循环神经网络

1. 循环神经网络`recurrent neural network:RNN`：用于处理序列数据 $\vec{\mathbf{x}}^{(1)},\vec{\mathbf{x}}^{(2)},\cdots,\vec{\mathbf{x}}^{(\tau)}$ 的神经网络。
    - 每个样本 $\mathbf{x}$ 就是一个序列 $\mathbf{x}=\{\vec{\mathbf{x}}^{(1)},\vec{\mathbf{x}}^{(2)},\cdots,\vec{\mathbf{x}}^{(\tau)}\}$ 。
    - 序列的长度可以是固定的。

      对每个样本，其序列长度都是 L ；其中 L 可以很大，也可以很小。
    - 序列的长度也可以是可变的。

      对第 i 个样本 $\mathbf{x}_i$ ，令其序列长度为 L_i 。则 L_i 可能不等于 $L_j,i\ne j$ 。

   > 相比较而言，卷积神经网络专门处理网格化数据 $\mathbf{X}$ （如一个图形）。
   >
   > 它可以扩展到具有很大宽度和高度的网格。

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。

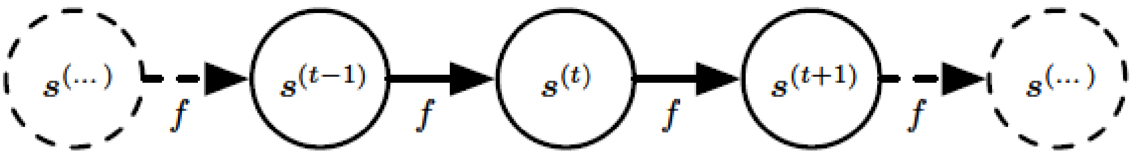
3. 就像几乎所有函数都可以被认为是前馈神经网络，几乎任何涉及循环的函数都可以被认为是循环神经网络。

一、RNN计算图

1. 考虑动态系统的经典形式：$\vec{\mathbf{s}}^{(t)}=f(\vec{\mathbf{s}}^{(t-1)};\vec{\theta})$ 。其中：$\vec{\mathbf{s}}^{(t)}$ 称作系统的状态，$\vec{\theta}$ 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：
   $$\vec{\mathbf{s}}^{(\tau)}=f(\vec{\mathbf{s}}^{(\tau-1)};\vec{\theta})=\cdots=f(\cdots f(\vec{\mathbf{s}}^{(1)};\vec{\theta})\cdots;\vec{\theta})$$

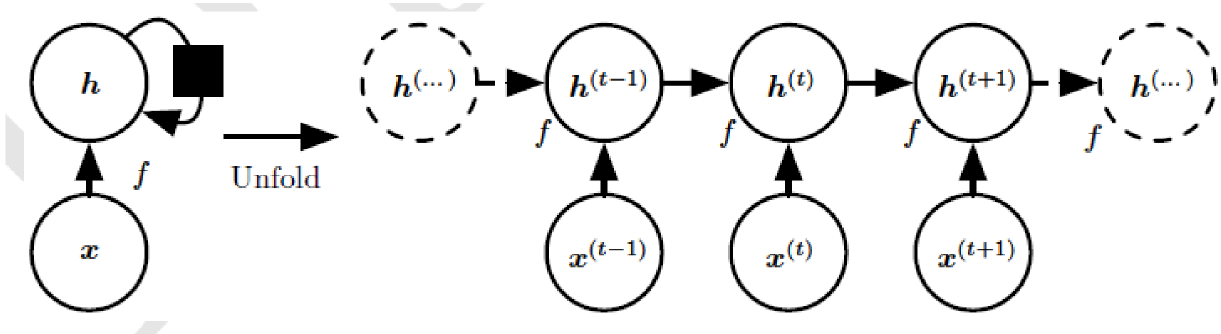
   利用有向无环图来表述：



2. 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：
   $$\vec{\mathbf{s}}^{(t)}=f(\vec{\mathbf{s}}^{(t-1)},\vec{\mathbf{x}}^{(t)};\vec{\theta})$$

   其展开图如下：左侧为循环图，右侧为展开图。黑色方块表示单位延时。

   为了表明状态 $\vec{\mathbf{s}}$ 就是神经网络的隐单元，这里用变量 $\vec{\mathbf{h}}$ 代表状态。则上式重写为：
   $$\vec{\mathbf{h}}^{(t)}=f(\vec{\mathbf{h}}^{(t-1)},\vec{\mathbf{x}}^{(t)};\vec{\theta})$$



3. 当训练`RNN`根据过去预测未来时，网络通常要将 $\vec{\mathbf{h}}^{(t)}$ 作为过去序列的一个有损的摘要。
   - 这个摘要一般是有损的。因为它使用一个固定长度的向量 $\vec{\mathbf{h}}^{(t)}$ 来映射任意长的序列 $(\vec{\mathbf{x}}^{(t-1)},\cdots,\vec{\mathbf{x}}^{(1)})$ 。
   - 根据不同的训练准则，摘要可能会有选择地保留过去序列的某些部分。
4. 展开图的两个主要优点：
   - 无论输入序列的长度如何，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，其输入都是相同大小的。
   - 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 $\vec{\theta}$ 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

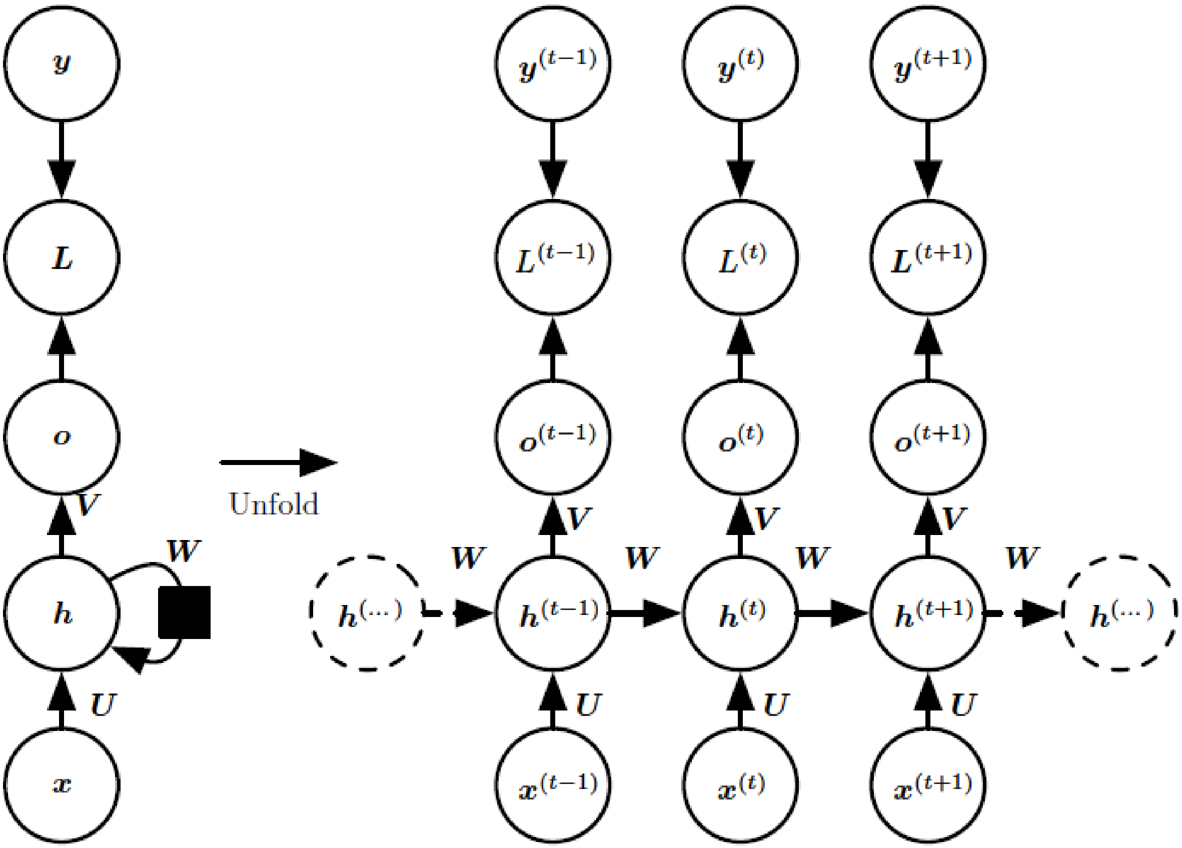
二、循环神经网络

2.1 网络模式

1. 基于图展开和参数共享的思想，可以设计各种模式的循环神经网络。
2. 下面的模式图中：左图为循环图，右图为计算图。L 为损失函数，衡量每个输出 $\vec{\mathbf{o}}$ 与标记 $\vec{\mathbf{y}}$ 的距离。

2.1.1 多输出&隐-隐连接

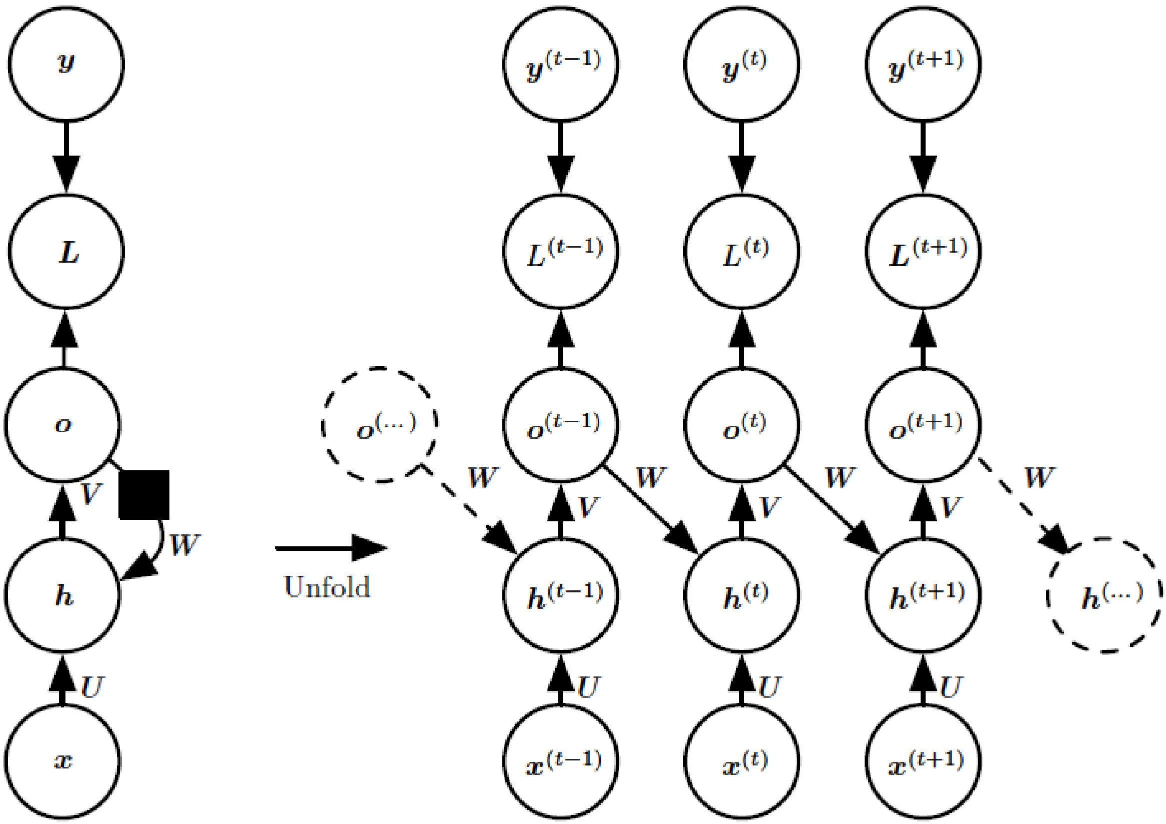
1. `多输出&隐-隐连接`循环网络：每个时间步都有输出，并且隐单元之间有循环连接。



