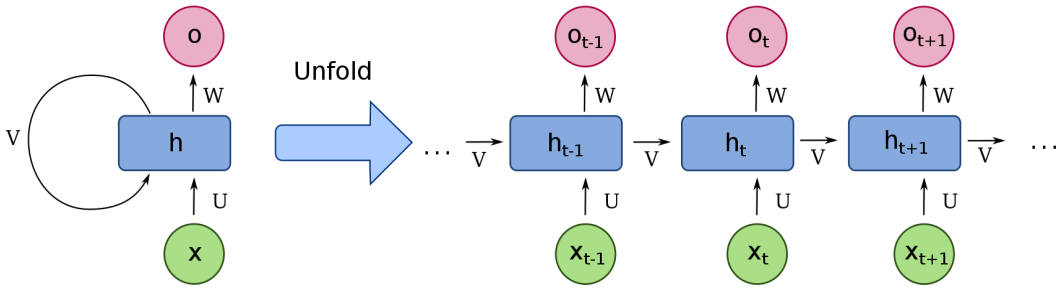
Chapter-9 RNNs and LSTMs

1. *Language is an inherently temporal phenomenon*

Language always relates to temporal nature because spoken language is a sequence of continuous signals.

We intuitively use language processing algorithms like HMM, Viterbi and N-gram etc for its temporal attribute.

9.1 Recurrent Neural Networks



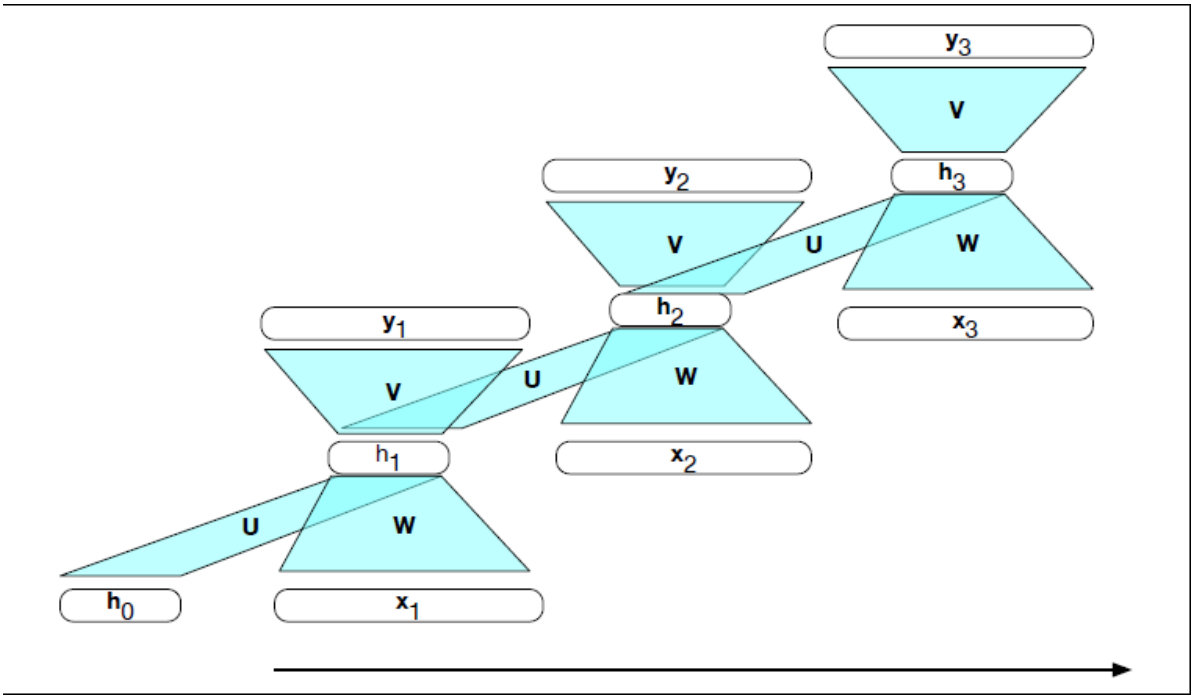
对于RNN网络来说，对于$step = t$为了考虑context或者说之前时间步X所蕴含的信息来影响后续输出的O_t，网络中同时储存了h_{t-1}作为蕴含了前文信息的向量：

$$
\begin{aligned}
h_t &= g(Uh_{t-1} + Wx_t) \\
y_t &= f(Vh_t)
\end{aligned}
$$

其中U是方阵、$U \in \mathbb{R}^{d_h \times d_h}$

9.1.1 Inference in RNNS

对于RNN的前向传播来说，想要计算某一时刻t的输出必须得到$t-1$时刻以及之前的隐藏层的输入，推理的时间复杂度为$\mathcal{O}(n)$



9.1.2 training in RNNS

specific the following units:

`training set` - `loss function` - `backpropagation for gradient descent`

对于可训练的参数有W, U, V

值得注意的是：

对于涉及更长的输入序列的应用，如语音识别、字符级处理，或流式连续输入，展开整个输入序列可能不可行。在这些情况下，我们可以将输入分解为可处理的固定长度的片段，并将每个片段作为一个不同的训练单元。

9.2 RNNs as Language Models

将RNN网络应用于语言建模，例如：对于给定的上文计算下一个词的概率，对整个vocabulary建立概率模型

$$P(fish|Thanks\ for\ all\ the)$$

$$P(w_{1:n}) = \prod_{i=1}^{n} P(w_i|w_{<i})$$

由于RNN网络的step by step特性，无需考虑n-gram模型的马尔科夫链假设，相较之前提出的FFN网络有着巨大优势

9.2.1 forward inference in RNNs

ex：

对于给定的输入序列$X = [x_1; ..; x_t; .. x_N]$，其中任意一项为one-hot向量，$dim = |V|$, 输出是y，是一份概率向量表示最终的下一个词汇的概率分布。

$$
\begin{aligned}
\mathbf{e}_t &= \mathbf{E}\mathbf{x}_t \\
\mathbf{h}_t &= g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{e}_t) \\
\mathbf{y}_t &= \text{softmax}(\mathbf{V}\mathbf{h}_t)
\end{aligned}
$$

