mathbf{h}}^{(t)} + \mathbf{V}_2 \vec{\mathbf{g}}^{(t)}\right)
   \end{aligned}$$

   其中输入到隐状态的权重为 $\mathbf{U}_1, \mathbf{U}_2$，隐状态到输出的权重为 $\mathbf{V}_1, \mathbf{V}_2$ ，隐状态到隐状态的权重为 $\mathbf{W}_1, \mathbf{W}_2$ ，$\vec{\mathbf{b}}_1, \vec{\mathbf{b}}_2, \vec{\mathbf{c}}$ 为输入偏置向量和输出偏置向量。



3. 如果输入是 2 维的（如图像），则双向 RNN 可以扩展到4个方向：上、下、左、右。

   每个子 RNN 负责一个时间移动方向，t 时刻的输出 $\vec{\mathbf{o}}^{(t)}$ 同时依赖于四个方向、以及时刻 t 的输入 $\vec{\mathbf{x}}^{(t)}$ 。

   与 CNN 相比：

   - RNN 可以捕捉到大多数局部信息，还可以捕捉到依赖于远处的信息；CNN 只能捕捉到卷积窗所在的局部信息。
   - RNN 计算成本通常更高，而 CNN 的计算成本较低。

4.2 深度 RNN

1. 前述 RNN 中的计算都可以分解为三种变换：从输入 $\vec{\mathbf{x}}^{(t)}$ 到隐状态 $\vec{\mathbf{h}}^{(t)}$ 的变换、从前一个隐状态 $\vec{\mathbf{h}}^{(t)}$ 到下一个隐状态 $\vec{\mathbf{h}}^{(t+1)}$ 的变换、从隐状态 $\vec{\mathbf{h}}^{(t)}$ 到输出 $\vec{\mathbf{o}}^{(t)}$ 的变换。这三个变换都是浅层的，即：由一个仿射变换加一个激活函数组成。

   事实上，可以对这三种变换中引入深度。实验表明：引入深度会带来好处。

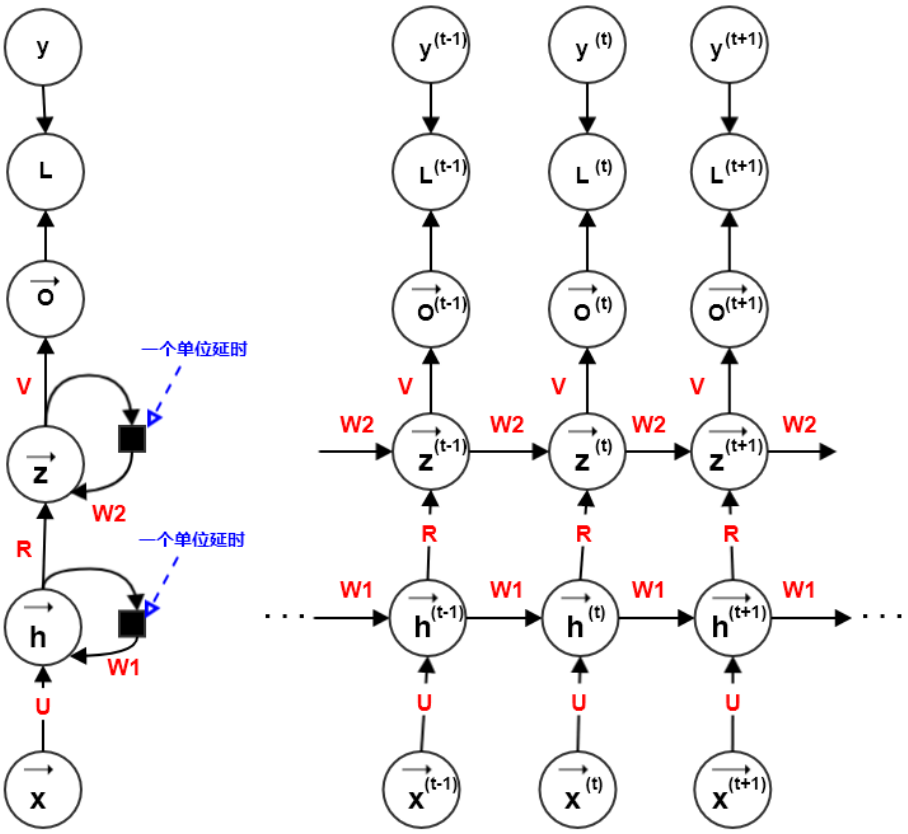
   - 方式一：通过将 RNN 的隐状态分为多层来引入深度。
   - 方式二：在这三种变换中，各自使用一个独立的 MLP（可能是较浅的，也可能是较深的）。
   - 方式三：在第二种方式的基础上，类似 ResNet 的思想，在 “隐状态-隐状态” 的路径中引入跳跃连接。

2. 通过将 RNN 的隐状态分为多层来引入深度：如下所示，隐状态有两层：$\vec{\mathbf{h}}^{(t)}$ 和 $\vec{\mathbf{z}}^{(t)}$ 。隐状态层中层次越高，对输入提取的概念越抽象。

   - 模型的数学表示：$o_k^{(t)} = p(y^{(t)} = k \mid \vec{\mathbf{x}}^{(1)}, \cdots, \vec{\mathbf{x}}^{(t)}), \quad k = 1, 2, \cdots, K$
   - 单个样本的损失：$L = -\sum_{t=1}^{\tau} \sum_{k=1}^{K} \mathbb{I}_{(k=y^{(t)})} \log o_k^{(t)}$
   - 更新方程：

$$\begin{aligned}
\vec{\mathbf{a}}_1^{(t)} &= \vec{\mathbf{b}}_1 + \mathbf{W}_1 \vec{\mathbf{h}}^{(t-1)} + \mathbf{U}\vec{\mathbf{x}}^{(t)} \\
\vec{\mathbf{h}}^{(t)} &= \tanh(\vec{\mathbf{a}}_1^{(t)}) \\
\vec{\mathbf{a}}_2^{(t)} &= \vec{\mathbf{b}}_2 + \mathbf{W}_2 \vec{\mathbf{z}}^{(t-1)} + \mathbf{R}\vec{\mathbf{h}}^{(t)} \\
\vec{\mathbf{z}}^{(t)} &= \tanh(\vec{\mathbf{a}}_2^{(t)}) \\
\vec{\mathbf{o}}^{(t)} &= \text{softmax}\left(\vec{\mathbf{c}} + \mathbf{V}\vec{\mathbf{z}}^{(t)}\right)
\end{aligned}$$

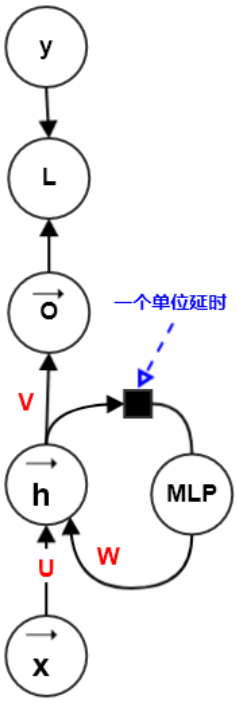
其中输入到隐状态的权重为 $\mathbf{U}$，隐状态到输出的权重为 $\mathbf{V}$，隐状态到隐状态的权重为 $\mathbf{W}_1, \mathbf{W}_2, \mathbf{R}$，$\vec{\mathbf{b}}_1, \vec{\mathbf{b}}_2, \vec{\mathbf{c}}$ 为输入偏置向量和输出偏置向量。



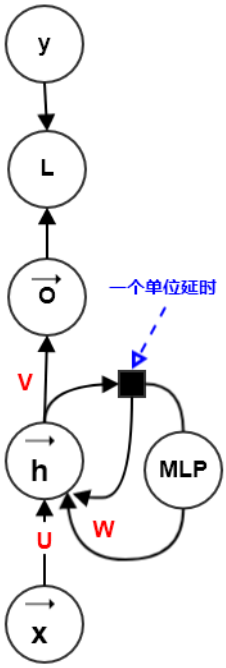
RNN：循环图　　RNN：展开图

3. 在这三种变换中，各自使用一个独立的`MLP`（可能是深度的），如下图所示。

   该方法有一个主要问题：额外深度将导致从时间步 t 到时间步 $t+1$ 的最短路径变得更长，这可能导致优化困难而破坏学习效果。



4. 在第二种方式的基础上，类似`ResNet`的思想，在“隐状态-隐状态”的路径中引入跳跃连接，从而缓解最短路径变得更长的问题。



4.3 LSTM 和 GRU

1. 目前实际应用中最有效的序列模型是门控 RNN，包括基于 LSTM: long short-term memory 的循环网络，和基于门控循环单元 GRU: gated recurrent unit 的循环网络。

   围绕门控 RNN 这一主题可以设计更多的变种。然而一些调查发现：这些 LSTM 和 GRU 架构的变种，在广泛的任务中难以明显的同时击败这两个原始架构。

2. 门控 RNN 的思路和渗漏单元一样：生成通过时间的快捷路径，使得梯度既不消失也不爆炸。
   - 可以手动选择常量的连接权重来实现这个目的，如跳跃连接。权重为固定的常量，且不随时间改变。
   - 可以使用参数化的连接权重来实现这个目的，如渗漏单元。权重是样本的函数，且不随时间改变。
   - 门控 RNN 将其推广为：连接权重在每个时间步都可能改变。权重是样本和时间的函数，随时间改变。
3. 渗漏单元允许网络在较长持续时间内积累信息，但它有个缺点：有时候希望一旦某个信息被使用（即：被消费掉了），那么这个信息就要被遗忘（丢掉它，使得它不再继续传递）。

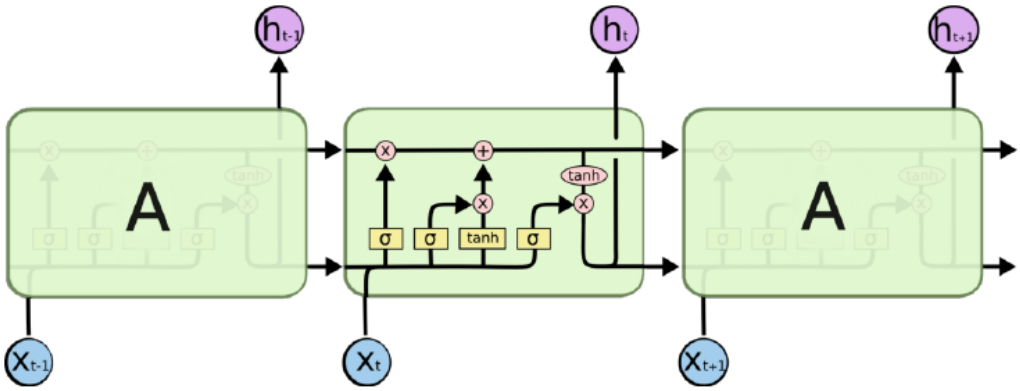
   门控 RNN 能够学会何时清除信息，而不需要手动决定。

4.3.1 LSTM

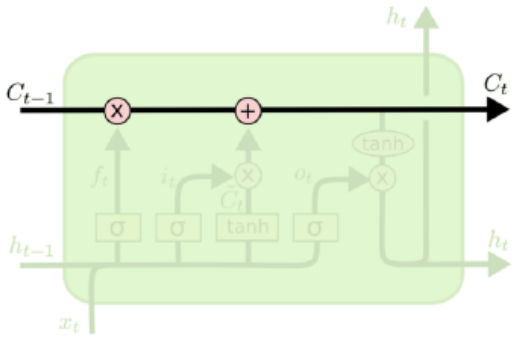
1. LSTM 在手写识别、语音识别、机器翻译、为图像生成标题等领域获得重大成功。
2. LSTM 循环网络除了外部的 RNN 循环之外，还有内部的 LSTM cell 循环（自环）。LSTM 的 cell 代替了普通 RNN 的隐单元，而 LSTM 的 $\vec{\mathbf{h}}^{(t)}$ 是 cell 的一个输出。