2. `多输出&隐-隐RNN`将一个输入序列映射到相同长度的输出序列。
3. 任何图灵可计算的函数都可以通过一个有限维的循环网络计算。在这个意义上，`多输出&隐-隐连接`循环网络是万能的。

2.1.2 多输出&输出-隐连接

1. `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。



2. `多输出&输出-隐RNN` 将一个输入序列映射到相同长度的输出序列。
3. `多输出&输出-隐连接` 循环网络只能表示更小的函数集合。
    - `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：只有输出 $\vec{\mathbf{o}}$ 会被传播信息到未来。

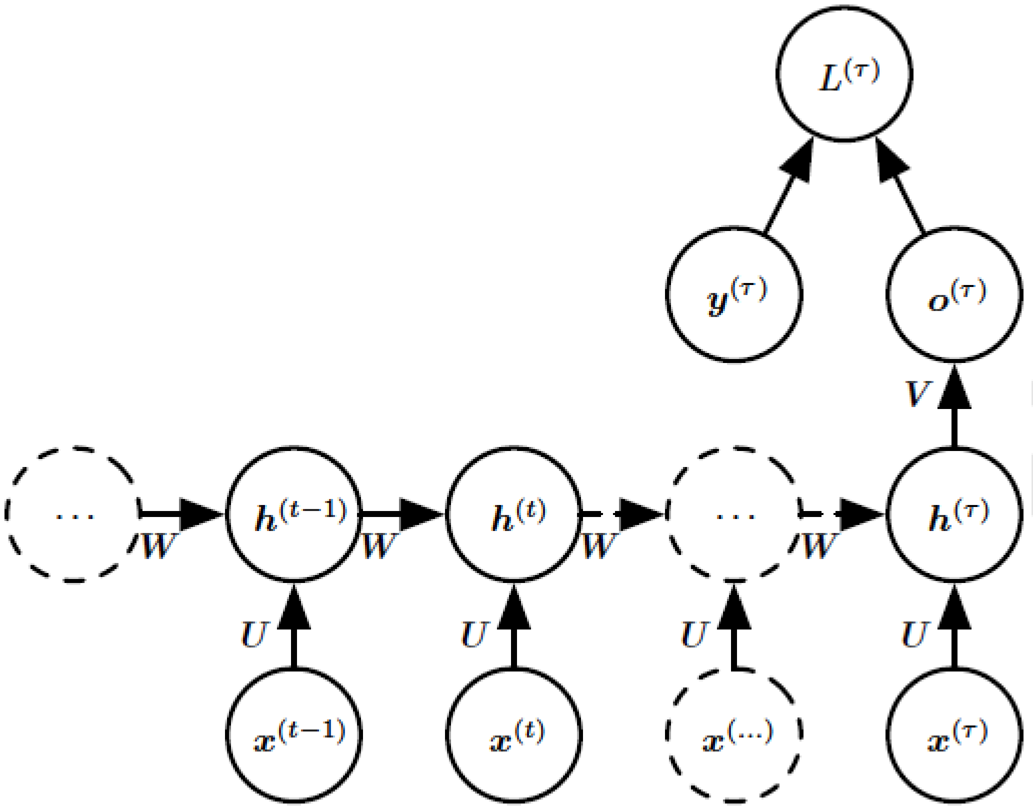
      通常 $\vec{\mathbf{o}}$ 的维度远小于 $\vec{\mathbf{h}}$，并且缺乏过去的重要信息。
4. `多输出&输出-隐连接` 循环网络虽然表达能力不强，但是更容易训练：因为每个时间步可以与其他时间步分离训练，从而允许训练期间更多的并行化（使用前一个时间步的真实标记 $\vec{\mathbf{y}}$ 来代替输出 $\vec{\mathbf{o}}$ ）。
5. 对于序列输入-序列输出的 `RNN`，网络对应了条件分布：$P(\vec{\mathbf{y}}^{(1)},\cdots,\vec{\mathbf{y}}^{(\tau)}\mid\vec{\mathbf{x}}^{(1)},\cdots,\vec{\mathbf{x}}^{(\tau)})$ 。
    - 在 `多输出&隐-隐RNN` 中，由于给定输入序列的条件下输出是条件独立的，因此有：

$$P(\vec{\mathbf{y}}^{(1)},\cdots,\vec{\mathbf{y}}^{(\tau)}\mid\vec{\mathbf{x}}^{(1)},\cdots,\vec{\mathbf{x}}^{(\tau)})=\prod_t P(\vec{\mathbf{y}}^{(t)}\mid\vec{\mathbf{x}}^{(1)},\cdots,\vec{\mathbf{x}}^{(t)})$$

    - 在 `多输出&输出-隐连接RNN` 中，通过添加了时刻 t 的输出到时刻 $t+1$ 的隐单元的连接，打破了这种条件独立性。

2.1.3 单输出&隐-隐连接

1. `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。



2. `单输出&隐-隐连接RNN`将一个输入序列映射到单个输出。

2.2 BPTT 算法

1. 假设真实标记 $\vec{\mathbf{y}}$ 的取值是一个离散的标量，如类别：$1,2,3,\cdots,K$。假设类别为 l，则可以将真实标记记做（第 l 个位置为1,其它位置全0）：

$$\vec{\mathbf{y}}=(0,0,\cdots,0,1,0,\cdots,0)^T$$

将输出 $\vec{\mathbf{o}}$ 应用 `softmax` 函数处理后，获得标准化的各类别概率的输出向量 $\hat{\vec{\mathbf{y}}}$。

$$\hat{\vec{\mathbf{y}}}^{(t)}=\text{softmax}(\vec{\mathbf{o}}^{(t)})=\left(\frac{\exp(o_1^{(t)})}{\sum_{j=1}^K\exp(o_j^{(t)})},\frac{\exp(o_2^{(t)})}{\sum_{j=1}^K\exp(o_j^{(t)})},\cdots,\frac{\exp(o_K^{(t)})}{\sum_{j=1}^K\exp(o_j^{(t)})}\right)^T$$

- softmax 函数在展开图中并没有显式给出。大多数情况下，它是作为计算损失函数 $L^{(t)}$ 的一部分。
- 损失函数 $L^{(t)}$ 为：（l 为真实类别）

$$L^{(t)}=-\log\frac{\exp(o_l^{(t)})}{\sum_{j=1}^K\exp(o_j^{(t)})}$$

它就是 $\hat{\vec{\mathbf{y}}}$ 中对应于 l 的分量。

2. `多输出&隐-隐RNN`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，更新方程：

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

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 $\mathbf{W}$
- $\vec{\mathbf{b}},\vec{\mathbf{c}}$ 为偏置向量
- 激活函数为双曲正切激活函数 $\tanh$

3. 对给定样本 $(\mathbf{x},\mathbf{y})$，其输入 $\mathbf{x}$ 和标记 $\mathbf{y}$ 均为一个序列。假设 $\mathbf{x}=\{\vec{\mathbf{x}}^{(1)},\vec{\mathbf{x}}^{(2)},\cdots,\vec{\mathbf{x}}^{(\tau)}\}$，$\mathbf{y}=\{\vec{\mathbf{y}}^{(1)},\vec{\mathbf{y}}^{(2)},\cdots,\vec{\mathbf{y}}^{(\tau)}\}$。

   假设损失函数 $L^{(t)}$ 为给定 $\vec{\mathbf{x}}^{(1)},\vec{\mathbf{x}}^{(2)},\cdots,\vec{\mathbf{x}}^{(t)}$ 的条件下，输出为 $\vec{\mathbf{y}}^{(t)}$ 的负对数似然。根据样本的损失为所有时间步的损失之和，则有：

$$L(\mathbf{x},\mathbf{y})=\sum_{t=1}^{t=\tau}L^{(t)}=-\sum_{t=1}^{t=\tau}\log p_{model}\left(\vec{\mathbf{y}}^{(t)}\mid\{\vec{\mathbf{x}}^{(1)},\vec{\mathbf{x}}^{(2)},\cdots,\vec{\mathbf{x}}^{(t)}\}\right)$$

   其中 $p_{model}\left(\vec{\mathbf{y}}^{(t)}\mid\{\vec{\mathbf{x}}^{(1)},\vec{\mathbf{x}}^{(2)},\cdots,\vec{\mathbf{x}}^{(t)}\}\right)$ 需要读取模型输出向量 $\hat{\vec{\mathbf{y}}}^{(t)}$ 中对应于 $\vec{\mathbf{y}}^{(t)}$ 中非零位置的那个分量。

4. 损失函数 $L(\mathbf{x},\mathbf{y})$ 的梯度计算是昂贵的。

   梯度计算涉及执行一次前向传播，一次反向传播。

   - 因为每个时间步只能一前一后的计算，无法并行化，运行时间为 $O(\tau)$
   - 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，因此内存代价也是 $O(\tau)$

   在展开图中代价为 $O(\tau)$ 的反向传播算法称作通过时间反向传播 `back-propagation through time:BPTT`

5. 由反向传播计算得到梯度，再结合任何通用的、基于梯度的技术就可以训练 `RNN`。

2.2.1 输出单元的梯度

1. 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 $\vec{\mathbf{x}}^{(t)},\vec{\mathbf{h}}^{(t)},\vec{\mathbf{o}}^{(t)}$ 以及 $L^{(t)}$。
2. 根据

$$L\left(\vec{\mathbf{x}}^{(1)},\vec{\mathbf{x}}^{(2)},\cdots,\vec{\mathbf{x}}^{(\tau)},\vec{\mathbf{y}}^{(1)},\vec{\mathbf{y}}^{(2)},\cdots,\vec{\mathbf{y}}^{(\tau)}\right)=\sum_{t=1}^{t=\tau}L^{(t)}$$