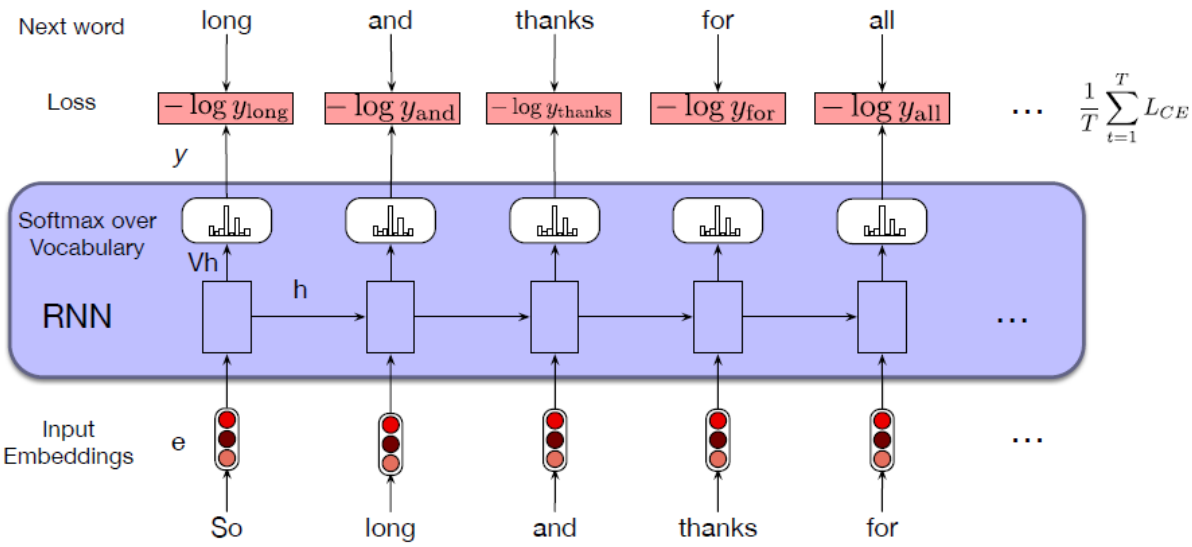
在RNNs modeling下：

$$P(w_{1:n}) = \prod \mathrm{y}_i[\mathrm{w}_i]$$

9.2.2 training in RNNs

对于下一个词的**多分类**任务，比较自然的想到了交叉熵作为损失函数：

$$L_{CE} = -\sum_{w \in V} \mathrm{y}_t[w] \log \hat{\mathrm{y}}[w]$$




```
RNN = torch.nn.RNN(input_size, output_size, layer_nums)
#input=[b, sq_len, vector_dim]
#h0 = [layer_num, sq_len, out_dim]
outp, hidden_states = RNN(input, h0)
loss = nn.CrossEntropyLoss(outp[idx], y)
```

This idea that we always give the model the correct history sequence to predict the next word (rather than feeding the model its best case from the previous time step) is called teacher forcing.

对于RNN网络建立语言模型，我们在训练过程中有两种策略：

1. 对于每次计算完$Loss$后**修正当前的输入为正确**的$token$然后在进入下一次的循环中（**teacher forcing)**：方法用于解决在推理过程下因为RNN网络生成的y_t会在当前序列前向计算后产生累计误差

例如想生成一句话：

```
<S> 我 非常 喜欢 吃 饺子 和 馄饨 </S>
```

在训练时：

```
0. 输入                    输出          Ground Truth
1. <S>                     我            我
2. <S> 我                  ？            非常
3. <S> 我 非常             ？            喜欢
4. <S> 我 非常 喜欢        ？            吃
```

2. 对对于每次计算完$Loss$后将本次循环的不使用ground truth，而是用RNN生成的$token$作为下一次的输出去建立语言模型

```
0. 输入                    输出          Ground Truth
1. <S>                     你            我
2. 你                      咋            非常
3. 咋                      ？            喜欢
4. ？                      ？            吃
...
```

分析：不使用*TF*，loss往往非常大，无法使模型收敛，只使用*TF*容易造成是模型的生成能力下降（自回归性质），在使用过程中往往使用随机策略混合TF

RNN反向传播式子形如：

$$a_t = b_t + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} c_j \right) b_i.$$

完全计算

显然，我们可以仅仅计算 (8.7.7)中的全部总和，然而，这样的计算非常缓慢，并且可能会发生梯度爆炸，因为初始条件的微小变化就可能会对结果产生巨大的影响。也就是说，我们可以观察到类似于蝴蝶效应的现象，即初始条件的很小变化就会导致结果发生不成比例的变化。这对于我们想要估计的模型而言是非常不可取的。毕竟，我们正在寻找的是能够很好地泛化高稳定性模型的估计器。因此，在实践中，这种方法几乎从未使用过。

截断时间步

或者，我们可以在τ步后截断 (8.7.7)中的求和计算。这是我们到目前为止一直在讨论的内容，例如在 8.5节中分离梯度时。这会带来真实梯度的近似，只需将求和终止为$\partial h_{t-\tau}/\partial w_h$。在实践中，这种方式工作得很好。它通常被称为截断的通过时间反向传播 [Jaeger, 2002]。这样做导致该模型主要侧重于短期影响，而不是长期影响。这在现实中是可取的，因为它会将估计值偏向更简单和更稳定的模型。

随机截断

最后，我们可以用一个随机变量替换$\partial h_t/\partial w_h$，该随机变量在预期中是正确的，但是会截断序列。这个随机变量是通过使用序列ξ_t来实现的，序列预定义了$0 \leq \pi_t \leq 1$，其中$P(\xi_t = 0) = 1 - \pi_t$且$P(\xi_t = \pi_t^{-1}) = \pi_t$，因此$E[\xi_t] = 1$。我们使用它来替换 (8.7.4)中的梯度$\partial h_t/\partial w_h$得到：

$$z_t = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \xi_t \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial w_h}. \tag{8.7.8}$$