   LSTM 引入 cell 循环以保持梯度长时间持续流动。其中一个关键是：cell 循环的权重视上下文而定，而不是固定的。

   具体做法是：通过 gate 来控制这个 cell 循环的权重，而这个 gate 由上下文决定。
   - 注意：cell 输出是 $\vec{\mathbf{h}}^{(t)}$，而不是整个 RNN 单元的输出 $\vec{\mathbf{o}}^{(t)}$。
   - cell 之间的连接是通过 $\vec{\mathbf{h}}^{(t)}, \vec{\mathbf{C}}^{(t)}$ 来连接的。



3. `LSTM` 最重要的就是`cell` 状态 $\vec{\mathbf{C}}^{(t)}$ ，它以水平线在图上方贯穿运行。



4. `sigmoid` 函数 (σ) 的输出在 `0` 到 `1` 之间，描述每个部分有多少量可以通过。它起到门`gate` 的作用：`0` 表示不允许通过，`1` 表示允许全部通过，`0~1` 之间表示部分通过。

   `LSTM` 拥有三个门：遗忘门、输入门、输出门。

5. 遗忘门：控制了`cell` 上一个状态 $\vec{\mathbf{C}}^{(t-1)}$ 中，有多少信息进入当前状态 $\vec{\mathbf{C}}^{(t)}$ 。

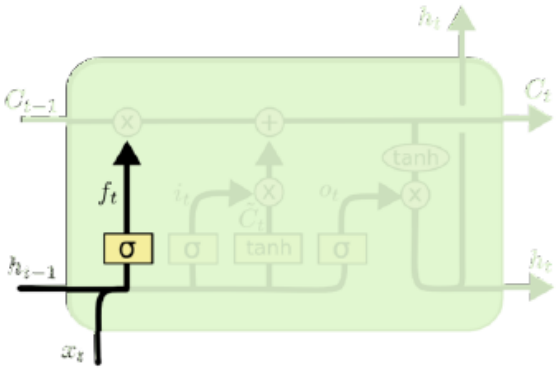
   与渗漏单元类似，`LSTM cell` 也有线性自环。遗忘门 $f_i^{(t)}$ 控制了自环的权重，而不再是常数：

   $$f_i^{(t)} = \sigma(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)})$$

   写成向量形式为：(σ 为逐元素的`sigmoid` 函数)

   $$\vec{\mathbf{f}}^{(t)} = \sigma(\vec{\mathbf{b}}^f + \mathbf{U}^f \vec{\mathbf{x}}^{(t)} + \mathbf{W}^f \vec{\mathbf{h}}^{(t-1)})$$

   其中：$\vec{\mathbf{b}}^f$ 为遗忘门的偏置，$\mathbf{U}^f$ 为遗忘门的输入权重，$\mathbf{W}^f$ 为遗忘门的循环权重。



6. 输入门：控制了输入 $\vec{\mathbf{x}}^{(t)}$ 中，有多少信息进入`cell` 当前状态 $\vec{\mathbf{C}}^{(t)}$ 。

   输入门 $g_i^{(t)}$ 的方程：

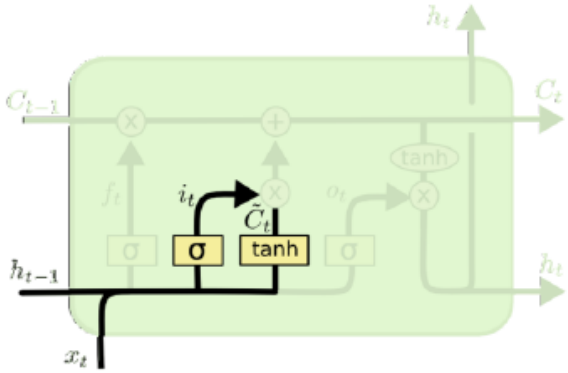
   $$g_i^{(t)} = \sigma(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)})$$

   写成向量的形式为：(σ 为逐元素的`sigmoid` 函数)

$$\vec{\mathbf{g}}^{(t)}=\sigma(\vec{\mathbf{b}}^{g}+\mathbf{U}^{g}\vec{\mathbf{x}}^{(t)}+\mathbf{W}^{g}\vec{\mathbf{h}}^{(t-1)})$$

其中：$\vec{\mathbf{b}}^{g}$ 为输入门的偏置，$\mathbf{U}^{g}$ 为输入门的输入权重，$\mathbf{W}^{g}$ 为输入门的循环权重。

> 图中的 i_t 就是 $\vec{\mathbf{g}}^{(t)}$



7. 输出门：控制了`cell` 状态 $\vec{\mathbf{C}}^{(t)}$ 中，有多少会进入`cell` 的输出 $\vec{\mathbf{h}}^{(t)}$。

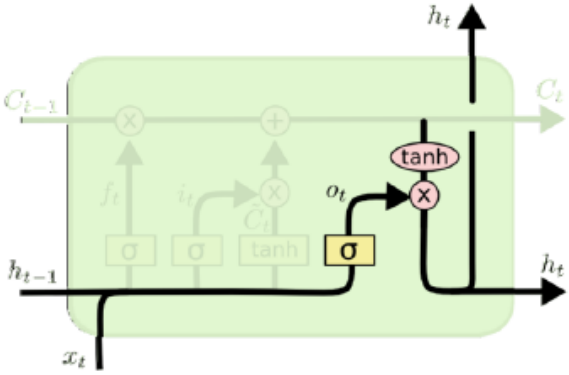
   输出门 $q_i^{(t)}$ 的更新方程：

$$q_i^{(t)}=\sigma(b_i^{o}+\sum_j U_{i,j}^{o}x_j^{(t)}+\sum_j W_{i,j}^{o}h_j^{(t-1)})$$

   写成向量的形式：（σ 为逐元素的`sigmoid` 函数）

$$\vec{\mathbf{q}}^{(t)}=\sigma(\vec{\mathbf{b}}^{o}+\mathbf{U}^{o}\vec{\mathbf{x}}^{(t)}+\mathbf{W}^{o}\vec{\mathbf{h}}^{(t-1)})$$

   其中：$\vec{\mathbf{b}}^{o}$ 为输出门的偏置，$\mathbf{U}^{o}$ 为输出门的输入权重，$\mathbf{W}^{o}$ 为输出门的循环权重。



8. `cell` 状态更新：`cell` 状态 $\vec{\mathbf{C}}^{(t)}$ 由两部分组成：

   - 一部分来自于上一次的状态 $\vec{\mathbf{C}}^{(t-1)}$ ：它经过了遗忘门 $\vec{\mathbf{f}}^{(t)}$ 的控制，使得只有部分状态进入下一次。
   - 一部分来自于输入（包括 $\vec{\mathbf{x}}^{(t)},\vec{\mathbf{h}}^{(t-1)}$ ）：输入需要经过 tanh 非线性层变换之后，然后经过输入门 $\vec{\mathbf{g}}^{(t)}$ 的控制，使得只有部分输入能进入状态更新。

   因此`cell` 状态更新方程为：

$$C_i^{(t)}=f_i^{(t)}C_i^{(t-1)}+g_i^{(t)}\tanh\left(b_i+\sum_j U_{i,j}x_j^{(t)}+\sum_j W_{i,j}h_j^{(t-1)}\right)$$

   写成向量的形式为：（tanh 为逐元素的函数，$\odot$ 为逐元素的向量乘积）

$$\vec{\mathbf{C}}^{(t)}=\vec{\mathbf{f}}^{(t)}\odot\vec{\mathbf{C}}^{(t-1)}+\vec{\mathbf{g}}^{(t)}\odot\tanh(\vec{\mathbf{b}}+\mathbf{U}\vec{\mathbf{x}}^{(t)}+\mathbf{W}\vec{\mathbf{h}}^{(t-1)})$$

   其中：$\vec{\mathbf{b}}$ 为`cell` 的偏置，$\mathbf{U}$ 为`cell` 的输入权重，$\mathbf{W}$ 为`cell` 的循环权重。

9. `cell` 输出更新：`cell` 输出就是 $\vec{\mathbf{h}}^{(t)}$，它是将`cell` 状态经过了 tanh 非线性层之后，再通过输出门 $\vec{\mathbf{q}}^{(t)}$ 控制输出的流量。

$$h_i^{(t)} = \tanh(C_i^{(t)}) q_i^{(t)}$$

写成向量的形式： ($\tanh$ 为逐元素的函数， $\odot$ 为逐元素的向量乘积)

$$\vec{\mathbf{h}}^{(t)} = \tanh(\vec{\mathbf{C}}^{(t)}) \odot \vec{\mathbf{q}}^{(t)}$$

10. 一旦得到了 `cell` 的输出 $\vec{\mathbf{h}}^{(t)}$，则获取整个 `RNN` 单元的输出 $\vec{\mathbf{o}}$ 就和普通的 `RNN` 相同。

$$\begin{aligned}
\text{forget gate:}\quad & \vec{\mathbf{f}}^{(t)} = \sigma(\vec{\mathbf{b}}^{f} + \mathbf{U}^{f}\vec{\mathbf{x}}^{(t)} + \mathbf{W}^{f}\vec{\mathbf{h}}^{(t-1)}) \\
\text{input gate:}\quad & \vec{\mathbf{g}}^{(t)} = \sigma(\vec{\mathbf{b}}^{g} + \mathbf{U}^{g}\vec{\mathbf{x}}^{(t)} + \mathbf{W}^{g}\vec{\mathbf{h}}^{(t-1)}) \\
\text{output gate:}\quad & \vec{\mathbf{q}}^{(t)} = \sigma(\vec{\mathbf{b}}^{o} + \mathbf{U}^{o}\vec{\mathbf{x}}^{(t)} + \mathbf{W}^{o}\vec{\mathbf{h}}^{(t-1)}) \\
\text{cell state:}\quad & \vec{\mathbf{C}}^{(t)} = \vec{\mathbf{f}}^{(t)} \odot \vec{\mathbf{C}}^{(t-1)} + \vec{\mathbf{g}}^{(t)} \odot \tanh(\vec{\mathbf{b}} + \mathbf{U}\vec{\mathbf{x}}^{(t)} + \mathbf{W}\vec{\mathbf{h}}^{(t-1)}) \\
\text{cell output:}\quad & \vec{\mathbf{h}}^{(t)} = \tanh(\vec{\mathbf{C}}^{(t)}) \odot \vec{\mathbf{q}}^{(t)} \\
& \vec{\mathbf{o}}^{(t)} = \text{softmax}\left(\vec{\mathbf{c}} + \mathbf{V}\vec{\mathbf{h}}^{(t)}\right) \\
& L = -\sum_{t=1}^{\tau}\sum_{k=1}^{K} \mathbb{I}_{(k=y^{(t)})} \log o_k^{(t)}
\end{aligned}$$