   得到：$\frac{\partial L}{\partial L^{(t)}}=1$

3. 假设 l_t 表示 $\vec{\mathbf{y}}^{(t)}$ 所属的类别。$\hat{y}_{l_t}^{(t)}$ 为模型预测为类别 l_t 的概率。则损失函数为：给定了迄今为止的输入后，真实目标 $\vec{\mathbf{y}}^{(t)}$ 的负对数似然

$$L^{(t)}=-\log\hat{y}_{l_t}^{(t)}=-\log\frac{\exp(o_{l_t}^{(t)})}{\sum_{j=1}^{K}\exp(o_j^{(t)})}=-o_{l_t}^{(t)}+\log\sum_{j=1}^{K}\exp(o_j^{(t)})$$

4. 损失函数对于输出 $\vec{\mathbf{o}}^{(t)}$ 的梯度为：

$$\begin{aligned}(\nabla_{\vec{\mathbf{o}}^{(t)}}L)_i&=\frac{\partial L}{\partial o_i^{(t)}}=\frac{\partial L}{\partial L^{(t)}}\times\frac{\partial L^{(t)}}{\partial o_i^{(t)}}=1\times\frac{\partial L^{(t)}}{\partial o_i^{(t)}}\\&=-\mathbf{1}_{i,l_t}+\frac{\exp(o_i^{(t)})}{\sum_{j=1}^{n}\exp(o_j^{(t)})}=\hat{y}_i^{(t)}-\mathbf{1}_{i,l_t}\end{aligned}$$

其中：

- $(\nabla_{\vec{\mathbf{o}}^{(t)}}L)_i$ 表示梯度的第 i 个分量
- $o_i^{(t)}$ 表示输出的第 i 个分量
- l_t 表示 $\vec{\mathbf{y}}^{(t)}$ 所属的真实类别
- $\hat{y}_i^{(t)}$ 表示模型预测为类别 i 的概率
- $\mathbf{1}$ 为示性函数：

$$\mathbf{1}_{i,l_t} = \begin{cases} 1, & i = l_t \\ 0, & i \neq l_t \end{cases}$$

2.2.2 隐单元的梯度

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

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

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

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

记：（N 为隐向量的长度）

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

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

则有：

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

2. 因为 $\vec{\mathbf{o}}^{(t)}=\vec{\mathbf{c}}+\mathbf{V}\vec{\mathbf{h}}^{(t)}$，即 $o_i^{(t)}=c_i+\sum_j V_{i,j}h_j^{(t)}$ 。

   则有：$\frac{\partial o_i^{(t)}}{\partial h_j^{(t)}}=V_{i,j}$ 。记做：

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

3. 当 $t=\tau$ 时，$\vec{\mathbf{h}}^{(\tau)}$ 只有一个后续结点 $\vec{\mathbf{o}}^{(\tau)}$，因此有：

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

4. 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：

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

   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。

2.2.3 网络参数的梯度

1. 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。
2. 由于参数在多个时间步共享，因此在参数节点的微积分操作时必须谨慎对待。

   微积分中的算子 $\nabla_{\mathbf{A}}f$ 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 $\nabla_{\mathbf{A}}f$ 。
3. 根据 $\vec{\mathbf{o}}^{(t)}=\vec{\mathbf{c}}^{(t)}+\mathbf{V}\vec{\mathbf{h}}^{(t)}$ ，则有：

$$\frac{\partial \vec{\mathbf{o}}^{(t)}}{\partial \vec{\mathbf{c}}^{(t)}} = \mathbf{I}$$

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：

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

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

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

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

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

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

$$\frac{\partial o_i^{(t)}}{\partial V_{i,j}^{(t)}} = h_j^{(t)}$$

记做：

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

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：

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

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

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

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

记做：

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

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。

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

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

则有：

$$\frac{\partial h_i^{(t)}}{\partial U_{i,j}^{(t)}}=(1-h_i^{(t)2})x_j^{(t)}$$

记做：

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

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。

8. 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 $\nabla_{\vec{\mathbf{x}}^{(t)}} L$

2.3 Teacher forcing 算法

1. 对于`多输出&输出-隐连接RNN`，由于它缺乏隐状态到隐状态的循环连接，因此它无法模拟通用图灵机。

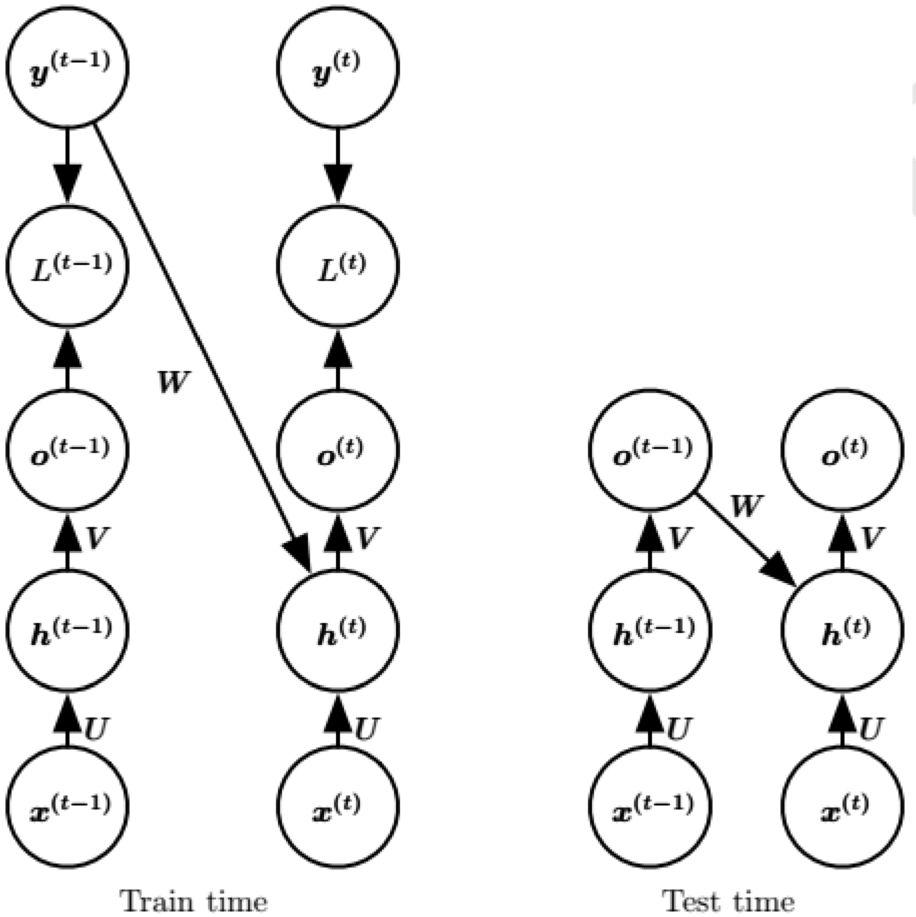
   其优点在于：训练可以解耦，各时刻 t 分别计算梯度。

2. `多输出&输出-隐连接RNN`模型可以使用`teacher forcing`算法进行训练。

   该方法在时刻 $t+1$ 接受真实值 $\vec{\mathbf{y}}^{(t)}$ 作为输入，而不必等待 t 时刻的模型输出。如下图所示为`teacher forcing`过程：

   - 训练过程：如左图所示，真实标记 $\vec{\mathbf{y}}^{(t-1)}$ 反馈到 $\vec{\mathbf{h}}^{(t)}$ 。
   - 推断过程：如右图所示，模型输出 $\vec{\mathbf{o}}^{(t-1)}$ 反馈到 $\vec{\mathbf{h}}^{(t)}$ 。

     因为推断时，真实的标记通常是未知的。因此必须用模型的输出 $\vec{\mathbf{o}}^{(t-1)}$ 来近似真实标记 $\vec{\mathbf{y}}^{(t-1)}$ 。



3. 考察只有两个时间步的序列。对数似然函数为:

$$\log p(\vec{\mathbf{y}}^{(1)},\vec{\mathbf{y}}^{(2)}\mid\vec{\mathbf{x}}^{(1)},\vec{\mathbf{x}}^{(2)})=\log p(\vec{\mathbf{y}}^{(2)}\mid\vec{\mathbf{y}}^{(1)},\vec{\mathbf{x}}^{(1)},\vec{\mathbf{x}}^{(2)})+\log p(\vec{\mathbf{y}}^{(1)}\mid\vec{\mathbf{x}}^{(1)},\vec{\mathbf{x}}^{(2)})$$

   对右侧两部分分别取最大化。则可以看到:

   - 在时间步 $t=1$,模型被训练为最大化 $\vec{\mathbf{y}}^{(1)}$ 的条件概率 $\log p(\vec{\mathbf{y}}^{(1)}\mid\vec{\mathbf{x}}^{(1)},\vec{\mathbf{x}}^{(2)})$ 。
   - 在时间步 $t=2$,模型被训练为最大化 $\vec{\mathbf{y}}^{(2)}$ 的条件概率 $\log p(\vec{\mathbf{y}}^{(2)}\mid\vec{\mathbf{y}}^{(1)},\vec{\mathbf{x}}^{(1)},\vec{\mathbf{x}}^{(2)})$ 。

4. `teacher forcing`训练的本质原因是:当前隐状态与早期隐状态没有连接(虽然有间接连接,但是由于 $\vec{\mathbf{y}}^{(t)}$ 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

2.4 损失函数

1. 可以将真实的标记 $\vec{\mathbf{y}}^{(t)}=(0,\cdots,0,1,0,\cdots,0)$ 视作一个概率分布。其中在真实的类别 l_t 位置上其分量为 1,而其它位置上的分量为 0。

   模型归一化输出 $\hat{\vec{\mathbf{y}}}^{(t)}$ 也是一个概率分布,它给出了预测该样本为各类别的概率。

   将`RNN`的代价函数 $L^{(t)}$ 定义为:训练目标 $\vec{\mathbf{y}}^{(t)}$ 和归一化输出 $\hat{\vec{\mathbf{y}}}^{(t)}$ 之间的交叉熵。

$$L^{(t)}=-\sum_i y_i^{(t)}\log\hat{y}_i^{(t)}=-\log\hat{y}_{l_t}^{(t)}$$

2. 原则上可以使用任何损失函数,但是必须根据具体任务来选择一个合适的损失函数。

3. 如果使用对数似然作为建模策略，则这里有两种选择：
   - 根据之前的输入 $\{\vec{\mathbf{x}}^{(1)},\cdots,\vec{\mathbf{x}}^{(t)}\}$，估计下一个序列元素 $\vec{\mathbf{y}}^{(t)}$ 的条件分布。即最大化对数似然函数：
     $$\log p(\vec{\mathbf{y}}^{(t)} \mid \vec{\mathbf{x}}^{(1)},\cdots,\vec{\mathbf{x}}^{(t)})$$
     此时 $\hat{\vec{\mathbf{y}}}^{(t)}=p(\vec{\mathbf{y}}^{(t)} \mid \vec{\mathbf{x}}^{(1)},\cdots,\vec{\mathbf{x}}^{(t)})$
   - 如果模型包含了前一个时间步的输出到下一个时间步的连接（比如`多输出&输出-隐连接RNN`），则最大化对数似然函数：
     $$\log p(\vec{\mathbf{y}}^{(t)} \mid \vec{\mathbf{x}}^{(1)},\cdots,\vec{\mathbf{x}}^{(t)},\vec{\mathbf{y}}^{(1)},\cdots,\vec{\mathbf{y}}^{(t-1)})$$
     此时 $\hat{\vec{\mathbf{y}}}^{(t)}=p(\vec{\mathbf{y}}^{(t)} \mid \vec{\mathbf{x}}^{(1)},\cdots,\vec{\mathbf{x}}^{(t)},\vec{\mathbf{y}}^{(1)},\cdots,\vec{\mathbf{y}}^{(t-1)})$