从ξ_t的定义中推导出来$E[z_t] = \partial h_t/\partial w_h$。每当$\xi_t = 0$时，递归计算终止在这个$t$时间步。这导致了不同长度序列的加权和，其中长序列出现的很少，所以将适当地加大权重。这个想法是由塔莱克和奥利维尔 [Tallec & Ollivier, 2017]提出的。

9.2.3 Weight Tying

对于之前的公式

$$\begin{aligned} h_t &= g(Uh_{t-1} + Wx_t) \\ y_t &= f(Vh_t) \end{aligned}$$

我们发现在语言建模过程中理解为:

$$\begin{aligned} e_t &= Ex_t \\ h_t &= g(Uh_{t-1} + We_t) \\ \hat{y} &= \text{softmax}(Vh_t) \end{aligned}$$

发现对于$(1)(3)$:

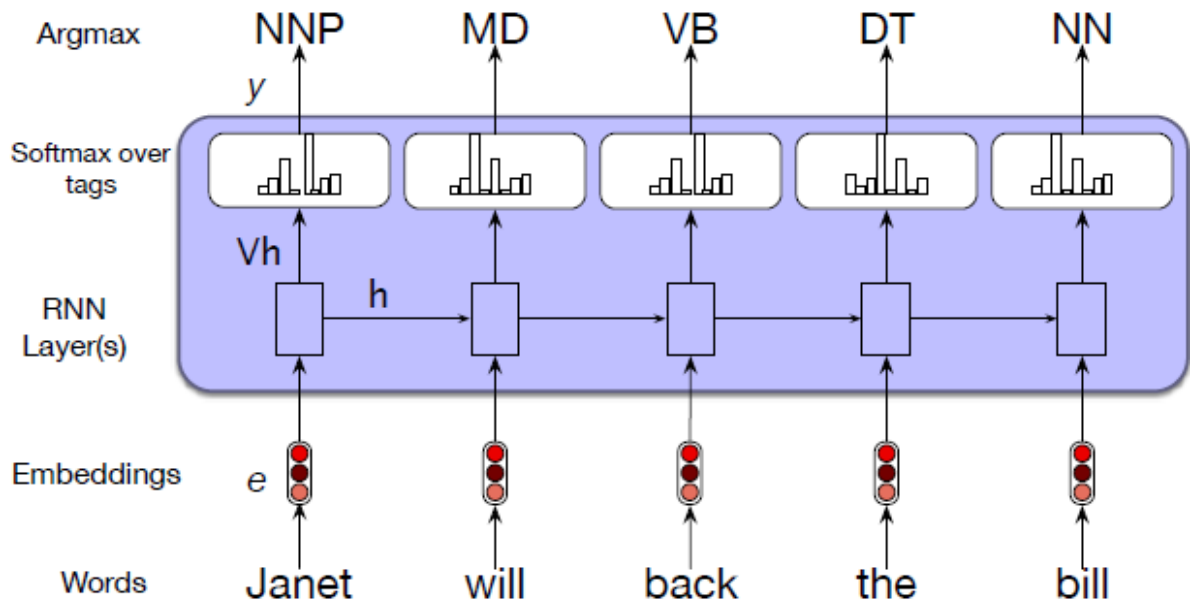
```
E.dim=[d_r, |V|]
e_t.dim=[d_r, 1]
h_t.dim=[d_hidden, 1]
y_hat -> similarity cal()
```