令节点 $\vec{\mathbf{s}}^{(t)} = \vec{\mathbf{c}} + \mathbf{V}\vec{\mathbf{h}}^{(t)}$，根据激活函数的性质： $\frac{d}{dx}\sigma(x) = \sigma(x)(1-\sigma(x))$，$\frac{d}{dx}\tanh(x) = 1 - \tanh(x)^2$，则有：

- 考虑到 $\vec{\mathbf{h}}^{(t)}$ 的后续节点为：当 $\vec{\mathbf{h}}^{(t)}$ 为最后一个节点时，后续节点为 $\vec{\mathbf{s}}^{(t)}$； 当 $\vec{\mathbf{h}}^{(t)}$ 不是最后一个节点时，后续节点为 $\vec{\mathbf{s}}^{(t)}, \vec{\mathbf{f}}^{(t+1)}, \vec{\mathbf{g}}^{(t+1)}, \vec{\mathbf{q}}^{(t+1)}, \vec{\mathbf{C}}^{(t+1)}$。因此有：

$$\nabla_{\vec{\mathbf{h}}^{(t)}} L = \begin{cases}
\left(\frac{\partial \vec{\mathbf{s}}^{(t)}}{\partial \vec{\mathbf{h}}^{(t)}}\right)^{T} \nabla_{\vec{\mathbf{s}}^{(t)}} L = \mathbf{V}^{T} \nabla_{\vec{\mathbf{s}}^{(t)}} L, & t = \tau \\
\mathbf{V}^{T} \nabla_{\vec{\mathbf{s}}^{(t)}} L + \vec{\mathbf{f}}^{(t+1)} \odot (1 - \vec{\mathbf{f}}^{(t+1)}) \odot (\mathbf{W}^{f})^{T} \nabla_{\vec{\mathbf{f}}^{(t+1)}} L \\
+\vec{\mathbf{g}}^{(t+1)} \odot (1 - \vec{\mathbf{g}}^{(t+1)}) \odot (\mathbf{W}^{g})^{T} \nabla_{\vec{\mathbf{g}}^{(t+1)}} L \\
+\vec{\mathbf{q}}^{(t+1)} \odot (1 - \vec{\mathbf{q}}^{(t+1)}) \odot (\mathbf{W}^{o})^{T} \nabla_{\vec{\mathbf{q}}^{(t+1)}} L \\
+\vec{\mathbf{g}}^{(t+1)} \odot \left(1 - \tanh^2(\vec{\mathbf{b}} + \mathbf{U}\vec{\mathbf{x}}^{(t+1)} + \mathbf{W}\vec{\mathbf{h}}^{(t)})\right) \mathbf{W}^{T} \nabla_{\vec{\mathbf{C}}^{(t+1)}} L, & t < \tau
\end{cases}$$

考虑到：

$$\begin{aligned}
\nabla_{\vec{\mathbf{f}}^{(t)}} L &= \vec{\mathbf{C}}^{(t-1)} \odot \nabla_{\vec{\mathbf{C}}^{(t)}} L \\
\nabla_{\vec{\mathbf{g}}^{(t)}} L &= \tanh(\vec{\mathbf{b}} + \mathbf{U}\vec{\mathbf{x}}^{(t)} + \mathbf{W}\vec{\mathbf{h}}^{(t-1)}) \odot \nabla_{\vec{\mathbf{C}}^{(t)}} L \\
\nabla_{\vec{\mathbf{q}}^{(t)}} L &= \tanh(\vec{\mathbf{C}}^{(t)}) \odot \nabla_{\vec{\mathbf{h}}^{(t)}} L
\end{aligned}$$

因此有：

$$\nabla_{\vec{\mathbf{h}}^{(t)}} L = \begin{cases}
\left(\frac{\partial \vec{\mathbf{s}}^{(t)}}{\partial \vec{\mathbf{h}}^{(t)}}\right)^{T} \nabla_{\vec{\mathbf{s}}^{(t)}} L = \mathbf{V}^{T} \nabla_{\vec{\mathbf{s}}^{(t)}} L, & t = \tau \\
\mathbf{V}^{T} \nabla_{\vec{\mathbf{s}}^{(t)}} L + \vec{\mathbf{f}}^{(t+1)} \odot (1 - \vec{\mathbf{f}}^{(t+1)}) \odot (\mathbf{W}^{f})^{T} \vec{\mathbf{C}}^{(t)} \odot \nabla_{\vec{\mathbf{C}}^{(t+1)}} L \\
+\vec{\mathbf{g}}^{(t+1)} \odot (1 - \vec{\mathbf{g}}^{(t+1)}) \odot (\mathbf{W}^{g})^{T} \tanh(\vec{\mathbf{b}} + \mathbf{U}\vec{\mathbf{x}}^{(t+1)} + \mathbf{W}\vec{\mathbf{h}}^{(t)}) \odot \nabla_{\vec{\mathbf{C}}^{(t+1)}} L \\
+\vec{\mathbf{q}}^{(t+1)} \odot (1 - \vec{\mathbf{q}}^{(t+1)}) \odot (\mathbf{W}^{o})^{T} \tanh(\vec{\mathbf{C}}^{(t+1)}) \odot \nabla_{\vec{\mathbf{h}}^{(t+1)}} L \\
+\vec{\mathbf{g}}^{(t+1)} \odot \left(1 - \tanh^2(\vec{\mathbf{b}} + \mathbf{U}\vec{\mathbf{x}}^{(t+1)} + \mathbf{W}\vec{\mathbf{h}}^{(t)})\right) \mathbf{W}^{T} \nabla_{\vec{\mathbf{C}}^{(t+1)}} L, & t < \tau
\end{cases}$$

  - 由于 $\nabla_{\vec{\mathbf{h}}^{(t)}} L$ 中存在常量部分 $\mathbf{V}^{T} \nabla_{\vec{\mathbf{s}}^{(\tau)}} L$，因此 `LSTM` 可以缓解梯度消失。
  - 由于各种门的存在，因此 $\nabla_{\vec{\mathbf{h}}^{(t)}} L$ 中的非常量部分会被缩小，因此可以缓解梯度爆炸。
- 考虑到 $\vec{\mathbf{C}}^{(t)}$ 的后续节点为：当 $\vec{\mathbf{C}}^{(t)}$ 为最后一个节点时，后续节点为 $\vec{\mathbf{h}}^{(t)}$； 当 $\vec{\mathbf{C}}^{(t)}$ 不是最后一个节点时，后续节点为 $\vec{\mathbf{h}}^{(t)}, \vec{\mathbf{C}}^{(t+1)}$。因此有：

$$\nabla_{\vec{\mathbf{C}}^{(t)}} L=\begin{cases}\left(1-\tanh^2(\vec{\mathbf{C}}^{(t)})\right)\odot\vec{\mathbf{q}}^{(t)}\odot\nabla_{\vec{\mathbf{h}}^{(t)}}L, & t=\tau\\ \left(1-\tanh^2(\vec{\mathbf{C}}^{(t)})\right)\odot\vec{\mathbf{q}}^{(t)}\odot\nabla_{\vec{\mathbf{h}}^{t)}}L+\vec{\mathbf{f}}^{(t+1)}\odot\nabla_{\vec{\mathbf{C}}^{(t+1)}}L, & t<\tau\end{cases}$$

- 考虑到 $\mathbf{V},\vec{\mathbf{c}}$ 对于每个输出 $\vec{\mathbf{o}}^{(1)},\cdots,\vec{\mathbf{o}}^{(\tau)}$ 都有贡献，则有：

$$\nabla_{\vec{\mathbf{c}}}L=\sum_{t=1}^{t=\tau}\left(\frac{\partial\vec{\mathbf{s}}^{(t)}}{\partial\vec{\mathbf{c}}^{(t)}}\right)^T\nabla_{\vec{\mathbf{s}}^{(t)}}L=\sum_{t=1}^{t=\tau}\nabla_{\vec{\mathbf{s}}^{(t)}}L$$

$$\nabla_{V_{i,:}}L=\sum_{t=1}^{t=\tau}\left(\frac{\partial L}{\partial s_i^{(t)}}\right)\nabla_{V_{i,:}^{(t)}}s_i^{(t)}=\sum_{t=1}^{t=\tau}(\nabla_{\vec{\mathbf{s}}^{(t)}}L)_i\vec{\mathbf{h}}^{(t)}$$

  其中 $(\nabla_{\vec{\mathbf{s}}^{(t)}}L)_i$ 表示 $\nabla_{\vec{\mathbf{s}}^{(t)}}L$ 的第 i 个分量。

- 考虑到 $\mathbf{U},\mathbf{W},\vec{\mathbf{b}}$ 对于每个状态 $\vec{\mathbf{C}}^{(1)},\cdots,\vec{\mathbf{C}}^{(\tau)}$ 都有贡献，则有：

$$\nabla_{\vec{\mathbf{b}}}L=\sum_{t=1}^{\tau}\vec{\mathbf{g}}^{(t)}\odot\left(1-\tanh^2(\vec{\mathbf{b}}+\mathbf{U}\vec{\mathbf{x}}^{(t)}+\mathbf{W}\vec{\mathbf{h}}^{(t-1)})\right)\odot\nabla_{\vec{\mathbf{C}}^{(t)}}L$$

$$\nabla_{U_{i,:}}L=\sum_{t=1}^{t=\tau}(\nabla_{\vec{\mathbf{C}}^{(t)}}L)_i\vec{\mathbf{g}}^{(t)}\odot\left(1-\tanh^2(\vec{\mathbf{b}}+\mathbf{U}\vec{\mathbf{x}}^{(t)}+\mathbf{W}\vec{\mathbf{h}}^{(t-1)})\right)\odot\vec{\mathbf{x}}^{(t)}$$

$$\nabla_{W_{i,:}}L=\sum_{t=1}^{t=\tau}(\nabla_{\vec{\mathbf{C}}^{(t)}}L)_i\vec{\mathbf{g}}^{(t)}\odot\left(1-\tanh^2(\vec{\mathbf{b}}+\mathbf{U}\vec{\mathbf{x}}^{(t)}+\mathbf{W}\vec{\mathbf{h}}^{(t-1)})\right)\odot\vec{\mathbf{h}}^{(t-1)}$$

  其中 $(\nabla_{\vec{\mathbf{C}}^{(t)}}L)_i$ 表示 $\nabla_{\vec{\mathbf{C}}^{(t)}}L$ 的第 i 个分量。

- 考虑到 $\mathbf{U}^f,\mathbf{W}^f,\vec{\mathbf{b}}^f$ 对于每个遗忘门 $\vec{\mathbf{f}}^{(1)},\cdots,\vec{\mathbf{f}}^{(\tau)}$ 都有贡献，则有：

$$\nabla_{\vec{\mathbf{b}}^{\mathrm{f}}}L=\sum_{t=1}^{\tau}\vec{\mathbf{f}}^{(t)}\odot(1-\vec{\mathbf{f}}^{(t)})\odot\nabla_{\vec{\mathbf{f}}^{(t)}}L$$

$$\nabla_{U^f_{i,:}}L=\sum_{t=1}^{t=\tau}(\nabla_{\vec{\mathbf{f}}^{(t)}}L)_i\vec{\mathbf{f}}^{(t)}\odot(1-\vec{\mathbf{f}}^{(t)})\odot\vec{\mathbf{x}}^{(t)}$$

$$\nabla_{W^f_{i,:}}L=\sum_{t=1}^{t=\tau}(\nabla_{\vec{\mathbf{f}}^{(t)}}L)_i\vec{\mathbf{f}}^{(t)}\odot(1-\vec{\mathbf{f}}^{(t)})\odot\vec{\mathbf{h}}^{(t-1)}$$