2.5 输入序列长度

1. `RNN` 网络的输入序列的长度可以有多种类型：
   - 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：句子生成算法。
     - 首先选取一个词作为起始词
     - 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。

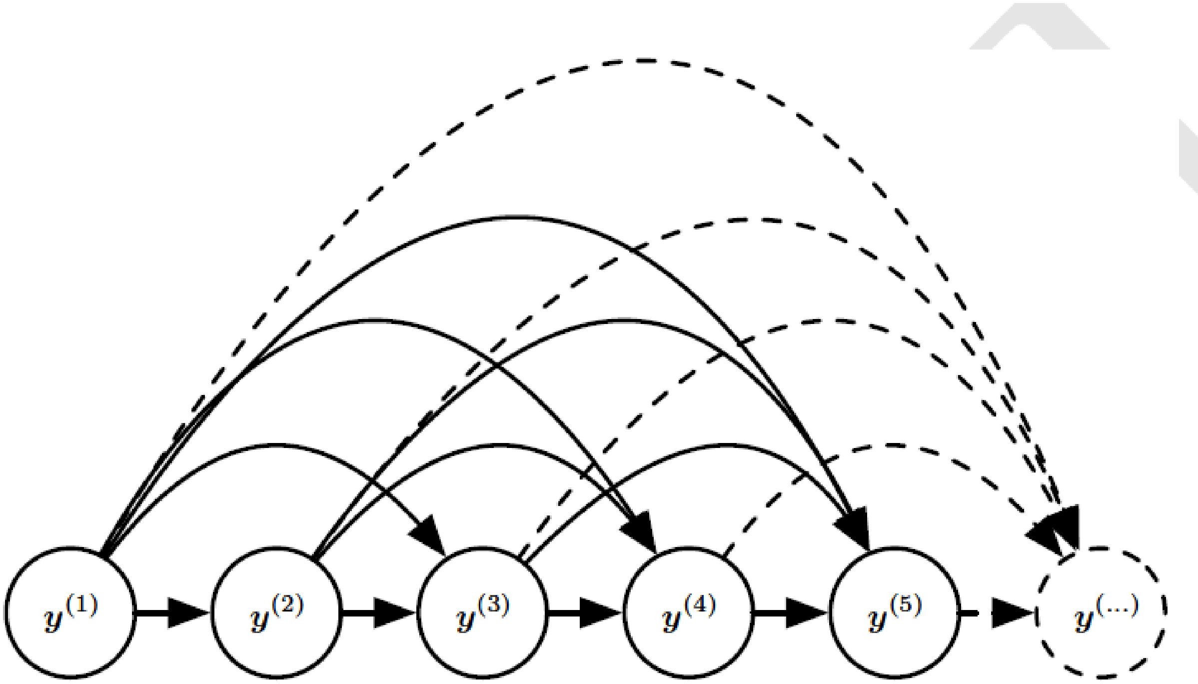
     在这个例子中，任何早期输出的单词都会对它后面的单词产生影响。
   - 输入序列长度为`1`：模型包含单个 $\vec{\mathbf{x}}$ 作为输入。
   - 输入序列长度为`N`：模型包含序列 $\{\vec{\mathbf{x}}_1,\vec{\mathbf{x}}_2,\cdots,\vec{\mathbf{x}}_N\}$ 作为输入。

     对于`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`，它们都是这种类型的输入。

2.5.1 零长度输入序列

1. 在零输入`RNN` 网络中，一个时间步拥有到任意后续时间步的连接。每一个连接代表着早期的输出对后续输出施加的影响。

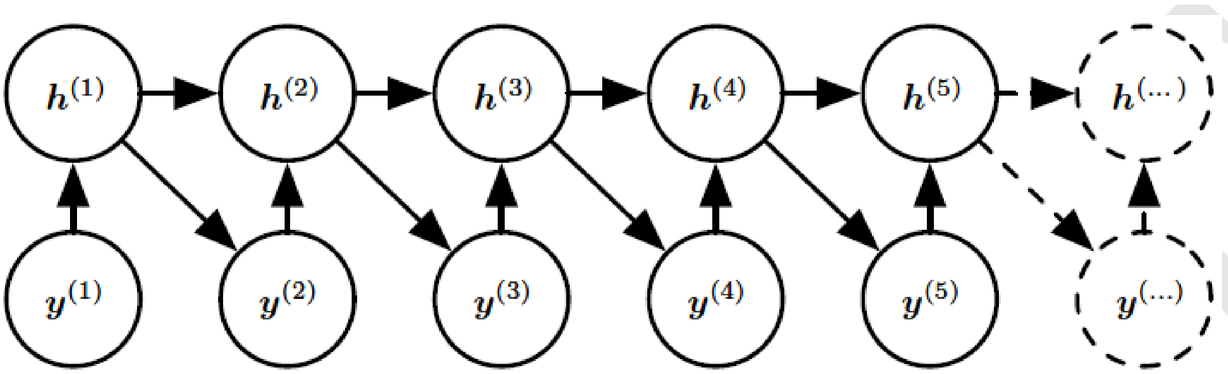
   此时网络的有向图非常复杂，如下所示。根据有向图直接参数化的做法非常低效，因为对于每一个连接，都需要给出一个参数。



2. `RNN` 的解决方案是：引入了状态变量 $\vec{\mathbf{h}}$。

   它作为连接过去和未来之间的桥梁：过去的输出 $\{\vec{\mathbf{y}}^{(1)},\vec{\mathbf{y}}^{(2)},\cdots,\vec{\mathbf{y}}^{(t-1)}\}$ 通过影响 $\vec{\mathbf{h}}$ 来影响当前的输出 $\vec{\mathbf{y}}^{(t)}$ ，从而解耦 $\vec{\mathbf{y}}^{(i)}$ 和 $\vec{\mathbf{y}}^{(t)}$ 。

   因此，状态变量有助于得到非常高效的参数化：序列中的每个时间步使用相同的结构，并且共享相同的参数。



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。

   给定时刻 t 的变量，时刻 $t+1$ 变量的条件概率分布是平稳的。这意味着：前一个时间步和后一个时间步之间的关系并不依赖于 t 。

2.5.2 单长度输入序列

1. 当 `RNN` 网络的输入序列长度为 `1` 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 $\vec{\mathbf{h}}^{(0)}$ ，每个时间步没有额外的输入。

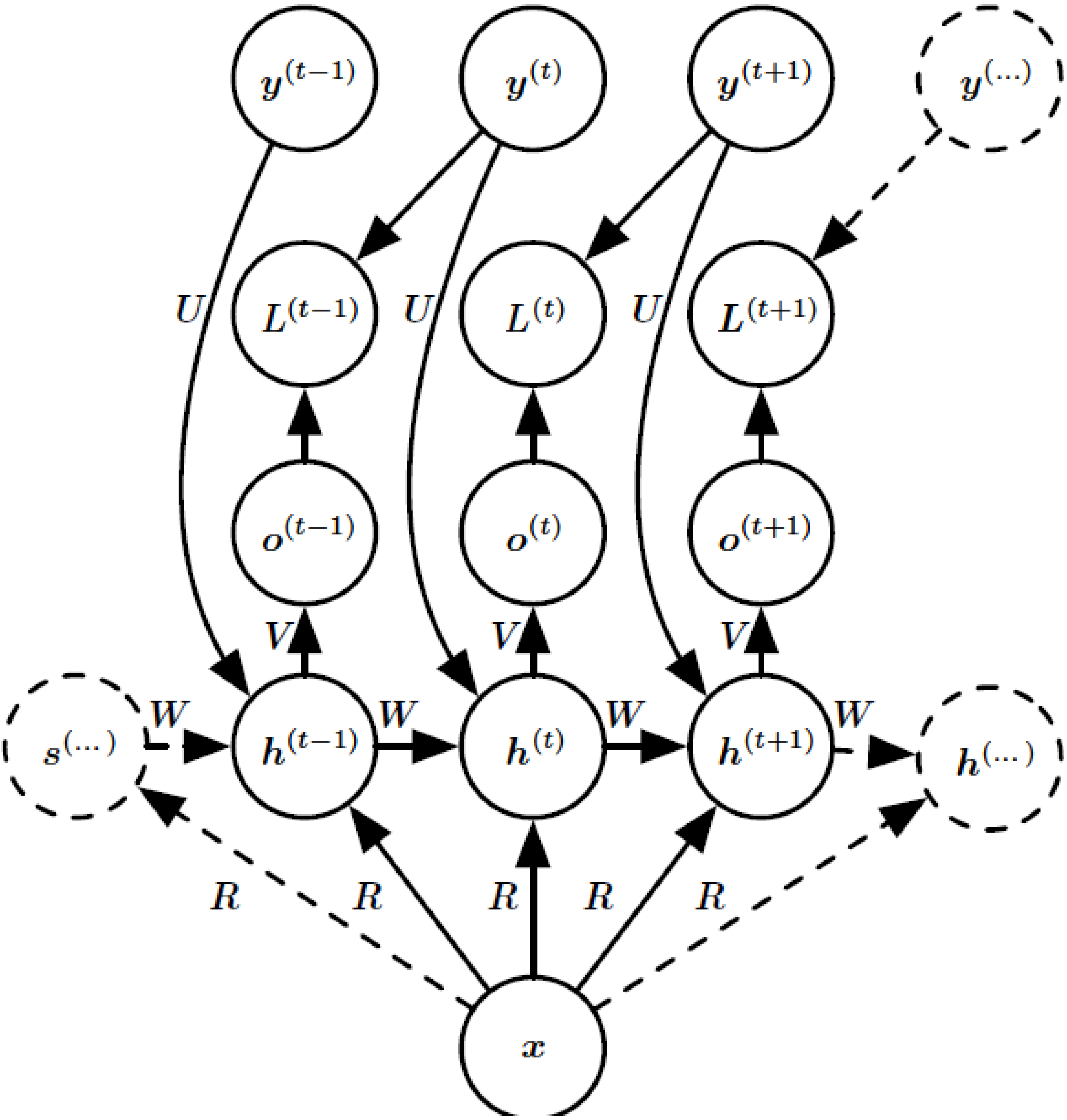
     此时输入 $\vec{\mathbf{x}}$ 仅仅直接作为 $\vec{\mathbf{h}}^{(1)}$ 的输入。
   - 结合上述两种方式。