我们可以发现其实对V来说ht只需要找到词向量矩阵中最相似的向量即可：

于是我们将V替换成Embedding矩阵的转置E^T，这样不仅减少了可训练参数，并且更符合直觉

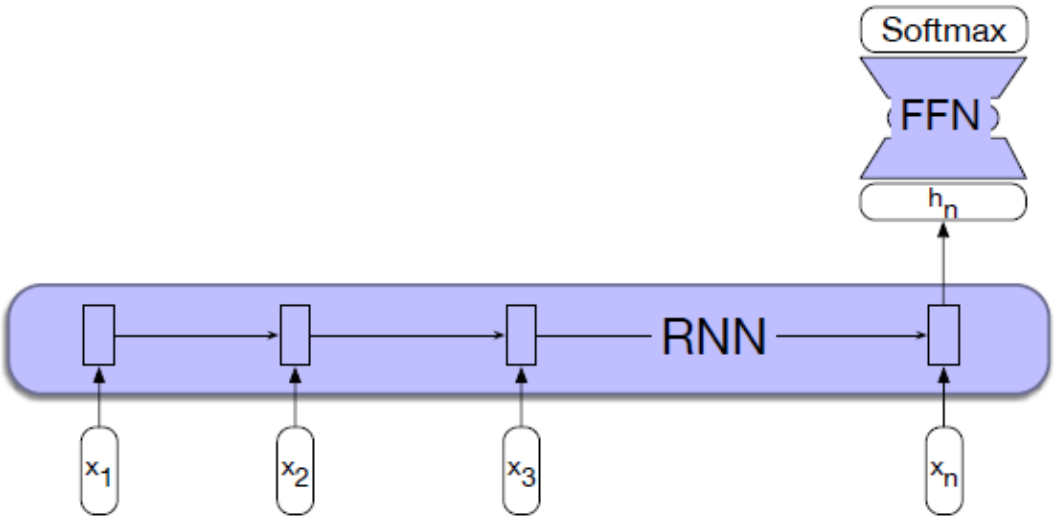
9.3 RNNs for other NLP tasks

9.3.1 Sequence Labeling



9.3.2 RNNs for Sequence Classification

对于句子情感分类或者对整个序列做分类任务RNN模块的范式如下所示：



分类时只对序列最后产生的h_n做*softmax*分类即可

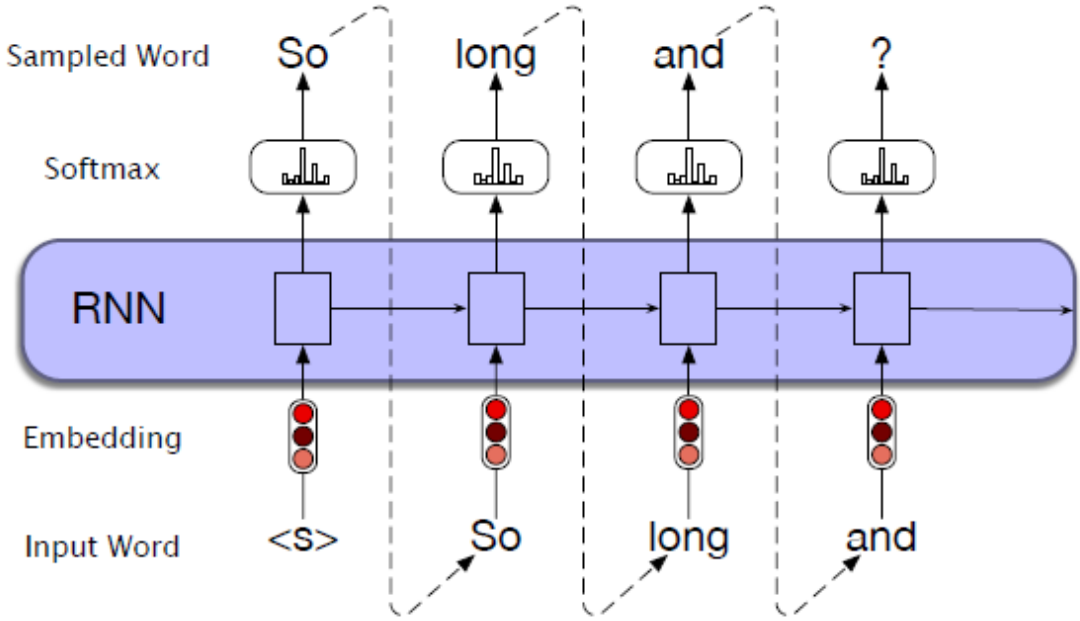
对于最后的h_n获取存在两种方式：

1. 取得最后根据t个step最终计算出的h_n 但是缺点是loss会一直累计传播到W, V, U矩阵中
2. 根据每个step的t做pooling，$h_n = \frac{1}{n}\sum h_i$

9.3.3 Generation with RNN-Based Language Models

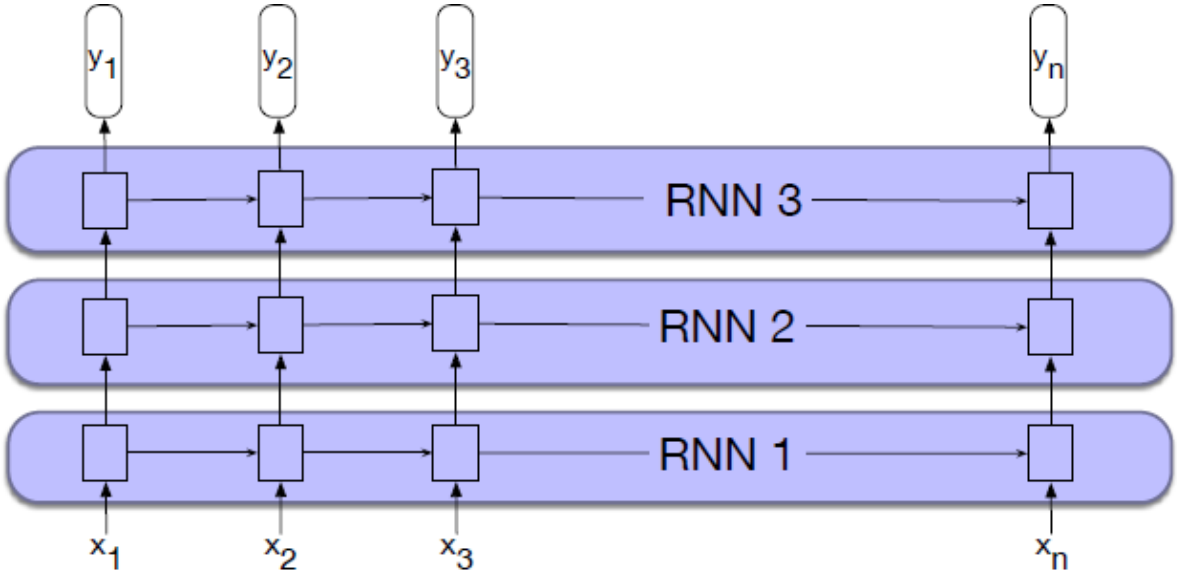
对于RNN式生成模型：

对于rnn模型来说有天然的自回归性质，在*LM generation*方面，对于给定的开始标签`<s>` 让RNN自由式的生成token



9.4 Stacked and Bidirectional RNN architectures

9.4.1 Stacked RNNs



根据不同的层数学习不同的潜在的状态时序信息(更抽象的信息)

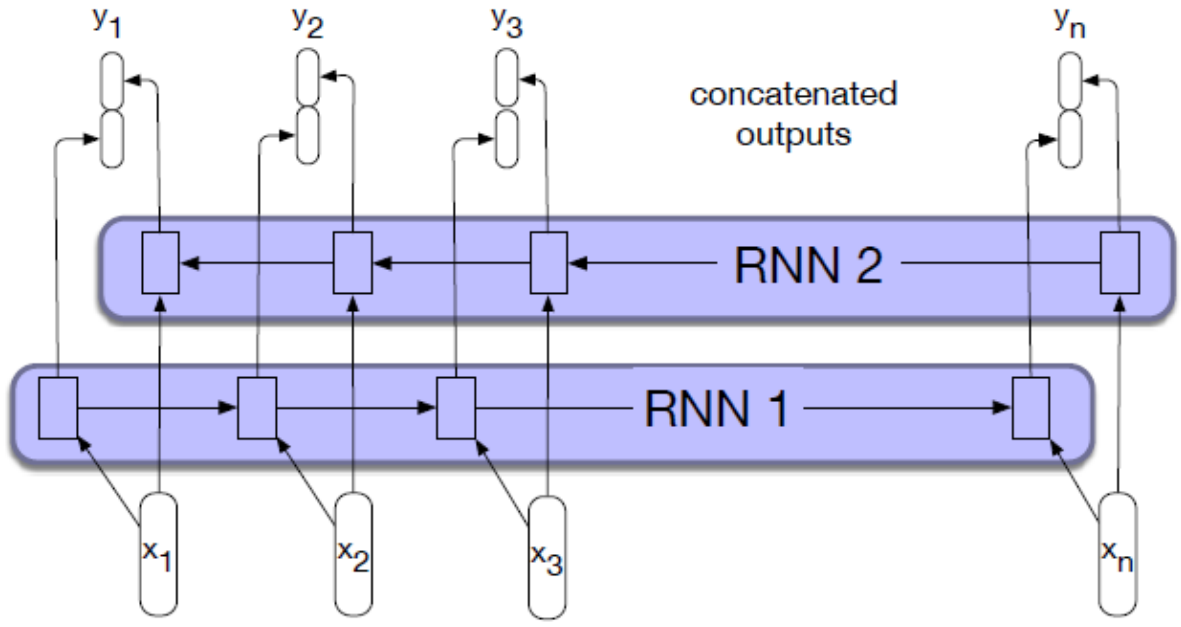
从直觉来看stacked RNNs outperform better的原因是想深层卷积网络一样可以获取时序信号中更加抽象的时序信息，但是对于RNNs的层数还是要具体情况具体分析；能力的提升也带来的计算开销和时间开销。