  其中 $(\nabla_{\vec{\mathbf{f}}^{(t)}}L)_i$ 表示 $\nabla_{\vec{\mathbf{f}}^{(t)}}L$ 的第 i 个分量。

- 考虑到 $\mathbf{U}^g,\mathbf{W}^g,\vec{\mathbf{b}}^g$ 对于每个输入门 $\vec{\mathbf{g}}^{(1)},\cdots,\vec{\mathbf{g}}^{(\tau)}$ 都有贡献，则有：

$$\nabla_{\vec{\mathbf{b}}^{\mathrm{g}}}L=\sum_{t=1}^{\tau}\vec{\mathbf{g}}^{(t)}\odot(1-\vec{\mathbf{g}}^{(t)})\odot\nabla_{\vec{\mathbf{g}}^{(t)}}L$$

$$\nabla_{U^g_{i,:}}L=\sum_{t=1}^{t=\tau}(\nabla_{\vec{\mathbf{g}}^{(t)}}L)_i\vec{\mathbf{g}}^{(t)}\odot(1-\vec{\mathbf{g}}^{(t)})\odot\vec{\mathbf{x}}^{(t)}$$

$$\nabla_{W^g_{i,:}}L=\sum_{t=1}^{t=\tau}(\nabla_{\vec{\mathbf{g}}^{(t)}}L)_i\vec{\mathbf{g}}^{(t)}\odot(1-\vec{\mathbf{g}}^{(t)})\odot\vec{\mathbf{h}}^{(t-1)}$$

  其中 $(\nabla_{\vec{\mathbf{g}}^{(t)}}L)_i$ 表示 $\nabla_{\vec{\mathbf{g}}^{(t)}}L$ 的第 i 个分量。

- 考虑到 $\mathbf{U}^o,\mathbf{W}^o,\vec{\mathbf{b}}^o$ 对于每个输出门 $\vec{\mathbf{q}}^{(1)},\cdots,\vec{\mathbf{q}}^{(\tau)}$ 都有贡献，则有：

$$\nabla_{\vec{\mathbf{b}}^{\mathrm{o}}}L=\sum_{t=1}^{\tau}\vec{\mathbf{q}}^{(t)}\odot(1-\vec{\mathbf{q}}^{(t)})\odot\nabla_{\vec{\mathbf{q}}^{(t)}}L$$

$$\nabla_{U^o_{i,:}}L=\sum_{t=1}^{t=\tau}(\nabla_{\vec{\mathbf{q}}^{(t)}}L)_i\vec{\mathbf{q}}^{(t)}\odot(1-\vec{\mathbf{q}}^{(t)})\odot\vec{\mathbf{x}}^{(t)}$$

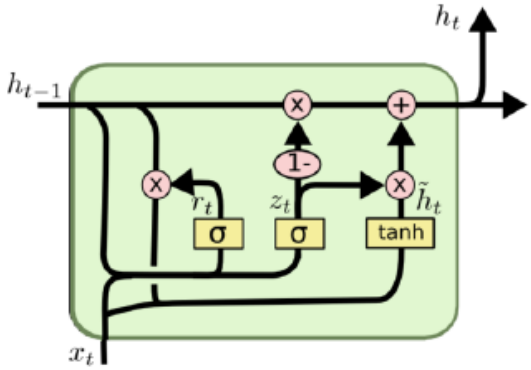
$$\nabla_{W^o_{i,:}}L=\sum_{t=1}^{t=\tau}(\nabla_{\vec{\mathbf{q}}^{(t)}}L)_i\vec{\mathbf{q}}^{(t)}\odot(1-\vec{\mathbf{q}}^{(t)})\odot\vec{\mathbf{h}}^{(t-1)}$$

  其中 $(\nabla_{\vec{\mathbf{q}}^{(t)}}L)_i$ 表示 $\nabla_{\vec{\mathbf{q}}^{(t)}}L$ 的第 i 个分量。

11. 也可以选择使用`cell`状态 $\vec{\mathbf{C}}^{(t)}$ 作为这些门的额外输入。此时 $\vec{\mathbf{f}}^{(t)},\vec{\mathbf{g}}^{(t)},\vec{\mathbf{q}}^{(t)}$ 就多了额外的权重参数，这些参数对应于 $\vec{\mathbf{C}}^{(t-1)}$ 的权重和偏置。

4.3.2 GRU

1. 门控循环单元`GRU`比`LSTM`模型更简单：
   - `GRU`的单个门控单元同时作为遗忘门和输入门，整个`GRU`模型只有两个门：更新门、复位门。
   - `GRU`不再区分`cell`的状态 $\vec{\mathbf{C}}$ 和`cell`的输出 $\vec{\mathbf{h}}$。



2. 更新门：控制了新的信息 $\tilde{\vec{\mathbf{h}}}^{(t)}$（由 $\vec{\mathbf{x}}^{(t)},\vec{\mathbf{h}}^{(t-1)}$ 生成）、旧的信息 $\vec{\mathbf{h}}^{(t-1)}$ 中各有多少信息进入了 $\vec{\mathbf{h}}^{(t)}$。

   更新门 $z_i^{(t)}$ 的更新方程：

$$z_i^{(t)}=\sigma\left(b_i^z+\sum_j U_{i,j}^z x_j^{(t)}+\sum_j W_{i,j}^z h_j^{(t-1)}\right)$$

   写成向量的形式为：（σ 为逐元素的`sigmoid`函数）

$$\vec{\mathbf{z}}^{(t)}=\sigma(\vec{\mathbf{b}}^z+\mathbf{U}^z\vec{\mathbf{x}}^{(t)}+\mathbf{W}^z\vec{\mathbf{h}}^{(t-1)})$$

   其中：$\vec{\mathbf{b}}^z$ 为更新门的偏置，$\mathbf{U}^z$ 为更新门的输入权重，$\mathbf{W}^z$ 为更新门的循环权重。

3. 复位门：控制了新的信息 $\tilde{\vec{\mathbf{h}}}^{(t)}$ 中，$\vec{\mathbf{x}}^{(t)},\vec{\mathbf{h}}^{(t-1)}$ 之间的比例。它表示在新的信息中，旧的信息多大程度上影响新的信息。如果 $r=0$，则旧的信息不影响新的信息，可以理解为复位。

   复位门 $r_i^{(t)}$ 的更新方程：

$$r_i^{(t)}=\sigma\left(b_i^r+\sum_j U_{i,j}^r x_j^{(t)}+\sum_j W_{i,j}^r h_j^{(t-1)}\right)$$

   写成向量的形式为：（σ 为逐元素的`sigmoid`函数）

$$\vec{\mathbf{r}}^{(t)}=\sigma(\vec{\mathbf{b}}^r+\mathbf{U}^r\vec{\mathbf{x}}^{(t)}+\mathbf{W}^r\vec{\mathbf{h}}^{(t-1)})$$

   其中：$\vec{\mathbf{b}}^r$ 为复位门的偏置，$\mathbf{U}^r$ 为复位门的输入权重，$\mathbf{W}^r$ 为复位门的循环权重。

4. `cell`输出：`cell`输出就是 $\vec{\mathbf{h}}^{(t)}$。

   `cell`更新方程：

$$h_i^{(t)}=z_i^{(t)}h_i^{(t-1)}+(1-z_i^{(t)})\tanh\left(b_i+\sum_j U_{i,j}x_j^{(t)}+\sum_j W_{i,j}r_j^{(t)}h_j^{(t-1)}\right)$$

   写成向量的形式：（其中 $\odot$ 为逐元素的向量乘积；$\tanh$ 为逐元素的函数）

$$\vec{\mathbf{h}}^{(t)}=\vec{\mathbf{z}}^{(t)}\odot\vec{\mathbf{h}}^{(t-1)}+(1-\vec{\mathbf{z}}^{(t)})\odot\tanh(\vec{\mathbf{b}}+\mathbf{U}\vec{\mathbf{x}}^{(t)}+\mathbf{W}\vec{\mathbf{r}}^{(t)}\odot\vec{\mathbf{h}}^{(t-1)})$$

令 $\tilde{\vec{\mathbf{h}}}^{(t)}=\tanh(\vec{\mathbf{b}}+\mathbf{U}\vec{\mathbf{x}}^{(t)}+\mathbf{W}\vec{\mathbf{r}}^{(t)}\odot\vec{\mathbf{h}}^{(t-1)})$，它刻画了本次的更新。于是`cell`的输出更新方程为：

$$\vec{\mathbf{h}}^{(t)}=\vec{\mathbf{z}}^{(t)}\odot\vec{\mathbf{h}}^{(t-1)}+(1-\vec{\mathbf{z}}^{(t)})\odot\tilde{\vec{\mathbf{h}}}^{(t)}$$

其中：$\vec{\mathbf{b}}$ 为`cell`的偏置，$\mathbf{U}$ 为`cell`的输入权重，$\mathbf{W}$ 为`cell`的循环权重。

5. 一旦得到了`cell`的输出 $\vec{\mathbf{h}}^{(t)}$，则获取整个`RNN`单元的输出 $\vec{\mathbf{o}}$ 就和普通的`RNN`相同。

$$\begin{aligned}
\text{update gate:}\quad \vec{\mathbf{z}}^{(t)}&=\sigma(\vec{\mathbf{b}}^{z}+\mathbf{U}^{z}\vec{\mathbf{x}}^{(t)}+\mathbf{W}^{z}\vec{\mathbf{h}}^{(t-1)})\\
\text{reset gate:}\quad \vec{\mathbf{r}}^{(t)}&=\sigma(\vec{\mathbf{b}}^{r}+\mathbf{U}^{r}\vec{\mathbf{x}}^{(t)}+\mathbf{W}^{r}\vec{\mathbf{h}}^{(t-1)})\\
\text{cell output:}\quad \vec{\mathbf{h}}^{(t)}&=\vec{\mathbf{z}}^{(t)}\odot\vec{\mathbf{h}}^{(t-1)}+(1-\vec{\mathbf{z}}^{(t)})\odot\tanh(\vec{\mathbf{b}}+\mathbf{U}\vec{\mathbf{x}}^{(t)}+\mathbf{W}\vec{\mathbf{r}}^{(t)}\odot\vec{\mathbf{h}}^{(t-1)})\\
\vec{\mathbf{o}}^{(t)}&=\text{softmax}\left(\vec{\mathbf{c}}+\mathbf{V}\vec{\mathbf{h}}^{(t)}\right)\\
L&=-\sum_{t=1}^{\tau}\sum_{k=1}^{K}\mathbb{I}_{(k=y^{(t)})}\log o_k^{(t)}
\end{aligned}$$

令节点 $\vec{\mathbf{s}}^{(t)}=\vec{\mathbf{c}}+\mathbf{V}\vec{\mathbf{h}}^{(t)}$，根据激活函数的性质：$\frac{d}{dx}\sigma(x)=\sigma(x)(1-\sigma(x))$，$\frac{d}{dx}\tanh(x)=1-\tanh(x)^2$，则有：