2. 如图注任务：单个图像作为模型输入，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 t 时刻为止的单词和图像，来预测 $t+1$ 时刻的单词。

  一旦得到了 $t+1$ 时刻的单词，就需要通过 $t+1$ 时刻为止的单词和图像来预测 $t+2$ 时刻的单词。
- 输出序列的每个 $\vec{\mathbf{y}}^{(t)}$ 有两个作用：
  - 用作输入（对于当前时间步）来预测 $\vec{\mathbf{y}}^{(t+1)}$
  - 用于计算损失函数（对于前一个时间步）$L^{(t-1)}$。

  > $\vec{\mathbf{o}}^{(t-1)}$ 是通过 $\vec{\mathbf{h}}^{(t-1)}$ 和 $\vec{\mathbf{y}}^{(t-1)}$ 来计算得到的 t 时刻的预测输出，因此需要通过 $\vec{\mathbf{y}}^{(t)}$ 来计算损失。



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` 网络有如下区别：
   - 零输入 `RNN` 的初始输出 $\vec{\mathbf{y}}^{(0)}$ 是随机选择的；而这里的 $\vec{\mathbf{h}}^{(0)}$ 是给定的。
   - 零输入 `RNN` 初始化的是输出；而这里初始化的是隐变量。

2.6 输出序列长度

1. 当输入序列长度为`N` 时：

    - 对于 `多输出&隐-隐连接RNN`、`多输出&输出-隐连接RNN`：输出序列长度等于输入序列长度。
    - 对于`单输出&隐-隐连接RNN`：输出序列长度为`1`。

2. 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。

    如果输入序列长度为零，则训练集中的样本只有输出数据。这种 `RNN` 可以用于自动生成文章、句子等。

3. 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。当输出该标记符时，序列终止。

        在训练集中，需要对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伯努利输出单元，用于指示：每个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` 。
        - 该伯努利输出单元通常使用`sigmoid`单元，被训练为最大化正确地预测到序列结束的对数似然。

    - 添加一个额外的输出单元，它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： $P(\vec{\mathbf{y}}^{(1)},\vec{\mathbf{y}}^{(2)},\cdots,\vec{\mathbf{y}}^{(\tau)})=P(\tau)P(\vec{\mathbf{y}}^{(1)},\vec{\mathbf{y}}^{(2)},\cdots,\vec{\mathbf{y}}^{(\tau)}\mid\tau)$ 。

2.7 双向 RNN

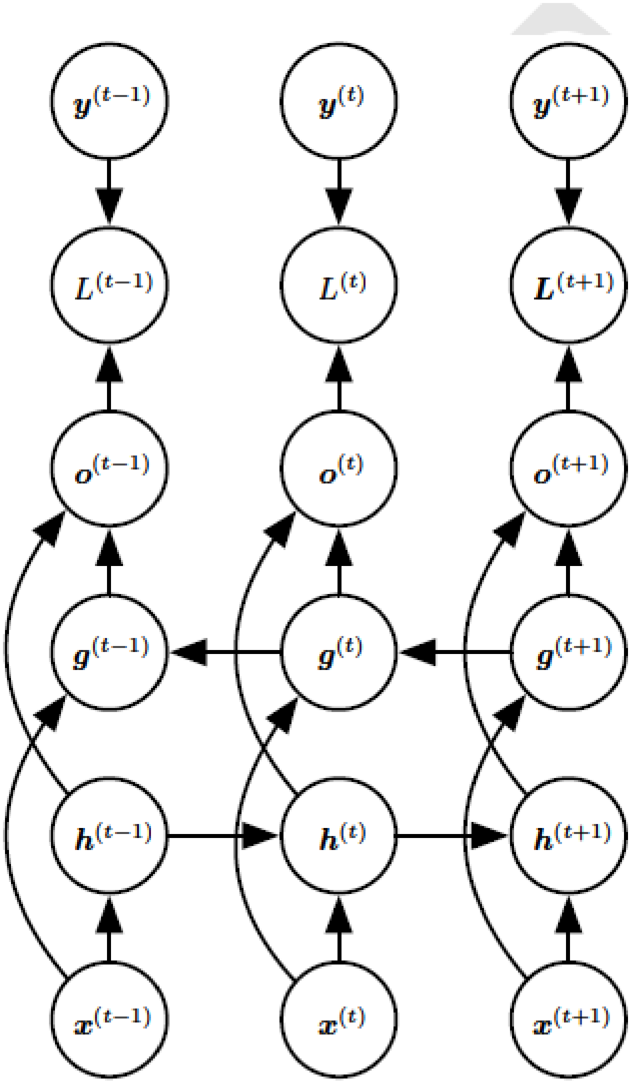
1. 前面介绍的`RNN` 隐含了一个假设：时刻 t 的状态只能由过去的序列 $\{\vec{\mathbf{x}}^{(1)},\vec{\mathbf{x}}^{(2)},\cdots,\vec{\mathbf{x}}^{(t-1)}\}$ ，以及当前的输入 $\vec{\mathbf{x}}^{(t)}$ 来决定。

    实际应用中，输出 $\vec{\mathbf{y}}^{(t)}$ 可能依赖于整个输入序列。如：语音识别中，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。

2. 双向`RNN`就是应对这种双向依赖问题而发明的，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。

3. 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
    - 输出单元 $\vec{\mathbf{o}}^{(t)}$ ：同时依赖于过去、未来、以及时刻 t 的输入 $\vec{\mathbf{x}}^{(t)}$



4. 如果输入是 2 维的，比如图像，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
    - 输出单元 $\vec{\mathbf{o}}^{(t)}$ 同时依赖于四个方向、以及时刻 t 的输入 $\vec{\mathbf{x}}^{(t)}$
5. 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。

2.8 深度RNN

1. 大多数`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。

    > 多层感知机内单个层来表示的变换称作浅层变换。

2. 事实上，可以这三种变换中引入深度。实验表明：引入深度会带来好处。

- 第一种引入深度的方式：将 `RNN` 的隐状态分为多层来引入深度。

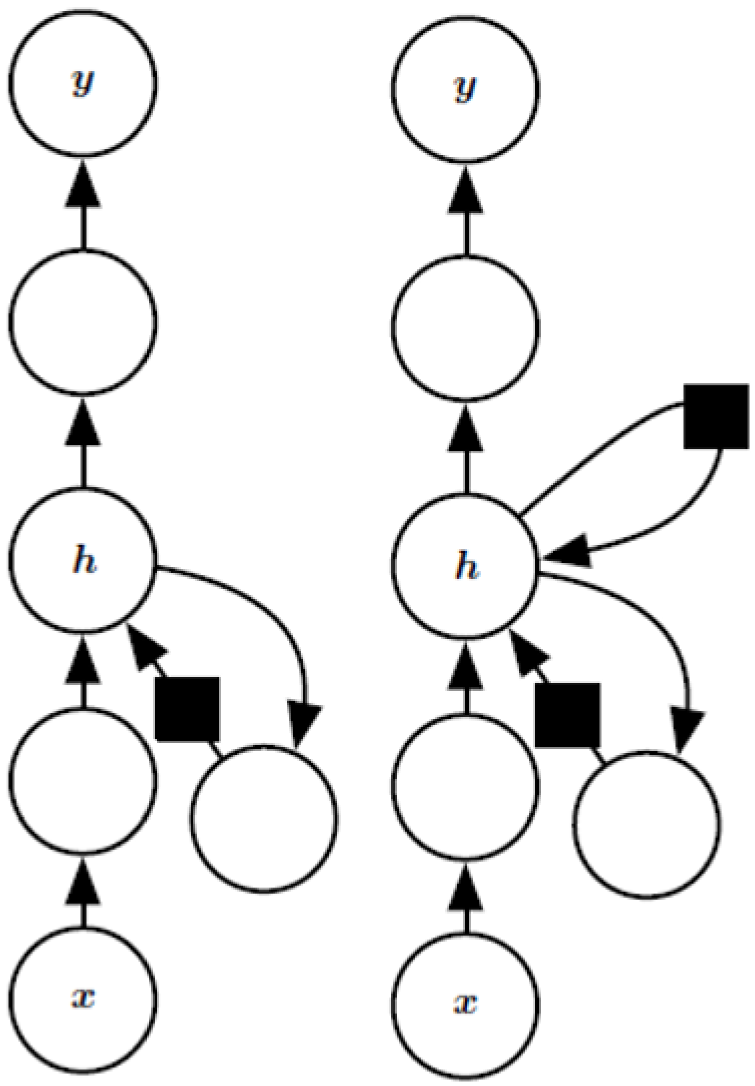
  如下所示，隐状态有两层： $\vec{\mathbf{h}}^{(t)}$ 和 $\vec{\mathbf{z}}^{(t)}$ 。隐状态层中，层次越高，对输入表达的概念越抽象。

  ![more_hiden]

- 第二种引入深度的方式：可以在这三种变换中，各自使用一个独立的 `MLP` （可能是深度的）。如下左图所示。

  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。

  解决方法是：类似 `ResNet` 的思想，在 “隐状态-隐状态” 的路径中引入跳跃连接即可缓解该问题。如下右图所示。



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。

2. `RNN` 涉及到许多相同函数的多次组合，每个时间步一次。这种组合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。
3. 考虑一个非常简单的循环结构（没有非线性，没有偏置）：

$$\vec{\mathbf{h}}^{(t)}=\mathbf{W}\vec{\mathbf{h}}^{(t-1)}$$

则有：

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

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

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

当 $\mathbf{W}$ 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 $w^{(t)}$ （初值为 1）。假设 $w^{(t)}$ 是随机生成、各自独立、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：设计工作在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遥远过去的信息

3.2.1 跳跃连接

1. 增加从遥远过去的变量到当前变量的直接连接，是得到粗粒度时间尺度的一种方法。
    - 普通的`RNN`中，循环从时刻 t 单元连接到了时刻 $t+1$ 单元。
    - 跳跃连接会增加一条从时刻 t 到时刻 $t+d$ 单元的连接。
2. 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。

3.2.2 删除连接

1. 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它们。
2. 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运作；而跳跃连接可以选择在长时间尺度上运作，也可以在短时间尺度上运作。