9.4.2 Bidirectional RNNs

original RNN只使用了单向的从人的角度来理解的prior context来做出下列的预测，但是在很多场景下我们可以通过整段序列的信息来做出预测，为了充分预测某个step下的h_t我们也可以使用上后序信息即为$token_{n=>t}$来获得新的逆向时序信号。

$$
\begin{aligned}
h_t^f &= \mathrm{RNN}_f(x1, \ldots, x_t) \\
h_t^b &= \mathrm{RNN}_b(x_t, \ldots, x_n) \\
h_t &= \mathrm{concat}[h_t^f; h_t^b]
\end{aligned}
$$

这就是双向RNN网络，*Bidirectional RNNs*



对于整个Sequence的整体预测变化：$softmax(\text{FFN}([h_t^f \bigoplus h_t^b]))$

9.5 LSTM

`The flights the airline was cancelling were full`

对于上述的句子来说对于airline to was是简单的，但是对于flights to were来说困难的，当前语境下flight，were距离远并且文中也设计单数的语境。

1. 对rnn来说第一个难点就是forward重要信息，因为U,W权重往往要兼顾两件事情，一是对当前状态做判断，二是对当前隐藏状态前向输出；
2. 当序列足够长的时候由于链式求导法则会使得RNN具有非常多的梯度相乘，不是梯度爆炸就是梯度消失~

综上原因提出的LSTM网络：

*long short-term memory*主要为了解决如下问题：对上下文管理问题变化成两个子问题：

1.移除不重要的信息

2.添加或者保持可能有用的信息

LSTM使用的方法是添加gate unit

1.forget gate 我该遗忘那些信息

$$\begin{aligned} \mathbf{f}_t &= \sigma(\mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{W}_f \mathbf{x}_t) \\ \mathbf{k}_t &= \mathbf{c}_{t-1} \odot \mathbf{f}_t \end{aligned}$$

2.needed infomation 我该记忆哪些信息

$$\mathbf{g}_t = \tanh(\mathbf{U}_g \mathbf{h}_{t-1} + \mathbf{W}_g \mathbf{x}_t)$$

3.add gate

$$\mathbf{i}_t = \sigma(\mathbf{U}_i\mathbf{h}_{t-1} + \mathbf{W}_i\mathbf{x}_t)$$
$$\mathbf{j}_t = \mathbf{g}_t \odot \mathbf{i}_t$$

4.merge infomation 本轮cell所暂留的信息

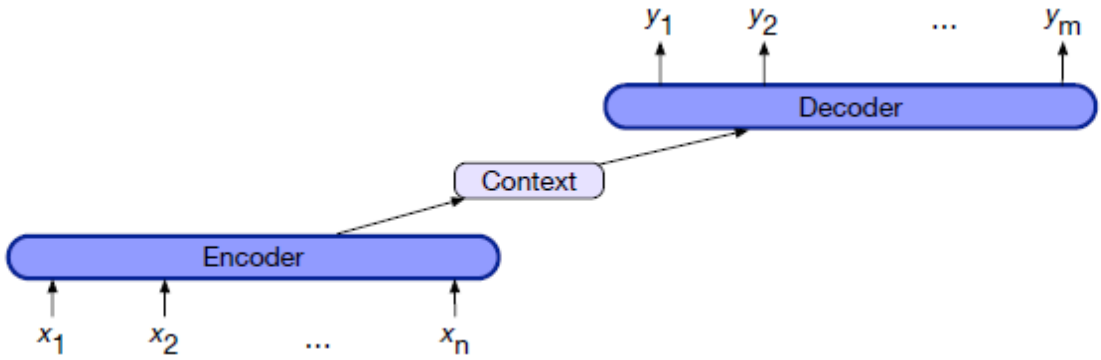
$$\mathbf{c}_t = \mathbf{j}_t + \mathbf{k}_t$$

5.最后获得隐藏层输出 信息

$$\mathbf{o}_t = \sigma(\mathbf{U}_o\mathbf{h}_{t-1} + \mathbf{W}_o\mathbf{x}_t)$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