- 考虑到 $\vec{\mathbf{h}}^{(t)}$ 的后续节点为：当 $\vec{\mathbf{h}}^{(t)}$ 为最后一个节点时，后续节点为 $\vec{\mathbf{s}}^{(t)}$；当 $\vec{\mathbf{h}}^{(t)}$ 不是最后一个节点时，后续节点为 $\vec{\mathbf{s}}^{(t)},\vec{\mathbf{z}}^{(t+1)},\vec{\mathbf{r}}^{(t+1)},\vec{\mathbf{h}}^{(t+1)}$。记 $\vec{\mathbf{e}}^{(t)}=\vec{\mathbf{b}}+\mathbf{U}\vec{\mathbf{x}}^{(t)}+\mathbf{W}\vec{\mathbf{r}}^{(t)}\odot\vec{\mathbf{h}}^{(t-1)}$，因此有：

$$\nabla_{\vec{\mathbf{h}}^{(t)}}L=\begin{cases}
\left(\frac{\partial\vec{\mathbf{s}}^{(t)}}{\partial\vec{\mathbf{h}}^{(t)}}\right)^{T}\nabla_{\vec{\mathbf{s}}^{(t)}}L=\mathbf{V}^{T}\nabla_{\vec{\mathbf{s}}^{(t)}}L, & t=\tau\\
\mathbf{V}^{T}\nabla_{\vec{\mathbf{s}}^{(t)}}L+\vec{\mathbf{z}}^{(t+1)}\odot(1-\vec{\mathbf{z}}^{(t+1)})\odot(\mathbf{W}^{z})^{T}\nabla_{\vec{\mathbf{z}}^{(t+1)}}L & \\
+\vec{\mathbf{r}}^{(t+1)}\odot(1-\vec{\mathbf{r}}^{(t+1)})\odot(\mathbf{W}^{r})^{T}\nabla_{\vec{\mathbf{r}}^{(t+1)}}L+ & \\
\left(\vec{\mathbf{z}}^{(t+1)}+(1-\vec{\mathbf{z}}^{(t+1)})\odot\left(1-\tanh^{2}(\vec{\mathbf{e}}^{(t+1)})\right)\odot\mathbf{W}\vec{\mathbf{r}}^{(t+1)}\right)\nabla_{\vec{\mathbf{h}}^{(t+1)}}L, & t<\tau
\end{cases}$$

考虑到：

$$\begin{aligned}
\nabla_{\vec{\mathbf{z}}^{(t)}}L&=\left(\vec{\mathbf{h}}^{(t-1)}-\tanh(\vec{\mathbf{e}}^{(t)})\right)\odot\nabla_{\vec{\mathbf{h}}^{(t)}}L\\
\nabla_{\vec{\mathbf{r}}^{(t)}}L&=(1-\vec{\mathbf{z}}^{(t)})\odot\mathbf{W}^{T}\left(1-\tanh^{2}(\vec{\mathbf{e}}^{(t)})\right)\odot\vec{\mathbf{h}}^{(t-1)}\nabla_{\vec{\mathbf{h}}^{(t)}}L
\end{aligned}$$

因此有：

$$\nabla_{\vec{\mathbf{h}}^{(t)}}L=\begin{cases}
\left(\frac{\partial\vec{\mathbf{s}}^{(t)}}{\partial\vec{\mathbf{h}}^{(t)}}\right)^{T}\nabla_{\vec{\mathbf{s}}^{(t)}}L=\mathbf{V}^{T}\nabla_{\vec{\mathbf{s}}^{(t)}}L, & t=\tau\\
\mathbf{V}^{T}\nabla_{\vec{\mathbf{s}}^{(t)}}L+\vec{\mathbf{z}}^{(t+1)}\odot(1-\vec{\mathbf{z}}^{(t+1)})\odot(\mathbf{W}^{z})^{T}\left(\vec{\mathbf{h}}^{(t)}-\tanh(\vec{\mathbf{e}}^{(t+1)})\right)\odot\nabla_{\vec{\mathbf{h}}^{(t+1)}}L & \\
+\vec{\mathbf{r}}^{(t+1)}\odot(1-\vec{\mathbf{r}}^{(t+1)})\odot(\mathbf{W}^{r})^{T}(1-\vec{\mathbf{z}}^{(t+1)}) & \\
\odot\mathbf{W}^{T}\left(1-\tanh^{2}(\vec{\mathbf{e}}^{(t+1)})\right)\odot\vec{\mathbf{h}}^{(t)}\nabla_{\vec{\mathbf{h}}^{(t+1)}}L+ & \\
\left(\vec{\mathbf{z}}^{(t+1)}+(1-\vec{\mathbf{z}}^{(t+1)})\odot\left(1-\tanh^{2}(\vec{\mathbf{e}}^{(t+1)})\right)\odot\mathbf{W}\vec{\mathbf{r}}^{(t+1)}\right)\nabla_{\vec{\mathbf{h}}^{(t+1)}}L, & t<\tau
\end{cases}$$

  - 由于 $\nabla_{\vec{\mathbf{h}}^{(t)}}L$ 中存在常量部分 $\mathbf{V}^{T}\nabla_{\vec{\mathbf{s}}^{(\tau)}}L$，因此`GRU`可以缓解梯度消失。
  - 由于各种门的存在，因此 $\nabla_{\vec{\mathbf{h}}^{(t)}}L$ 中的非常量部分会被缩小，因此可以缓解梯度爆炸。
- 考虑到 $\mathbf{V},\vec{\mathbf{c}}$ 对于每个输出 $\vec{\mathbf{o}}^{(1)},\cdots,\vec{\mathbf{o}}^{(\tau)}$ 都有贡献，则有：

$$\begin{aligned}
\nabla_{\vec{\mathbf{c}}}L&=\sum_{t=1}^{t=\tau}\left(\frac{\partial\vec{\mathbf{s}}^{(t)}}{\partial\vec{\mathbf{c}}^{(t)}}\right)^{T}\nabla_{\vec{\mathbf{s}}^{(t)}}L=\sum_{t=1}^{t=\tau}\nabla_{\vec{\mathbf{s}}^{(t)}}L\\
\nabla_{V_{i,:}}L&=\sum_{t=1}^{t=\tau}\left(\frac{\partial L}{\partial s_i^{(t)}}\right)\nabla_{V_{i,:}^{(t)}}s_i^{(t)}=\sum_{t=1}^{t=\tau}(\nabla_{\vec{\mathbf{s}}^{(t)}}L)_i\vec{\mathbf{h}}^{(t)}
\end{aligned}$$

其中 $(\nabla_{\vec{\mathbf{s}}^{(t)}}L)_i$ 表示 $\nabla_{\vec{\mathbf{s}}^{(t)}}L$ 的第 i 个分量。

- 考虑到 $\mathbf{U},\mathbf{W},\vec{\mathbf{b}}$ 对于每个状态 $\vec{\mathbf{h}}^{(1)},\cdots,\vec{\mathbf{h}}^{(\tau)}$ 都有贡献，则有：

$$\nabla_{\vec{\mathbf{b}}}L=\sum_{t=1}^{\tau}(1-\vec{\mathbf{z}}^{(t)})\odot\left(1-\tanh^2(\vec{\mathbf{b}}+\mathbf{U}\vec{\mathbf{x}}^{(t)}+\mathbf{W}\vec{\mathbf{r}}^{(t)}\odot\vec{\mathbf{h}}^{(t-1)})\right)\odot\nabla_{\vec{\mathbf{h}}^{(t)}}L$$

$$\nabla_{U_{i,:}}L=\sum_{t=1}^{t=\tau}(\nabla_{\vec{\mathbf{h}}^{(t)}}L)_i(1-\vec{\mathbf{z}}^{(t)})\odot\left(1-\tanh^2(\vec{\mathbf{b}}+\mathbf{U}\vec{\mathbf{x}}^{(t)}+\mathbf{W}\vec{\mathbf{r}}^{(t)}\odot\vec{\mathbf{h}}^{(t-1)})\right)\odot\vec{\mathbf{x}}^{(t)}$$

$$\nabla_{W_{i,:}}L=\sum_{t=1}^{t=\tau}(\nabla_{\vec{\mathbf{h}}^{(t)}}L)_i(1-\vec{\mathbf{z}}^{(t)})\odot\left(1-\tanh^2(\vec{\mathbf{b}}+\mathbf{U}\vec{\mathbf{x}}^{(t)}+\mathbf{W}\vec{\mathbf{r}}^{(t)}\odot\vec{\mathbf{h}}^{(t-1)})\right)\odot\vec{\mathbf{r}}^{(t)}\odot\vec{\mathbf{h}}^{(t-1)}$$

其中 $(\nabla_{\vec{\mathbf{h}}^{(t)}}L)_i$ 表示 $\nabla_{\vec{\mathbf{h}}^{(t)}}L$ 的第 i 个分量。

- 考虑到 $\mathbf{U}^z,\mathbf{W}^z,\vec{\mathbf{b}}^z$ 对于每个复位门 $\vec{\mathbf{z}}^{(1)},\cdots,\vec{\mathbf{z}}^{(\tau)}$ 都有贡献，则有：

$$\nabla_{\vec{\mathbf{b}}^z}L=\sum_{t=1}^{\tau}\vec{\mathbf{z}}^{(t)}\odot(1-\vec{\mathbf{z}}^{(t)})\odot\nabla_{\vec{\mathbf{z}}^{(t)}}L$$

$$\nabla_{U^z_{i,:}}L=\sum_{t=1}^{t=\tau}(\nabla_{\vec{\mathbf{z}}^{(t)}}L)_i\vec{\mathbf{z}}^{(t)}\odot(1-\vec{\mathbf{z}}^{(t)})\odot\vec{\mathbf{x}}^{(t)}$$

$$\nabla_{W^z_{i,:}}L=\sum_{t=1}^{t=\tau}(\nabla_{\vec{\mathbf{z}}^{(t)}}L)_i\vec{\mathbf{z}}^{(t)}\odot(1-\vec{\mathbf{z}}^{(t)})\odot\vec{\mathbf{h}}^{(t-1)}$$

其中 $(\nabla_{\vec{\mathbf{z}}^{(t)}}L)_i$ 表示 $\nabla_{\vec{\mathbf{z}}^{(t)}}L$ 的第 i 个分量。

- 考虑到 $\mathbf{U}^r,\mathbf{W}^r,\vec{\mathbf{r}}^r$ 对于每个更新门 $\vec{\mathbf{r}}^{(1)},\cdots,\vec{\mathbf{r}}^{(\tau)}$ 都有贡献，则有：

$$\nabla_{\vec{\mathbf{b}}^r}L=\sum_{t=1}^{\tau}\vec{\mathbf{r}}^{(t)}\odot(1-\vec{\mathbf{r}}^{(t)})\odot\nabla_{\vec{\mathbf{r}}^{(t)}}L$$

$$\nabla_{U^r_{i,:}}L=\sum_{t=1}^{t=\tau}(\nabla_{\vec{\mathbf{r}}^{(t)}}L)_i\vec{\mathbf{r}}^{(t)}\odot(1-\vec{\mathbf{r}}^{(t)})\odot\vec{\mathbf{x}}^{(t)}$$

$$\nabla_{W^r_{i,:}}L=\sum_{t=1}^{t=\tau}(\nabla_{\vec{\mathbf{r}}^{(t)}}L)_i\vec{\mathbf{r}}^{(t)}\odot(1-\vec{\mathbf{r}}^{(t)})\odot\vec{\mathbf{h}}^{(t-1)}$$