3.2.3 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。
2. 一个线性自连接的例子：（其中 $\mu^{(t)}$ 为隐单元， $v^{(t)}$ 为输入）
$$\mu^{(t)}=\alpha\mu^{(t-1)}+(1-\alpha)v^{(t)}$$
    - 当 α 接近1 时， $\mu^{(t)}$ 能记住过去很长一段时间的输入信息
    - 当 α 接近 0 时，$\mu^{(t)}$ 只能记住当前的输入信息。
3. 线性自连接的隐单元拥有类似 $\mu^{(t)}$ 的行为。这种隐单元称作渗漏单元。
4. 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先前的 d 个时间步的输入值所影响。
    - 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。
5. 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。
6. 强制不同的循环单元组在不同时间尺度上运作，有以下方法：
    - 让循环单元变成渗漏单元，但是不同的单元组有不同的、固定的 α 值。

- 在梯度下降的参数更新中，显式使得不同单元组的参数有不同的更新频率。

3.3 梯度截断

1. 对于长期依赖问题中的梯度爆炸，常用的解决方案是：梯度截断。
2. 梯度截断有两种方案：（假设 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。

3. 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。
4. 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。
5. 逐元素的梯度截断，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。

3.4 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：目标是希望梯度向量 $\nabla_{\vec{\mathbf{h}}^{(t)}}L$ 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：可以将反向传播向量 $\nabla_{\vec{\mathbf{h}}^{(t)}} L$ 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。

  如果没有梯度截断，则梯度爆炸会阻碍学习的成功。

3. 正则化引导信息流的一个主要缺点是：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、序列到序列架构

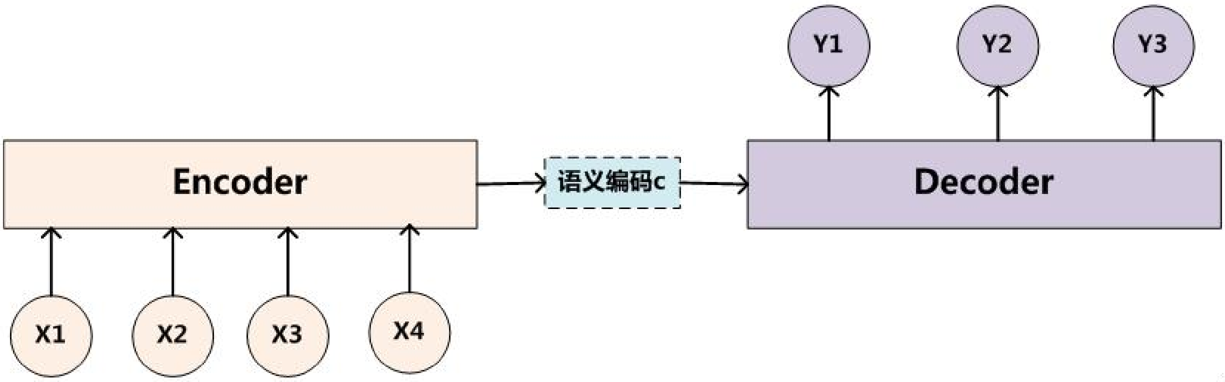
1. 采用`RNN` 的序列到序列结构主要有三种类型：
   - 将输入序列映射成固定大小的向量。即：多对一。如 `单输出&隐-隐连接 RNN`
   - 将固定大小的向量映射成一个序列。即：一对多。如`单输入多输出&隐-隐连接 RNN` 和 `单输入多输出&输出-隐连接 RNN` 。
   - 将一个输入序列映射到一个等长的输出序列。即：等长对等长。如： `多输出&隐-隐连接RNN` 和 `多输出&输出-隐连接RNN` 。
   - 将一个输入序列映射到一个不等长的输出序列。这通常采取“编码-解码”架构。

     如：语音识别、机器翻译、知识问答等任务都是不等长的映射任务。

4.1 编码-解码架构

1. 将`RNN`的输入序列 $\{\vec{\mathbf{x}}^{(1)},\vec{\mathbf{x}}^{(2)},\cdots,\vec{\mathbf{x}}^{(\tau_x)}\}$ 称作上下文，令 `C` 为该上下文的一个表达`representation` 。
   - `C` 可能是一个向量，或者一个向量序列。

     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。
   - `C` 需要提取输入序列 $\{\vec{\mathbf{x}}^{(1)},\vec{\mathbf{x}}^{(2)},\cdots,\vec{\mathbf{x}}^{(\tau_x)}\}$ 的有效信息
2. 编码-解码架构：



- 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` 。此时为“单长度输入序列”。

  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 $\vec{\mathbf{h}}^{(0)}$ ，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：

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

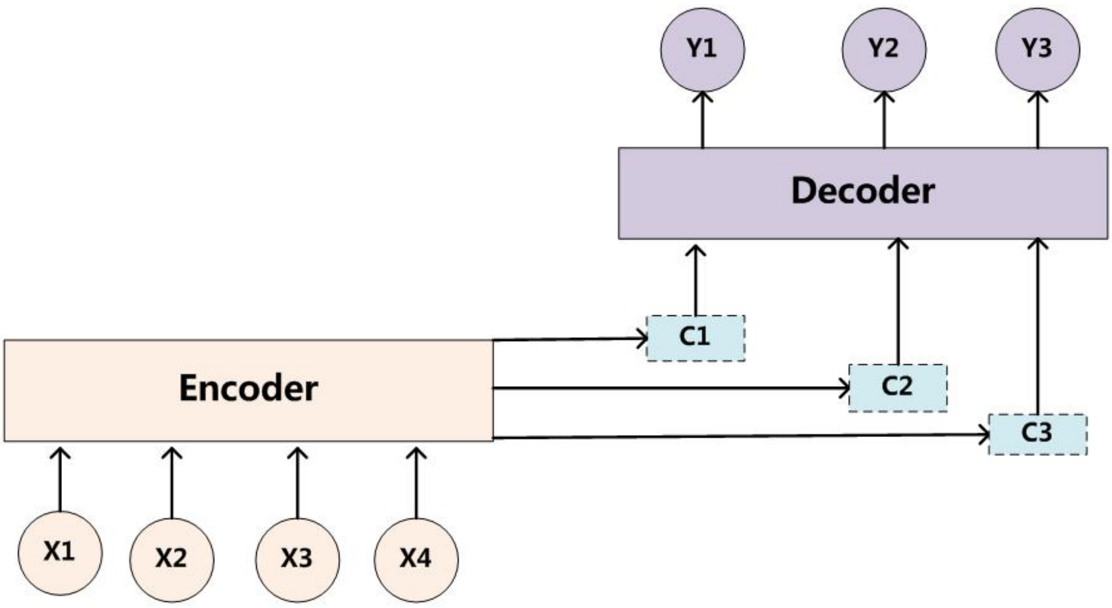
3. 编码-解码架构的创新之处在于：输入序列长度 τ_x 和输出序列长度 τ_y 可以不同。

   前面介绍的 `多输出&隐-隐连接RNN` 和 `多输出&输出-隐连接RNN` 均要求 $\tau_x = \tau_y$ 。

4. 编码-解码架构中，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。

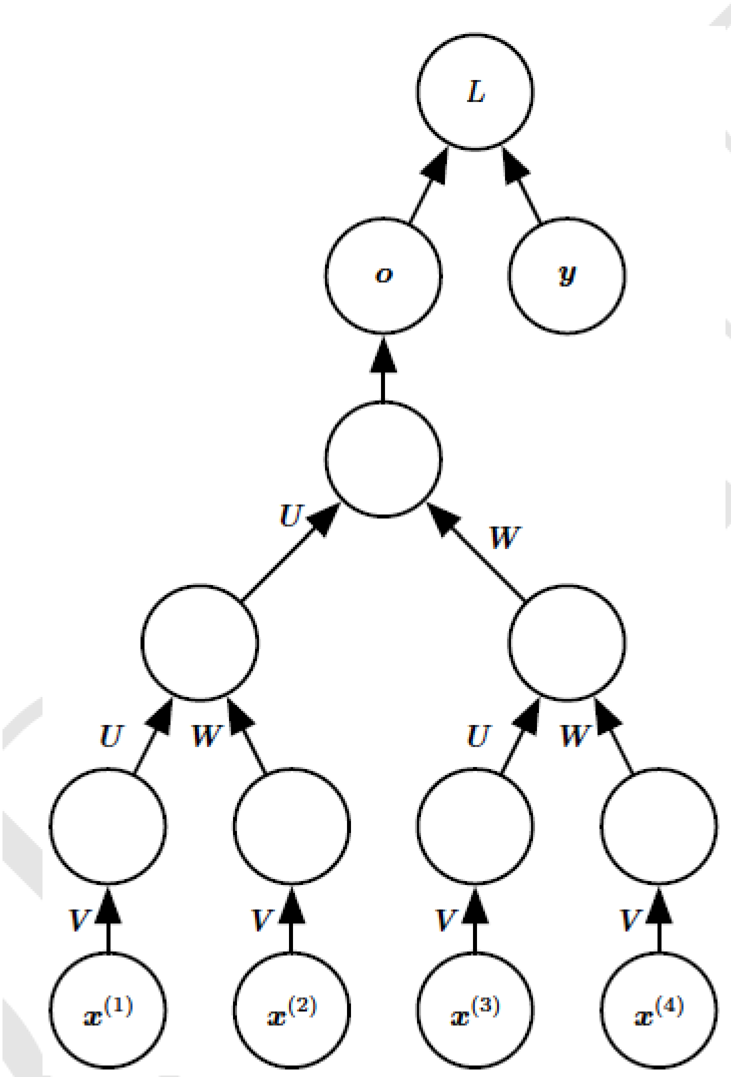
5. 编码-解码架构的主要缺点：编码器`RNN`输出的上下文`C`的维度太小，难以适当的概括一个长的输入序列。

   可以让`C`也成为一个可变长度的序列，同时引入将序列`C`的元素和输出序列元素相关联的`attention`机制。



五、递归神经网络

1. 递归神经网络是循环神经网络的另一个扩展：它被构造为深的树状而不是链状，具有一种不同类型的计算图。



2. 递归神经网络潜在用途是学习推论，目前成功用于输入是树状结构的神经网络，如：自然语言处理，计算机视觉。
3. 递归神经网络的显著优势：对于长度为 τ 的序列，网络深度可以急剧的从 τ 降低到 $O(\log\tau)$。 这有助于解决长期依赖。
4. 递归神经网络的一个悬而未决的难题是：如何以最佳的方式构造树。
   - 一种选择是：构造一个不依赖于数据的树。如：平衡二叉树。
   - 另一种选择是：可以借鉴某些外部方法。如：处理自然语言时，将句子的语法分析树作为递归网络的树。

六、回声状态网络

6.1 动机

1. 从 $\vec{\mathbf{h}}^{(t-1)}$ 到 $\vec{\mathbf{h}}^{(t)}$ 的循环映射权重，以及从 $\vec{\mathbf{x}}^{(t)}$ 到 $\vec{\mathbf{h}}^{(t)}$ 的输入映射权重是`RNN`中最难学习的参数。