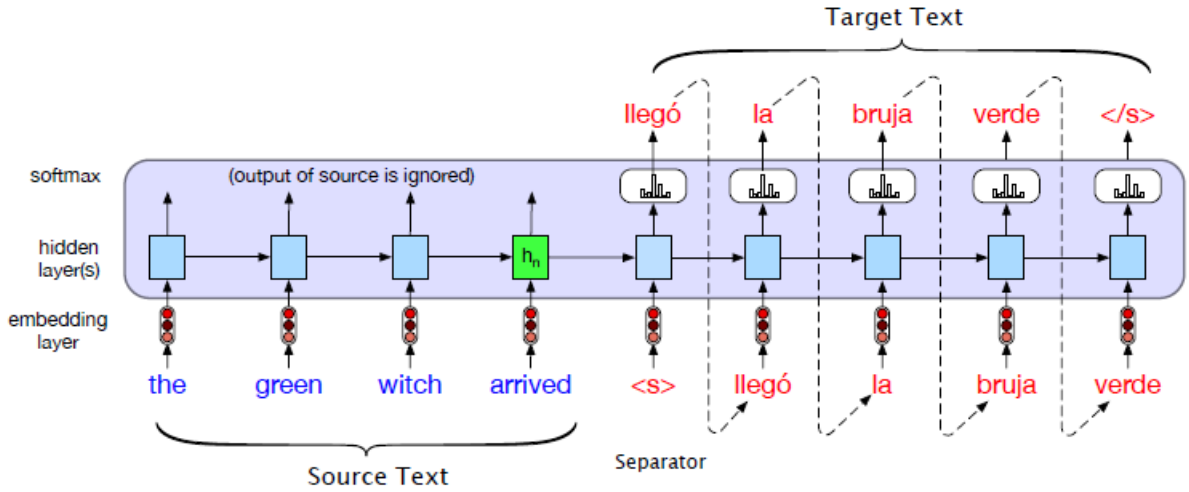
9.6 Summary: Common RNN NLP Architectures

RNN in encoder & decoder: machine translation && sequence2sequence



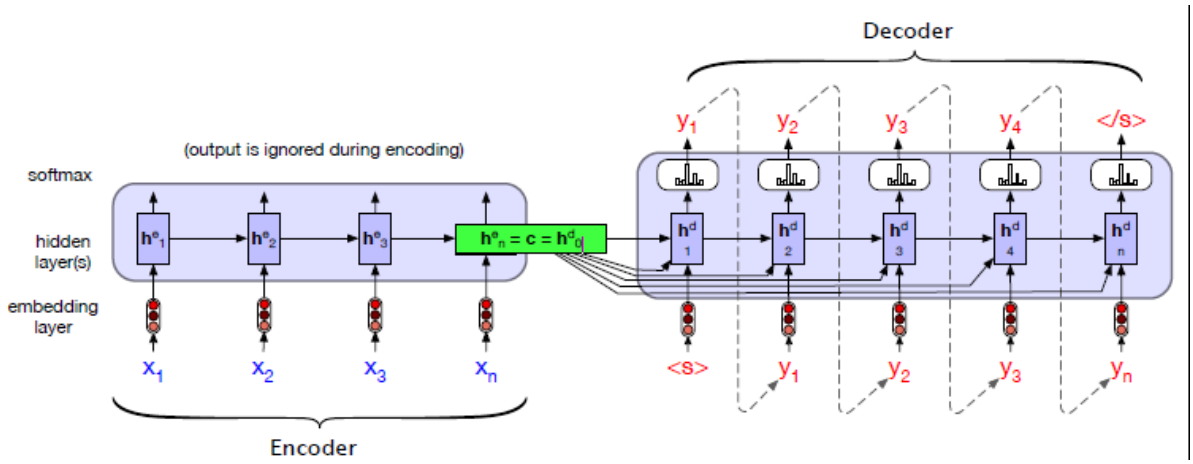
公式定义：

$$p(y|x) = p(y_1|x)p(y_2|y_1,x)p(y_3|y_1,y_2,x)...P(y_m|y_1,...,y_{m-1},x)$$



以encoder最后的y_n作为decoder的输入，对于decoder层来说解码 `<s>` +y_n的信息为文本，此时需要保存h_t的状态信息，映射为sequence的token输出。