其中 $(\nabla_{\vec{\mathbf{r}}^{(t)}}L)_i$ 表示 $\nabla_{\vec{\mathbf{r}}^{(t)}}L$ 的第 i 个分量。

4.3.3 讨论

1. 在`LSTM`与`GRU`中有两种非线性函数：`sigmoid`与`tanh`。
   - `sigmoid`用于各种门，是因为它的阈值为`0~1`，可以很好的模拟开关的关闭程度。
   - `tanh`用于激活函数，是因为它的阈值为 -1~1，它的梯度的阈值为 0~1。
     - 如果使用`sigmoid`作为激活函数，则其梯度范围为 0~0.5，容易发生梯度消失。
     - 如果使用`relu`作为激活函数，则前向传播时，信息容易爆炸性增长。

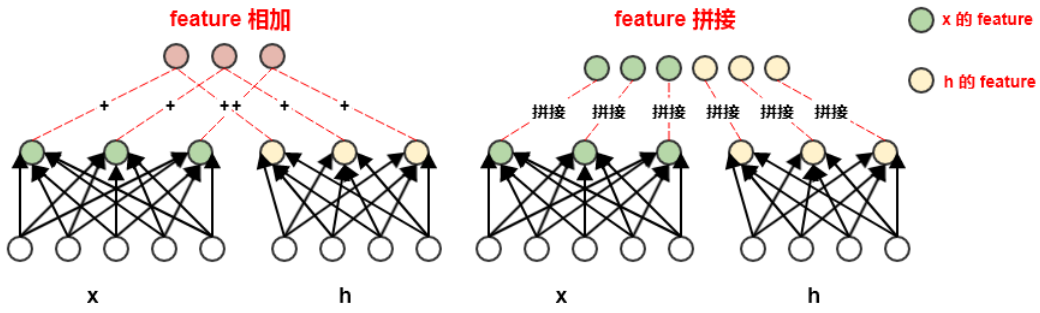
       另外`relu`激活函数也会使得输出只有大于等于`0`的部分。
2. 前面给出的`LSTM`和`GRU`中，$\vec{\mathbf{x}}^{(t)},\vec{\mathbf{h}}^{(t-1)}$ 是通过`feature map`直接相加，如`LSTM`中的状态更新方程：

$$\vec{\mathbf{C}}^{(t)}=\vec{\mathbf{f}}^{(t)}\odot\vec{\mathbf{C}}^{(t-1)}+\vec{\mathbf{g}}^{(t)}\odot\tanh(\vec{\mathbf{b}}+\mathbf{U}\vec{\mathbf{x}}^{(t)}+\mathbf{W}\vec{\mathbf{h}}^{(t-1)})$$

事实上，也可以通过`feature map`进行拼接，如：

$$\vec{\mathbf{C}}^{(t)}=\vec{\mathbf{f}}^{(t)}\odot\vec{\mathbf{C}}^{(t-1)}+\vec{\mathbf{g}}^{(t)}\odot\tanh(\vec{\mathbf{b}}+\mathbf{U}\vec{\mathbf{x}}^{(t)}:\mathbf{W}\vec{\mathbf{h}}^{(t-1)})$$

其中：表示将两个向量进行拼接。



4.4 编码-解码架构

1. 前面介绍的多长度输入序列的模式中，输出序列和输入序列长度相同。实际任务中，如：语音识别、机器翻译、知识问答等任务，输出序列和输入序列长度不相等。

   `编码-解码` 架构就是为了解决这类问题。设输入序列为 $\{\vec{\mathbf{x}}^{(1)},\vec{\mathbf{x}}^{(2)},\cdots,\vec{\mathbf{x}}^{(\tau_x)}\}$，输出序列为 $\{\vec{\mathbf{y}}^{(1)},\vec{\mathbf{y}}^{(2)},\cdots,\vec{\mathbf{y}}^{(\tau_y)}\}$。长度 $\tau_x\neq\tau_y$。

   设 `C` 为输入的一个表达 `representation`，包含了输入序列的有效信息。

   - 它可能是一个向量，也可能是一个固定长度的向量序列。
   - 如果 `C` 是一个向量序列，则它和输入序列的区别在于：序列 `C` 是定长的、较短的；而输入序列是不定长的、较长的。

   整个 `编码-解码` 结构分为：编码器，解码器。

   - 编码器（也叫作读取器，或者输入 `RNN`）：处理输入序列。

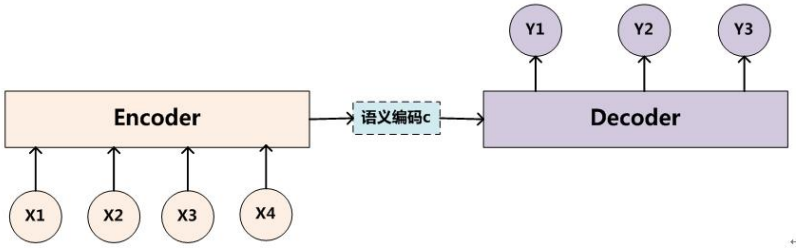
     编码器的最后一个状态 $\vec{\mathbf{h}}^{(\tau_x)}$ 通常就是输入序列的表达 `C`，并且作为解码器的输入向量。

   - 解码器（也叫作写入器，或者输出 `RNN`）：处理输入的表达 `C`。

     解码器有三种处理 `C` 的方式：输入 `C` 作为每个时间步的输入、输入 `C` 作为初始状态 $\vec{\mathbf{h}}^{(0)}$ 且每个时间步没有额外的输入、结合上述两种方式。

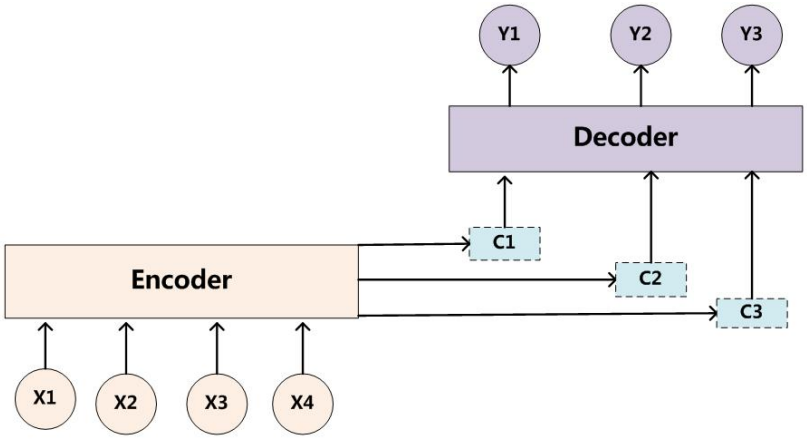
   - 训练时，编码器和解码器并不是单独训练，而是共同训练以最大化：

     $$\log P(\vec{\mathbf{y}}^{(1)},\vec{\mathbf{y}}^{(2)},\cdots,\vec{\mathbf{y}}^{(\tau_y)}\mid\vec{\mathbf{x}}^{(1)},\vec{\mathbf{x}}^{(2)},\cdots,\vec{\mathbf{x}}^{(\tau_x)})$$



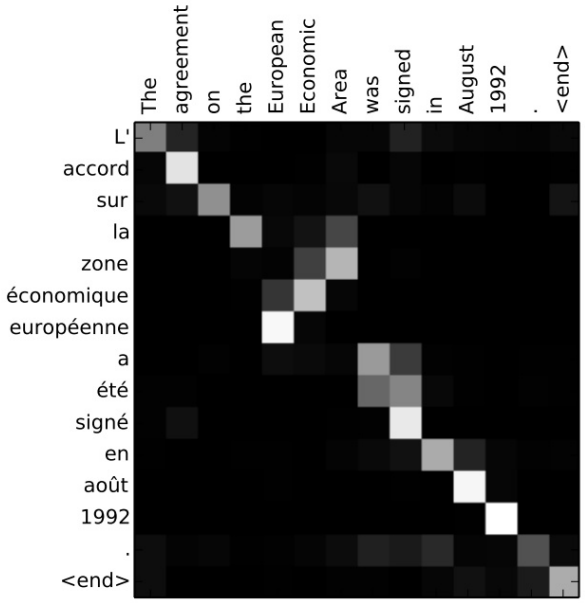
2. 编码-解码架构中：
   - 输入序列长度 τ_x 和输出序列长度 τ_y 可以不同。
   - 对于编码器与解码器隐状态是否具有相同尺寸并没有限制，它们是相互独立设置的。
3. 编码-解码架构的主要缺点：编码器 `RNN` 输出的上下文 `C` 的维度太小，难以恰当的概括一个长的输入序列的完整信息。

   可以通过引入 `attention` 机制来缓解该问题。



4.5 attention

1. `attention` 是一种提升 `encoder - decoder` 模型效果的机制，一般称作 `attention mechanism` 。
    - `attention` 被广泛用于机器翻译、语音识别、图像标注`Image Caption` 等领域。如：机器翻译中，为句子中的每个词赋予不同的权重。
    - `attention` 本身可以理解为一种对齐关系，给出了模型输入、输出之间的对齐关系，解释了模型到底学到了什么知识。
        - 在机器翻译中，解释了输入序列的不同位置对输出序列的影响程度。如下图所示为机器翻译中，输入-输出的 `attention` 矩阵。



        - 在图像标注中，解释了图片不同区域对输出文本序列的影响程度。如下图所示为图像标注中，影响输出单词的图像块。

Figure 5. Examples of mistakes where we can use attention to gain intuition into what the model saw.

2. 设输入序列为 $\{\vec{\mathbf{x}}^{(1)},\vec{\mathbf{x}}^{(2)},\cdots,\vec{\mathbf{x}}^{(\tau_x)}\}$，输出序列为 $\{\vec{\mathbf{y}}^{(1)},\vec{\mathbf{y}}^{(2)},\cdots,\vec{\mathbf{y}}^{(\tau_y)}\}$，长度 $\tau_x\neq\tau_y$。设 `encoder` 的隐向量为 $\vec{\mathbf{h}}_1,\vec{\mathbf{h}}_2,\cdots$，`decoder` 的隐向量为 $\vec{\mathbf{s}}_1,\vec{\mathbf{s}}_2,\cdots$。

   - 对于传统的 `encoder-decoder` 模型，`decoder` 的输入只有一个向量，该向量就是输入序列经过`encoder`编码的上下文向量 $\vec{\mathbf{c}}$。

     通常将 `encoder` 的最后一个隐单元的隐向量 $\vec{\mathbf{h}}_{\tau_x}$ 作为上下文向量。

   - 对于 `attention encoder-decoder` 模型，`decoder` 的输入是一个向量序列，序列长度为 τ_y。

     `decoder`位置 i 的输入是采用了 `attention` 机制的上下文向量 $\vec{\mathbf{c}}_i$，不同位置的上下文向量不同。

     - 上下文向量 $\vec{\mathbf{c}}_i$ 由 `encoder` 的所有隐向量加权得到：$\vec{\mathbf{c}}_i=\sum_{t=1}^{\tau_x}\alpha_{i,t}\vec{\mathbf{h}}_t$。