   一种解决方案是：设定隐单元，使得它能够很好地捕捉过去输入历史，并且只学习输出权重。这就是回声状态网络 `echo state networkd:ESN` 。

- 隐单元形成了捕获输入历史所有可能的、不同方面的临时特征池。
- 流体状态机`liquid state machine`也分别独立地提出了这种想法

2. 回声状态网络和流体状态机都被称作储层计算`reservoir computing`。储层计算`RNN`类似于支持向量机的核技巧：

   - 先将任意长度的序列（到时刻 t 的输入历史）映射为一个长度固定的向量 (循环状态 $\vec{\mathbf{h}}^{(t)}$)
   - 然后对 $\vec{\mathbf{h}}^{(t)}$ 施加一个线性预测算子以解决感兴趣的问题。

     线性预测算子通常是一个线性回归，此时损失函数就是均方误差。

3. 储层计算`RNN`的目标函数很容易设计为输出权重的凸函数，因此很容易用简单的学习算法解决。

6.2 原理

1. 储层计算`RNN`的核心问题：如何将任意长度的序列（到时刻 t 的输入历史）映射为一个长度固定的向量 (循环状态 $\vec{\mathbf{h}}^{(t)}$)？

   解决方案是：将网络视作动态系统，并且让动态系统接近稳定边缘，此时可以推导出满足条件的循环状态 $\vec{\mathbf{h}}^{(t)}$。

2. 考虑反向传播中，雅克比矩阵 $\mathbf{J}^{(t)}=\frac{\partial\vec{\mathbf{s}}^{(t)}}{\partial\vec{\mathbf{s}}^{(t-1)}}$。 定义其谱半径为特征值的最大绝对值。

   假设网络是纯线性的，此时 $\mathbf{J}^{(t)}=\mathbf{J}$ 。假设 $\mathbf{J}$ 特征值 λ 对应的单位特征向量为 $\vec{\mathbf{v}}$ 。

   设刚开始的梯度为 $\vec{\mathbf{g}}$，经过 n 步反向传播之后，梯度为 $\mathbf{J}^n\vec{\mathbf{g}}$ 。

   假设开始的梯度有一个扰动，为 $\vec{\mathbf{g}}+\delta\vec{\mathbf{v}}$，经过 n 步反向传播之后，梯度为 $\mathbf{J}^n\vec{\mathbf{g}}+\delta\mathbf{J}^n\vec{\mathbf{v}}$ 。则这个扰动在 n 步之后放大了 $|\lambda|^n$ 倍。

   - 当 $|\lambda|>1$ 时，偏差 $\delta|\lambda|^n$ 就会指数增长
   - 当 $|\lambda|<1$ 时，偏差 $\delta|\lambda|^n$ 就会指数衰减

3. 在正向传播的情况下，$\mathbf{W}$ 定义了信息如何前向传播；在反向传播的情况下，$\mathbf{J}$ 定义了梯度如何后向传播

   - 当`RNN` 是线性的情况时，二者相等。
   - 当引入压缩非线性时（如 `sigmoid/tanh`）：
     - 此时 $\mathbf{h}^{(t)}$ 有界，即前向传播有界。
     - 此时梯度仍然可能无界，即反向传播无界。
   - 在神经网络中，反向传播更重要。因为需要使用梯度下降法求解参数！

4. 回声状态`RNN` 的策略是：简单地固定权重，使其具有一定的谱半径（比如 3）。

   - 在信息前向传播过程中，由于饱和非线性单元的稳定作用，$\mathbf{h}^{(t)}$ 不会爆炸。
   - 在信息反向传播过程中，非线性的导数将在许多个时间步之后接近 0，梯度也不会发生爆炸。

七、LSTM 和其他门控RNN

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` 循环网络、和基于门控循环单元`GRU: gated recurrent unit`循环网络。

2. 门控`RNN` 的思路和渗漏单元一样：生成通过时间的路径，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。
   - 可以使用参数化的连接权重来实现这个目的。如渗漏单元。
   - 门控`RNN` 将其推广为：连接权重在每个时间步都可能改变。

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. 围绕门控 `RNN` 这一主题可以设计更多的变种。

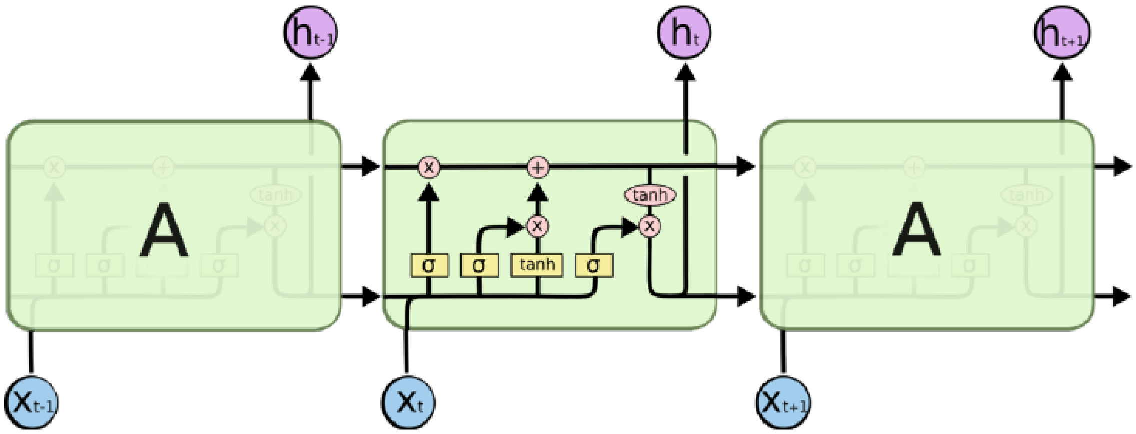
   然而一些调查发现：这些 `LSTM` 和 `GRU` 架构的变种，在广泛的任务中难以明显的同时击败这两个原始架构。

7.1 LSTM

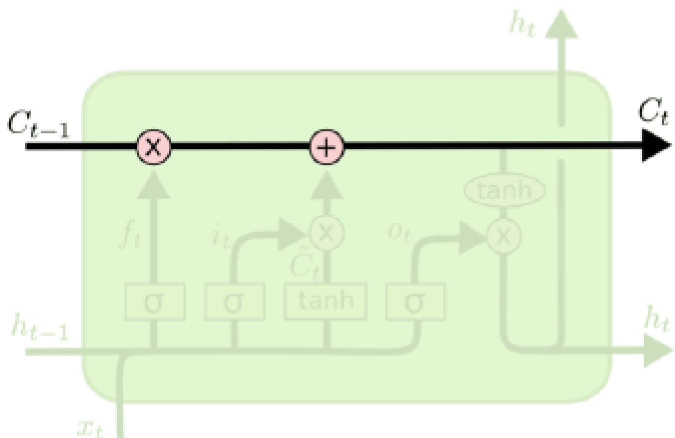
1. `LSTM` 在手写识别、语音识别、机器翻译、为图像生成标题等领域获得重大成功。
2. `LSTM` 引入 `ceil` 循环以保持梯度长时间持续流动。其中一个关键是：`ceil` 循环的权重视上下文而定，而不是固定的。

   具体做法是：通过 `gate` 来控制这个 `ceil` 循环的权重，而这个 `gate` 由上下文决定。
3. `LSTM` 循环网络除了外部的 `RNN` 循环之外，还有内部的 `LSTM ceil` 循环（自环）。

   `LSTM` 的 `ceil` 代替了普通 `RNN` 的隐单元。而 `LSTM` 的 $\vec{\mathbf{h}}^{(t)}$ 是 `ceil` 的一个输出。



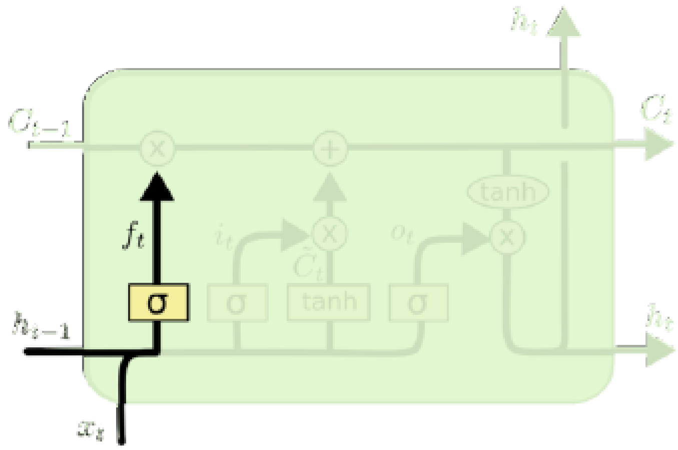
4. `LSTM` 最重要的就是 `ceil` 状态 $\vec{\mathbf{C}}^{(t)}$ ，它以水平线在图上方贯穿运行。



5. `sigmoid` 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
   - LSTM 拥有三个门：
6. `LSTM` 拥有三个门：遗忘门、输入门、输出门。

7.1.1 遗忘门

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



2. 与渗漏单元类似，`LSTM ceil` 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)

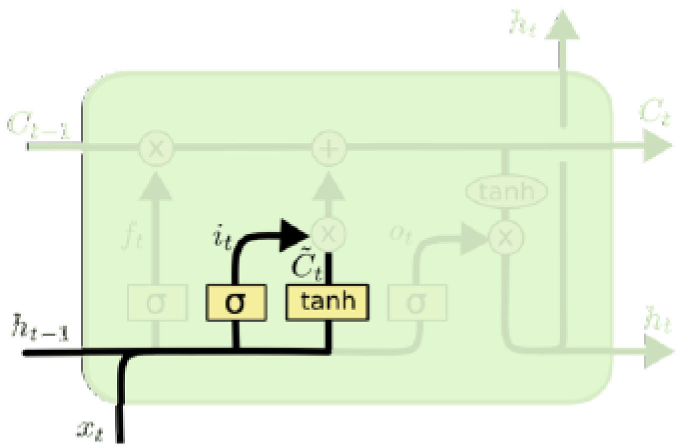
$$\vec{\mathbf{f}}^{(t)} = \sigma(\vec{\mathbf{b}}^f + \mathbf{U}^f \vec{\mathbf{x}}^{(t)} + \mathbf{W}^f \vec{\mathbf{h}}^{(t-1)})$$

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

7.1.2 输入门

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



2. 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 $\vec{\mathbf{g}}^{(t)}$

3. `ceil` 状态更新：`ceil`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`ceil` 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)