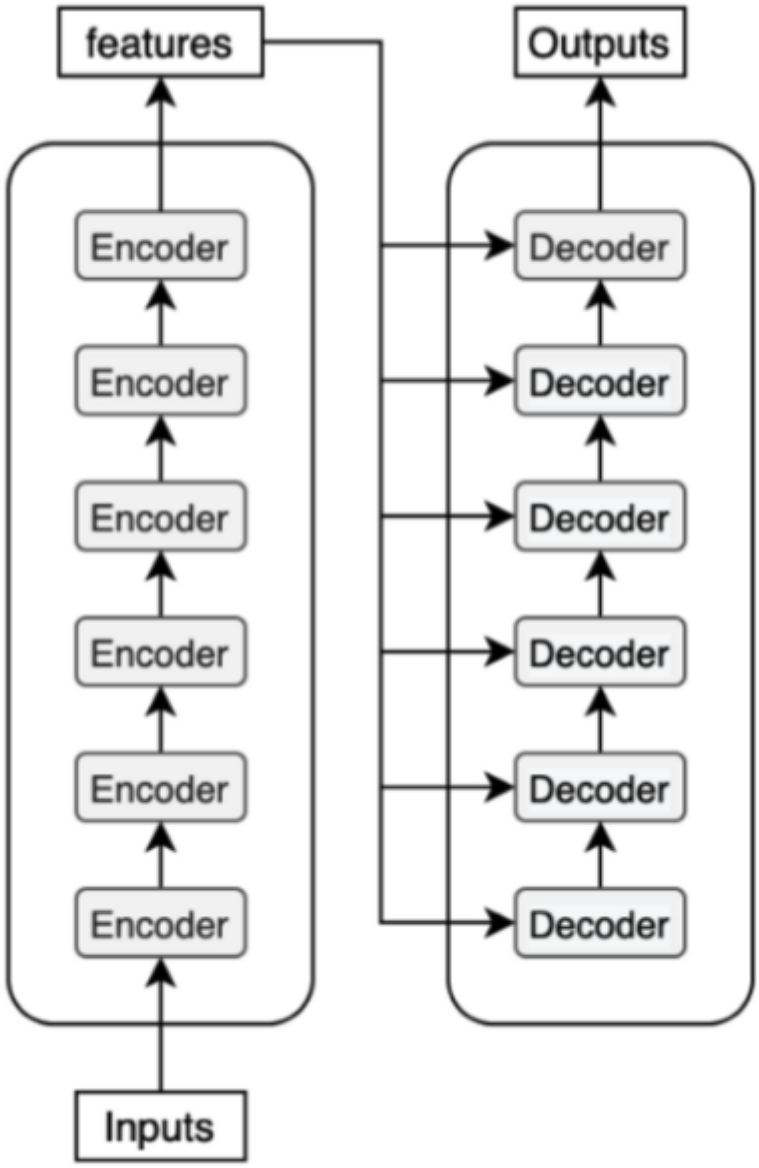
下图是更加正确的图像结构：



由于在encoder文内输出的信息c可能在后续decoder过程中被遗忘或者影响，对每一个step的解码都去做c的输入

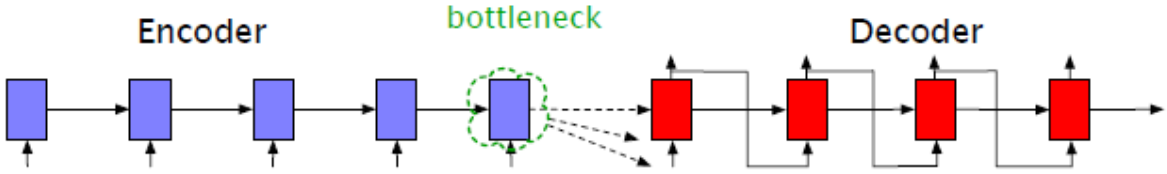
公式理解：

$$
\begin{aligned}
\mathbf{c} &= \mathbf{h}_n^e \\
\mathbf{h}_0^d &= \mathbf{c} \\
\mathbf{h}_t^d &= g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c}) \\
\mathbf{z}_t &= f(\mathbf{h}_t^d) \\
y_t &= \text{softmax}(\mathbf{z}_t)
\end{aligned}
$$



值得一提的是在RNN-DE架构的训练过程中，Decoder部分我们往往使用的是TF方法，为了加速训练并且让机器翻译正确的句子而不是生成流畅的外语句子。

RNN-DE架构的缺点非常明显训练速度非常缓慢，依赖于encoder部分生成的最终的$h_n = c$



但是对于decoder的不同step来说c在不同的情况下应该表征不同，而不是取最终的c_n，
$c = f(h_1, .. h_n)$

在不同step下 decoder应该要考虑的上下文信息c_t应该不同：

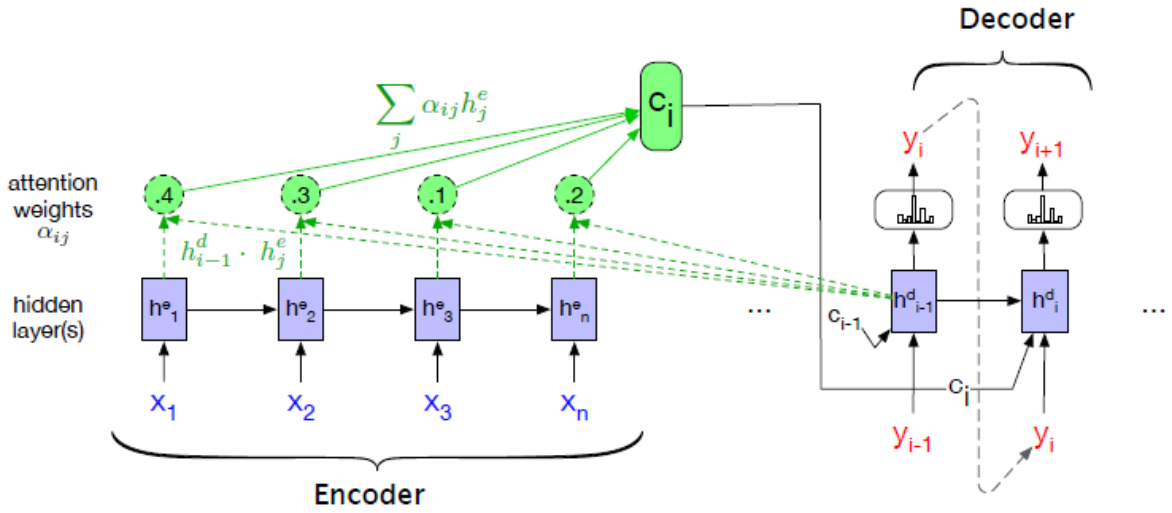
于是有人提出了ATTENTION机制，一个非常intuitive的想法是对于encoder的第i步和decoder的第j步，

如果$h_{j-1}^d \ and \ h_i^e$非常相似，说明本轮的生成的h_j^d和encoder过程中的第i步非常相近，于是更应该考虑此处的上下文含义即为c_i，对于相似属性自然而然的想到的是向量点击，最后对所有encoder过程中的步骤都做如下操作，最后取softmax计算注意力权重：

$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j) \ \forall j \in e)$$
$$= \frac{\exp(\text{score}(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j)}{\sum_k \exp(\text{score}(\mathbf{h}^d_{i-1}, \mathbf{h}^e_k))}$$

于是

$$c_t = \sum_j \alpha_{tj} h^e_j$$



除了点积求值，还可以维护一个可学习的W_s矩阵来计算分数:增大参数量，使得两个模块的向量不需要在表征上趋同

$$\text{score}(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j) = \mathbf{h}^d_{t-1} \mathbf{W}_s \mathbf{h}^e_j$$

EM algorithm

我们常见的MLE估计是对于给定的观测值求

$$\theta_{MLE} = \text{argmax}_\theta \log P(x|\theta)$$

但是对于存在隐变量的概率模型往往不能采用这种方法，例如在抛硬币过程中硬币的质地均匀性质影响每一次抛硬币的手感，会影响观测结果。于是对于这种问题MLE极大似然估计方法不能马上应用,N代表N次投掷或实验。

$$Y = (Y_1, Y_2, \ldots, Y_N)^T$$
$$Z = (Z_1, Z_2, \ldots, Z_N)^T$$
$$P(Y|\theta) = \sum_Z P(Z|\theta) P(Y|Z, \theta)$$
$$\hat{\theta} = \text{argmax}_\theta \log P(Y|\theta)$$