       其中 $\sum_{t=1}^{\tau_x}\alpha_{i,t}=1,\quad \alpha_{i,t}\geq 0$。

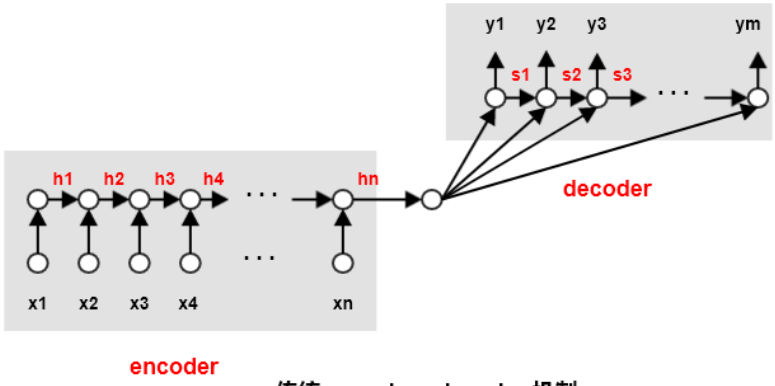
     - 权重 $\alpha_{i,t}$ 刻画了：在对第 i 个输出进行解码时，第 t 个输入的重要程度。

       一个直觉上的方法是：首先计算 $\vec{\mathbf{s}}_{i-1}$ 与 $\vec{\mathbf{h}}_t$ 的相关性，然后对所有的 $t=1,2,\cdots,\tau_x$ 归一化即可得到权重系数。即：

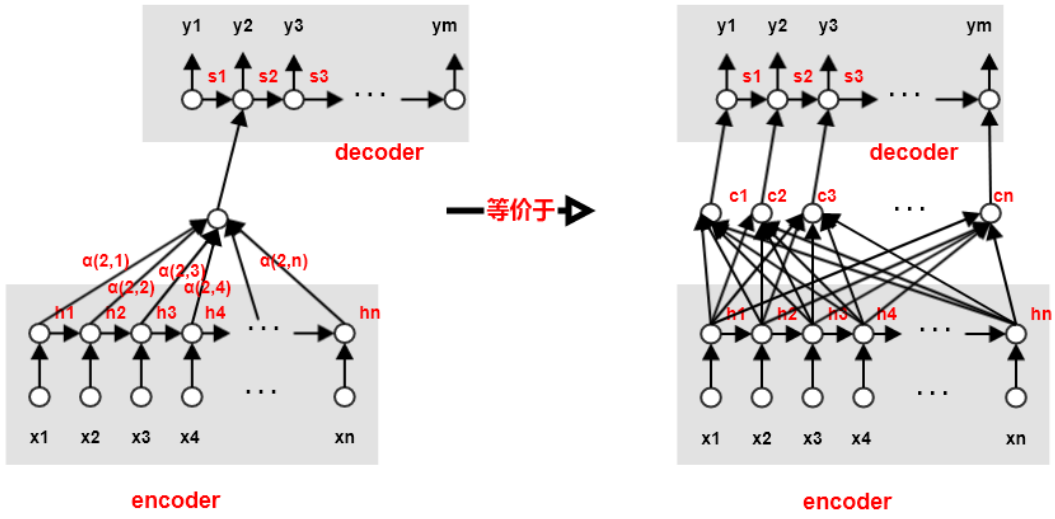
       $$e_{i,t}=\text{score}(\vec{\mathbf{s}}_{i-1},\vec{\mathbf{h}}_t),\quad \alpha_{i,t}=\frac{\exp(e_{i,t})}{\sum_{t'=1}^{\tau_x}\exp(e_{i,t'})},\quad t=1,2,\cdots,\tau_x$$

       其中 `score` 由多种计算方式，不同的计算方式代表不同的 `attention` 模型（$\vec{\mathbf{v}}_\alpha,\mathbf{W}_\alpha$ 为待学习的参数，`n` 为向量的维度）：

       $$\text{score}(\vec{\mathbf{s}}_{i-1},\vec{\mathbf{h}}_t)=\begin{cases}\frac{\vec{\mathbf{s}}_{i-1}\cdot\vec{\mathbf{h}}_t}{||\vec{\mathbf{s}}_{i-1}||\times||\vec{\mathbf{h}}_t||}, & \text{cosin}\\ \vec{\mathbf{s}}_{i-1}\cdot\vec{\mathbf{h}}_t, & \text{dot}\\ \frac{\vec{\mathbf{s}}_{i-1}\cdot\vec{\mathbf{h}}_t}{\sqrt{n}}, & \text{scaled-dot}\\ \vec{\mathbf{s}}_{i-1}^T\mathbf{W}_\alpha\vec{\mathbf{h}}_t, & \text{general}\\ \vec{\mathbf{v}}_\alpha^T\tanh(\mathbf{W}_\alpha[\vec{\mathbf{s}}_{i-1}:\vec{\mathbf{h}}_t]), & \text{concat}\end{cases}$$



传统 encoder - decoder 机制



attention encoder - decoder 机制

4.5.1 local attention

1. 上述的 `attention` 机制中为了计算上下文向量 $\vec{\mathbf{c}}_i$， 需要考虑 `encoder` 的所有隐向量。当输入序列较长时（如一段话或一篇文章），计算效率较低。

   `local attention` 在计算上下文向量 $\vec{\mathbf{c}}_i$ 时只需要考虑 `encoder` 的部分隐向量：首选预测 `encoder` 端对齐的位置 p_i，然后基于位置 p_i 选择一个窗口来计算上下文向量 $\vec{\mathbf{c}}_i$ 。

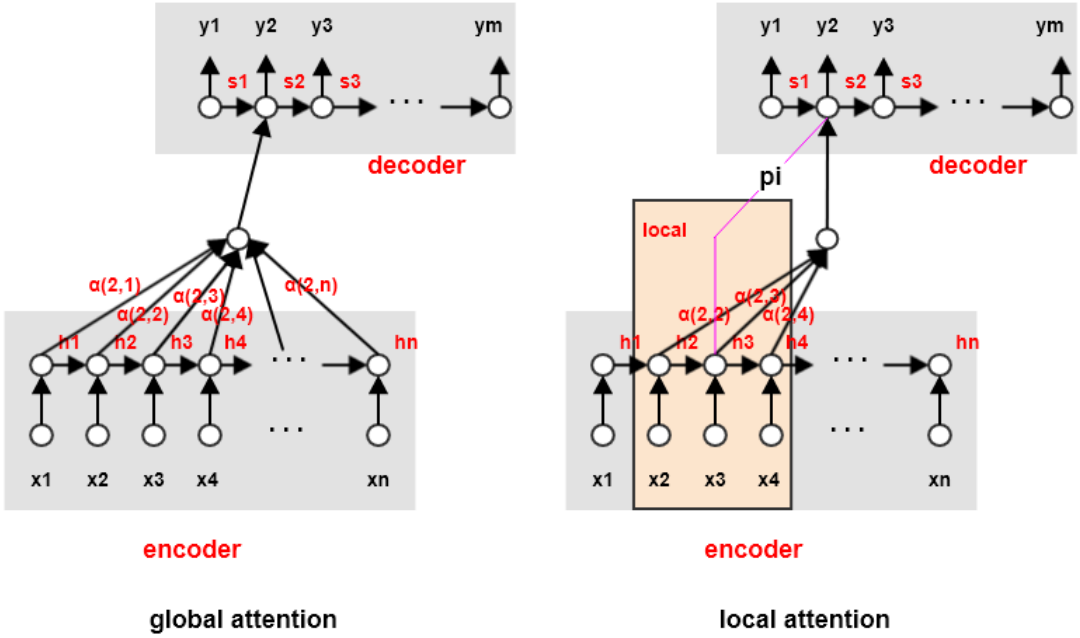
   $$p_i = \tau_x \times \text{sigmoid}\left(\vec{\mathbf{v}}_p \cdot \tanh(\mathbf{W}_p \vec{\mathbf{s}}_{i-1})\right)$$

   $$e_{i,t} = \text{score}(\vec{\mathbf{s}}_{i-1}, \vec{\mathbf{h}}_t) \exp\left(-\frac{(t-p_i)^2}{2d^2}\right)$$

   其中 $\vec{\mathbf{v}}_p, \mathbf{W}_p$ 为待学习的参数， d 为人工指定的固定常量。

   虽然 `local attention` 可以提高计算效率，但是会带来两个问题：

   - 当 `encoder` 的输入序列长度 τ_x 并不是很长时，计算量并没有显著减小。
   - 位置 p_i 的预测并不是非常准确，这就直接影响了计算 `attention` 的准确性。



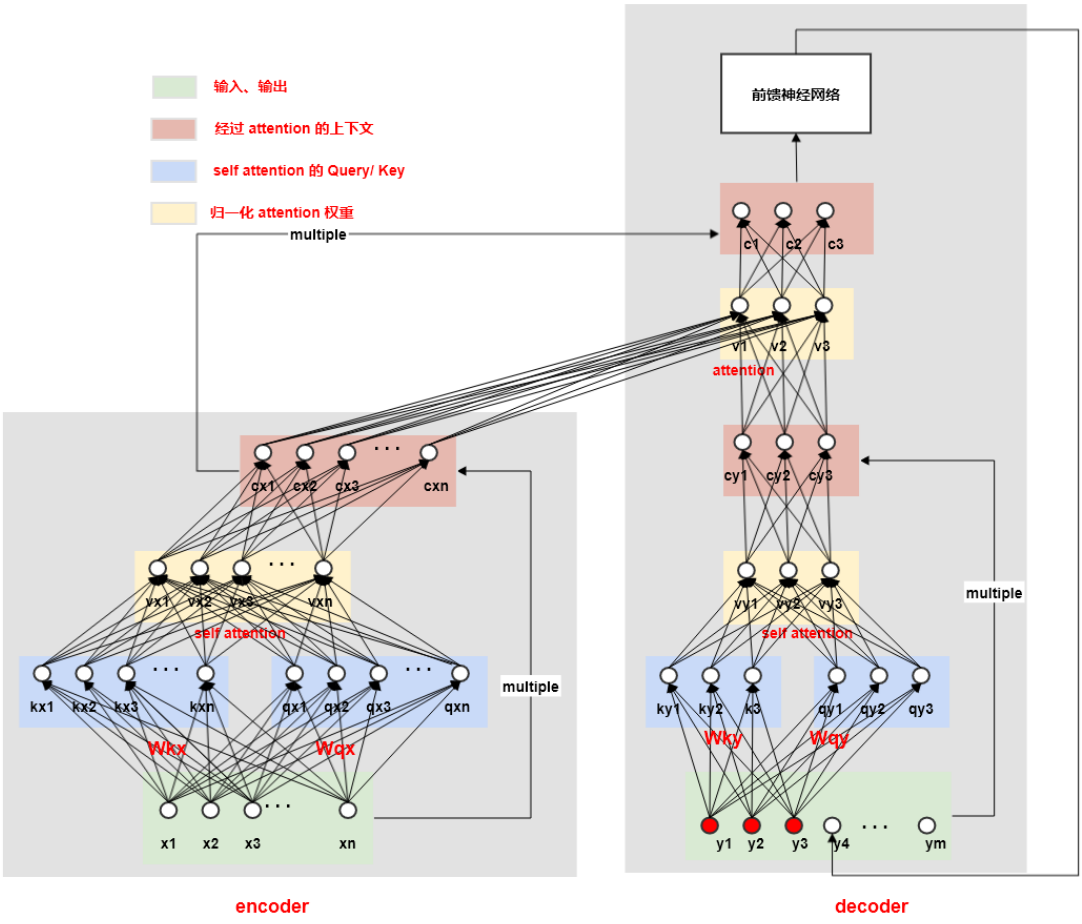
`

4.5.2 self attention

1. 传统的 `attention` 是基于 `encoder` 端和 `decoder` 端的隐向量来计算 `attention` 的，得到的是输入序列的每个 `input` 和输出序列的每个 `output` 之间的依赖关系。