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

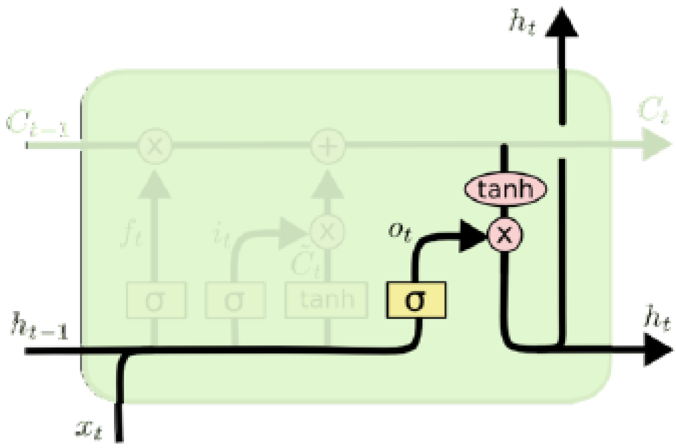
   其中：

   - $\vec{\mathbf{b}}$ 为`ceil`的偏置
   - $\mathbf{U}$ 为`ceil`的输入权重
   - $\mathbf{W}$ 为`ceil`的循环权重

7.1.3 输出门

1. 输出门控制了`ceil` 状态 $\vec{\mathbf{C}}^{(t)}$ 中，有多少会进入`ceil` 输出。

   - 注意：是`ceil` 输出。它就是 $\vec{\mathbf{h}}^{(t)}$，而不是整个`RNN` 单元的输出 $\vec{\mathbf{o}}^{(t)}$ 。
   - `ceil` 之间的连接是通过 $\vec{\mathbf{h}}^{(t)},\vec{\mathbf{C}}^{(t)}$ 来连接的。



2. 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

3. `ceil` 输出更新：`ceil` 输出就是 $\vec{\mathbf{h}}^{(t)}$。它是将 `ceil` 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)

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

4. 一旦得到了 `ceil` 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。

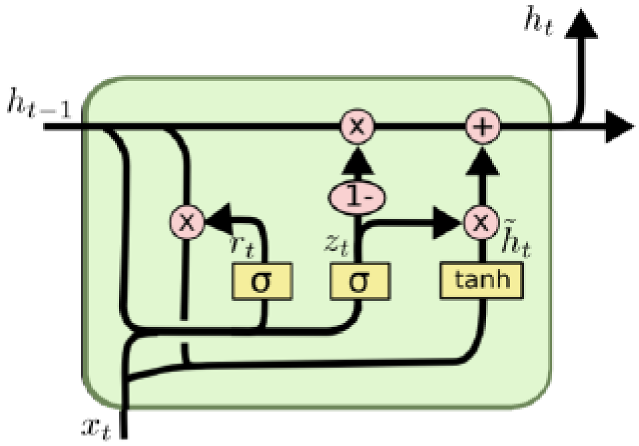
5. 也可以选择使用 `ceil` 状态 $\vec{\mathbf{C}}^{(t)}$ 作为这些门的额外输入。此时 $\vec{\mathbf{f}}^{(t)}, \vec{\mathbf{g}}^{(t)}, \vec{\mathbf{q}}^{(t)}$ 就多了额外的权重参数，这些参数对应于 $\vec{\mathbf{C}}^{(t-1)}$ 的权重和偏置。

7.2 GRU

1. 门控循环单元 `GRU` 与 `LSTM` 的主要区别是：

   - `GRU` 的单个门控单元同时作为遗忘门和输入门
   - `GRU` 不再区分 `ceil` 的状态 $\vec{\mathbf{C}}$ 和 `ceil` 的输出 $\vec{\mathbf{h}}$

   最终使得 `GRU` 要比 `LSTM` 模型更简单。



2. `GRU` 中有两个门：更新门、复位门。

7.2.1 更新门

1. 更新门决定了：

   - 根据当前的$\vec{\mathbf{x}}^{(\mathbf{t})}, \vec{\mathbf{h}}^{(\mathbf{t-1})}$ 得到的 $\tilde{\vec{\mathbf{h}}}^{(t)}$ 中，有多少信息进入了 $\vec{\mathbf{h}}^{(t)}$ 。

     即：新的更新中，有多少会进入结果。

   - $\vec{\mathbf{h}}^{(t-1)}$ 中，有多少信息进入 $\vec{\mathbf{h}}^{(t)}$ 。

     即：旧的结果中，有多少会进入新的结果。

     > $1-z$ 可以理解为保留门。

2. 更新门的更新公式为：

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

   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

7.2.2 复位门

1. 复位门决定了：在获取 $\tilde{\vec{\mathbf{h}}}^{(t)}$ 的过程中， $\vec{\mathbf{x}}^{(t)}, \vec{\mathbf{h}}^{(t-1)}$ 之间的比例。

   可以理解为：新的更新中，旧的结果多大程度上影响新的更新。如果 $r=0$ ，则旧的结果不影响新的更新，可以理解为复位。

2. 复位门的更新公式为：

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

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

7.2.3 输出更新

1. `GRU` 的 `ceil` 输出更新公式：

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

   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）

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

   其中：

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

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

   - $\vec{\mathbf{b}}$ 为 `ceil` 的偏置
   - $\mathbf{U}$ 为 `ceil` 的输入权重
   - $\mathbf{W}$ 为 `ceil` 的循环权重

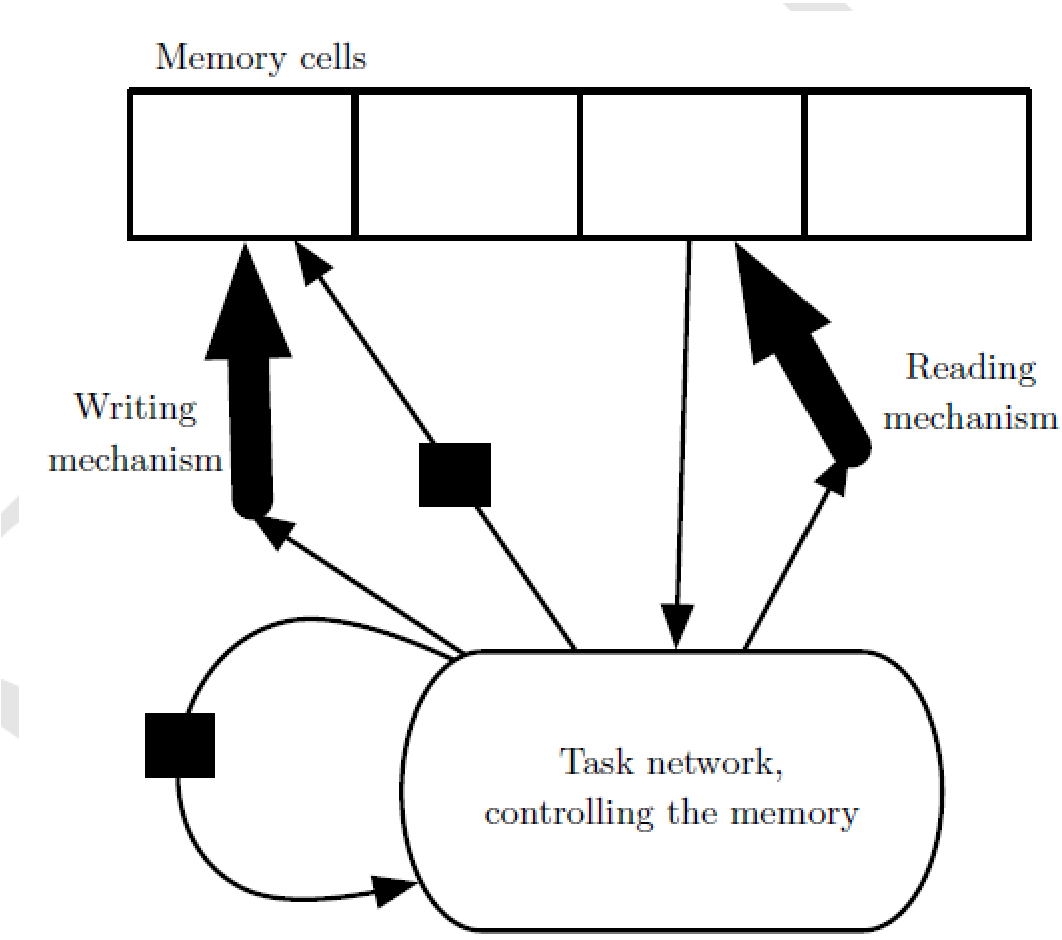
2. 在 `LSTM` 与 `GRU` 中有两种非线性函数： `sigmoid` 与 `tanh`
   - `sigmoid` 用于各种门，是因为它的阈值为 0~1，可以很好的模拟开关的关闭程度。
   - `tanh` 用于激活函数，是因为它的阈值为 -1~1，它的梯度的阈值为 0~1。
     - 如果使用 `sigmoid` 作为激活函数，则其梯度范围为 0~0.5，容易发生梯度消失。
     - 如果使用 `relu` 作为激活函数，则前向传播时，信息容易爆炸性增长。

       另外 `relu` 激活函数也会使得输出只有大于等于 `0` 的部分。

八、外显记忆

1. 神经网络擅长存储隐性知识，但是它们很难记住事实。这是因为神经网络缺乏工作记忆系统。
   - 隐性知识：难以用语言表达的知识。如：狗和猫有什么不同。
   - 事实：可以用知识明确表达的。如：猫是一种动物，十月一日是国庆节。
   - 工作记忆系统：类似人类为了实现一些目标而明确保存、操作相关信息片段的系统。

     它不仅能让系统快速的存储和检索具体的事实，还能用于进行循序推理。
2. `Weston et al.` 引入了记忆网络，其种包括一组可以通过寻址机制来访问的记忆单元。



- 记忆网络需要监督信号来指示它们：如何使用记忆单元。

  但是`Graves et al.`引入了神经网络图灵机(`Neural Turing machine`:`NTM`):

  - 它不需要明确的监督指示来学习如何从记忆单元读写任意内容
  - 它可以通过使用基于内容的软注意机制来执行端到端的训练
- 每个记忆单元都可以被认为是 `LSTM` 和 `GRU` 中记忆单元的扩展。不同的是：`NTM` 网络输出一个内部状态来选择从哪个单元读取或者输入。
- 由于产生整数地址的函数非常难以优化，因此`NTM` 实际上同时从多个记忆单元写入或者读取

  - 在读取时，`NTM`采取多个单元的加权平均值
  - 在写入时，`NTM`同时修改多个单元的值

  加权的系数由一系列产生小数值的单元生成（通常采用`softmax`函数产生）。
- 这些记忆单元通常扩充为包含向量，而不是由 `LSTM` 或者 `GRU` 记忆单元所存储的单个标量。原因有两个：

  - 降低读取单个记忆数值的成本。
  - 允许基于内容的寻址。

3. 如果一个存储单元的内容在大多数时间步上会被复制（不被忘记），那么：

   - 它所包含的的信息可以在时间上前向传播。
   - 在时间上反向传播的梯度也不会消失或者爆炸。
4. 外显记忆似乎允许模型学习普通 `RNN` 或者 `LSTM`不能学习的任务。

   这种优势的一个原因可能是：信息和梯度能够在非常长的持续时间内传播。