   `self attention` 计算三种 `attention`：

   - 在 `encoder` 端计算自身的 `attention`，捕捉 `input` 之间的依赖关系。
   - 在 `decoder` 端计算自身的 `attention`，捕捉 `output` 之间的依赖关系。
   - 将 `encoder` 端得到的 `self attention` 加入到 `decoder` 端得到的 `attention` 中，捕捉输入序列的每个 `input` 和输出序列的每个 `output` 之间的依赖关系。



2. 设输入序列为 $\{\vec{\mathbf{x}}^{(1)},\vec{\mathbf{x}}^{(2)},\cdots,\vec{\mathbf{x}}^{(\tau_x)}\}$，输出序列为 $\{\vec{\mathbf{y}}^{(1)},\vec{\mathbf{y}}^{(2)},\cdots,\vec{\mathbf{y}}^{(\tau_y)}\}$，长度 $\tau_x\neq\tau_y$。

   - `encoder` 端的 `self attention`：
     - 首先经过 $\mathbf{W}_k^{encode}$ 映射得到 `key` 向量序列 $\{\vec{\mathbf{k}}_1^{encode},\cdots,\vec{\mathbf{k}}_{\tau_x}^{encode}\}$、经过 $\mathbf{W}_q^{encode}$ 映射得到 `query` 向量序列 $\{\vec{\mathbf{q}}_1^{encode},\cdots,\vec{\mathbf{q}}_{\tau_x}^{encode}\}$。
     - 然后计算归一化的 `self attention`：

       $$e_{i,t}=\text{score}(\vec{\mathbf{q}}_i^{encode},\vec{\mathbf{k}}_t^{encode}),\quad v_{i,t}^{encode}=\frac{\exp(e_{i,t})}{\sum_{t'=1}^{\tau_x}\exp(e_{i,t'})},\quad t=1,2,\cdots,\tau_x$$

       其中 $\vec{\mathbf{v}}_i^{encode}=(v_{i,1}^{encode},\cdots,v_{i,\tau_x}^{encode})$ 表示各输入与第 i 个输入的相关因子，$i=1,2,\cdots,\tau_x$。
     - 最后得到经过加权的 `encoder` 输出向量：$\vec{\mathbf{c}}_i^{encode}=\sum_{t=1}^{\tau_x}v_{i,t}^{encode}\times\vec{\mathbf{x}}^{(t)}$。
   - `decoder` 端的 `self attention`：考虑到解码时 s 时刻不知道 s 时刻及其之后的结果，因此仅考虑 s 时刻之前的 `attention`，这被称作 `masked attention`。因此对于时刻 s：
     - 首先经过 $\mathbf{W}_k^{decode}$ 映射得到 `key` 向量序列 $\{\vec{\mathbf{k}}_1^{decode},\cdots,\vec{\mathbf{k}}_{s-1}^{decode}\}$、经过 $\mathbf{W}_q^{decode}$ 映射得到 `query` 向量序列 $\{\vec{\mathbf{q}}_1^{decode},\cdots,\vec{\mathbf{q}}_{s-1}^{decode}\}$。
     - 然后计算归一化的 `self attention`：

       $$\tilde{e}_{i,t}=\text{score}(\vec{\mathbf{q}}_i^{decode},\vec{\mathbf{k}}_t^{decode}),\quad v_{i,t}^{decode}=\frac{\exp(\tilde{e}_{i,t})}{\sum_{t'=1}^{s-1}\exp(\tilde{e}_{i,t'})},\quad t=1,2,\cdots,s-1$$

       其中 $\vec{\mathbf{v}}_i^{decode}=(v_{i,1}^{decode},\cdots,v_{i,s-1}^{decode})$ 表示各输入与第 i 个输入的相关因子，$i=1,2,\cdots,s-1$。
     - 最后得到经过加权的 `encoder` 输出向量：$\vec{\mathbf{c}}_i^{decode}=\sum_{t=1}^{s-1}v_{i,t}^{decode}\times\vec{\mathbf{y}}^{(t)}$。
   - `encoder` 和 `decoder` 的 `attention`：

- 计算归一化的 `self attention`：

$$\hat{e}_{i,t}=\text{score}(\vec{\mathbf{c}}_i^{decode},\vec{\mathbf{c}}_t^{encode}),\quad v_{i,t}=\frac{\exp(\hat{e}_{i,t})}{\sum_{t'=1}^{\tau_x}\exp(\hat{e}_{i,t'})},\quad t=1,2,\cdots,\tau_x$$

其中 $\vec{\mathbf{v}}_i=(v_{i,1},\cdots,v_{i,\tau_x})$ 表示各输入与第 i 个输出的相关因子，$i=1,2,\cdots,s-1$。

- 最后得到经过加权的 `attention` 上下文向量：$\vec{\mathbf{c}}_i=\sum_{t=1}^{\tau_x}v_{i,t}\times\vec{\mathbf{c}}_i^{encode}$。

- 最后将上下文向量 $\vec{\mathbf{c}}_i$ 作为一个前馈神经网络的输入，其输出就是 $\vec{\mathbf{y}}^{(s)}$。

3. 上述 `self attention` 机制完全抛弃了 `RNN` 的架构，著名的 `Transformer` 架构就是基于它来构建的。

   `self attention` 未能考虑到输入序列的先后顺序，因此 `Transformer` 架构中引入了位置 `embedding` 来解决该问题。

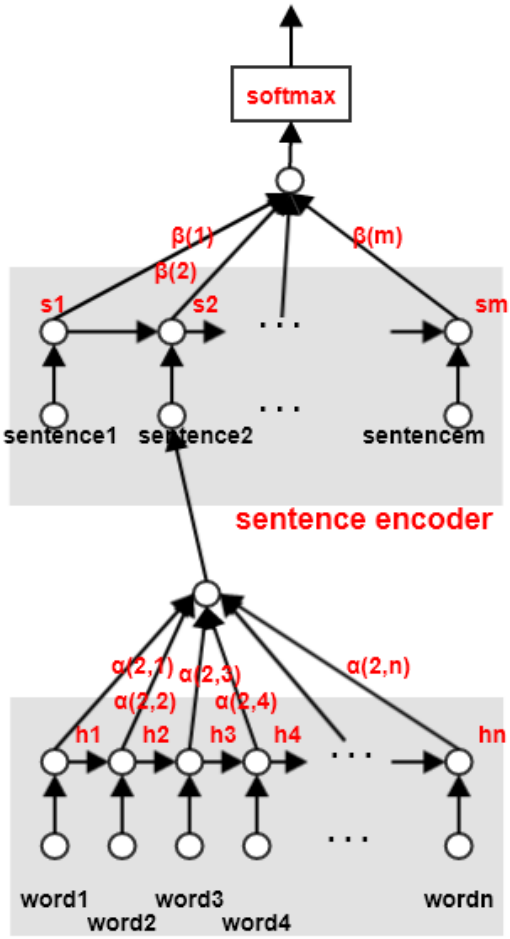
4. 理论上序列的 $\{\vec{\mathbf{x}}^{(1)},\vec{\mathbf{x}}^{(2)},\cdots,\vec{\mathbf{x}}^{(\tau_x)}\}$ 的 `self attention` 只需要简单计算：

$$e_{i,j}=\text{score}(\vec{\mathbf{x}}^{(i)},\vec{\mathbf{x}}^{(j)}),\quad v_{i,j}=\frac{\exp(e_{i,j})}{\sum_{j'=1}^{\tau_x}\exp(e_{i,j'})},\quad j=1,2,\cdots,\tau_x$$

   引入两个映射矩阵是为了更好的泛化：`attention` 并不仅仅考虑序列的原始信息，而是考虑了从序列中抽取的信息。

4.5.3 Hierarchical attention

1. 在论文 `《Hierarchical Attention Networks for Document Classification》` 中提出了分层 `attention` 用于文档分类。论文提出了两个层次的 `attention`：
   - 第一个层次是对句子中每个词进行 `attention`，即 `word attention`。
   - 第二个层次是对文档中每个句子进行 `attention`，即 `sentence attention`。



层次 attention

2. 层次 `attention` 涉及到四个部分：

- `word encoder`：（对于句子 i ）
  - `word embedding`：$\vec{\mathbf{x}}_i^{(t)} = \mathbf{W}_e^T \text{word}_i^{(t)}, t = 1, 2, \cdots, T$ 。
  - `GRU` 隐向量（原始论文中采用双向 `GRU` ）：

$$\overrightarrow{\mathbf{h}}_i^{(t)} = \overrightarrow{\text{GRU}}(\vec{\mathbf{x}}_i^{(t)}), \quad \overleftarrow{\mathbf{h}}_i^{(t)} = \overleftarrow{\text{GRU}}(\vec{\mathbf{x}}_i^{(t)})$$

- `word attention`：

$$\mathbf{h}_i^{(t)} = \left[\overrightarrow{\mathbf{h}}_i^{(t)} : \overleftarrow{\mathbf{h}}_i^{(t)}\right], \quad \vec{\mathbf{u}}_i^{(t)} = \tanh\left(\mathbf{W}_w \mathbf{h}_i^{(t)} + \vec{\mathbf{b}}_w\right)$$

$$\alpha_i^{(t)} = \frac{\exp(\vec{\mathbf{u}}_i^{(t)} \cdot \vec{\mathbf{u}}_w)}{\sum_{t'} \exp(\vec{\mathbf{u}}_i^{(t')} \cdot \vec{\mathbf{u}}_w)}, \quad \vec{\mathbf{s}}_i = \sum_{t'} \alpha_i^{(t)} \mathbf{h}_i^{(t)}$$

  其中 $\vec{\mathbf{u}}_w$ 表示这个单词序列的总信息，称作单词上下文。它是随机初始化并从网络训练得到。

- `sentence encoder`：

$$\overrightarrow{\mathbf{g}}_i = \overrightarrow{\text{GRU}}(\vec{\mathbf{s}}_i), \quad \overleftarrow{\mathbf{g}}_i = \overleftarrow{\text{GRU}}(\vec{\mathbf{s}}_i), \quad i = 1, 2, \cdots, L$$

- `sentence attention`：

$$\mathbf{g}_i = \left[\overrightarrow{\mathbf{g}}_i : \overleftarrow{\mathbf{g}}_i\right], \quad \vec{\mathbf{w}}_i = \tanh\left(\mathbf{W}_s \mathbf{g}_i + \vec{\mathbf{b}}_s\right)$$

$$\beta_i = \frac{\exp(\vec{\mathbf{w}}_i \cdot \vec{\mathbf{u}}_s)}{\sum_{i'} \exp(\vec{\mathbf{w}}_{i'} \cdot \vec{\mathbf{u}}_s)}, \quad \vec{\mathbf{v}} = \sum_{i'} \beta_{i'} \mathbf{g}_{i'}$$

其中 $\vec{\mathbf{u}}_s$ 表示这个句子序列的总信息，称作句子上下文。它是随机初始化并从网络训练得到。