例 9.1（三硬币模型） 假设有 3 枚硬币，分别记作 A，B，C。这些硬币正面出现的概率分别是 π，p 和 q。进行如下掷硬币试验：先掷硬币 A，根据其结果选出硬币 B 或硬币 C，正面选硬币 B，反面选硬币 C；然后掷选出的硬币，掷硬币的结果，出现正面记作 1，出现反面记作 0；独立地重复 n 次试验（这里，$n=10$），观测结果如下：

$$1,1,0,1,0,0,1,0,1,1$$

假设只能观测到掷硬币的结果，不能观测掷硬币的过程。问如何估计三硬币正面出现的概率，即三硬币模型的参数。

EM算法步骤：

输入：观测变量数据X,隐变量数据Z，联合分布$P(X,Z|\theta)$，条件分布$P(Z|X,\theta)$=>观测到后往回算;

输出：模型参数θ

步骤：

$$1.\text{选择初值}\theta^{(0)}\text{，开始迭代}$$

$$2.E\text{步：记}\theta^{(i)}\text{为第}i\text{次迭代参数}\theta\text{估计值，在第}i+1\text{次迭代的}E\text{步，计算：}$$

$$Q\left(\theta,\theta^{(i)}\right)=E_Z\left[\log P\left(X,Z|\theta\right)|X,\theta^{(i)}\right]$$

$$=\sum_Z\log P\left(X,Z|\theta\right)P\left(Z|x,\theta^{(i)}\right)$$

$$3.M\text{步：求}\theta^{(i+1)},st:$$

$$\theta^{(i+1)}=\arg\max_\theta Q\left(\theta,\theta^{(i)}\right)$$

由于隐式问题的存在所以参数theta无法写出解析解，于是希望通过多次迭代寻找逼近的近似最优

$$\log P\left(x|\theta^{(t)}\right)=\log P\left(x,z|\theta\right)-\log P\left(z|x,\theta\right)$$

$$=\log\frac{P\left(x,z|\theta\right)}{q\left(z\right)}-\log\frac{P\left(z|x,\theta\right)}{q\left(z\right)}$$

$$\text{对左右两边求期望}E_{q(z)}\left[f\left(x,z\right)\right]$$

$$left=\int_z q\left(z\right)\log P\left(x|\theta^{(t)}\right)dz=\log P\left(x|\theta^{(t)}\right)$$

$$right=\int_z q\left(z\right)\log\frac{P\left(x,z|\theta\right)}{q\left(z\right)}dz-\int_z q\left(z\right)\log\frac{P\left(z|x,\theta\right)}{q\left(z\right)}dz$$

$$=ELBO+KL(q||p)_{\geqslant 0}$$

$$\log P\left(x|\theta\right)=ELBO+KL$$

$$\text{假设}q\left(z\right)=p\left(z|x,\theta\right)$$

$$\hat{\theta}=\arg\max_\theta ELBO$$

$$=\arg\max_\theta\int_z q\left(z\right)\log\frac{P\left(x,z|\theta\right)}{q\left(z\right)}dz$$

$$=\arg\max_\theta\int_z p\left(z|x,\theta^{(t)}\right)\left[\log p\left(x,z|\theta\right)-\log p\left(z|x,\theta^{(t)}\right)\right]dz$$

$$=\arg\max_\theta\int_z p\left(z|x,\theta^{(t)}\right)\cdot\log p\left(x,z|\theta\right)dz=\arg\max_\theta E_z\left[\log p\left(x,z|\theta\right)|x,\theta^{(t)}\right]$$

另有关于KL散度的定义：

$$
\begin{aligned}
&D_{KL}\left(p||q\right)=H\left(p,q\right)-H\left(p\right)\\
&=-\sum p\left(x\right)\log q\left(x\right)+\sum p\left(x\right)\log p\left(x\right)\\
&=\sum p\left(x\right)\log\frac{p\left(x\right)}{q\left(x\right)}
\end{aligned}
$$

证明收敛性：

$$
\log P\left(x|\theta^{(t)}\right)\leq\log P\left(x|\theta^{(t+1)}\right)
$$

以下是推导

$$
\begin{aligned}
&\log P(x|\theta)=\log\frac{P(x,z|\theta)}{P(z|x,\theta)}=\log P(x,z|\theta)-\log P(z|x,\theta)\\
&left=\int_z\log P(x|\theta)P(z|x,\theta^{(t)})dz=\log P(x|\theta)\\
&right=\underbrace{\int_z\log P(x,z|\theta)P(z|x,\theta^{(t)})dz}_{Q\left(\theta,\theta^{(t)}\right)}-\underbrace{\int_z\log P(z|x,\theta)P(z|x,\theta^{(t)})dz}_{H\left(\theta,\theta^{(t)}\right)}
\end{aligned}
$$

引入

$$
\theta^{(t+1)}=\arg\max_\theta\int_z\log P\left(x,z|\theta\right)\cdot P\left(z|x,\theta^{(t)}\right)dz=\mathbf{E}_{z|x,\theta^{(t)}}\left[\log P\left(x,z|\theta\right)\right]
$$

$$
\begin{aligned}
&D_{KL}\left(p||q\right)=H\left(p,q\right)-H\left(p\right)\\
&=-\sum p\left(x\right)\log q\left(x\right)+\sum p\left(x\right)\log p\left(x\right)\\
&=\sum p\left(x\right)\log\frac{p\left(x\right)}{q\left(x\right)}
\end{aligned}
$$

EM算法例子

首先介绍一个使用 EM 算法的例子。

例 9.1（三硬币模型） 假设有 3 枚硬币，分别记作 A，B，C。这些硬币正面出现的概率分别是 π，p 和 q。进行如下掷硬币试验：先掷硬币 A，根据其结果选出硬币 B 或硬币 C，正面选硬币 B，反面选硬币 C；然后掷选出的硬币，掷硬币的结果，出现正面记作 1，出现反面记作 0；独立地重复 n 次试验（这里，$n=10$），观测结果如下：

$$
1,1,0,1,0,0,1,0,1,1
$$

假设只能观测到掷硬币的结果，不能观测掷硬币的过程。问如何估计三硬币正面出现的概率，即三硬币模型的参数。

EM算法首先选取参数的初值，记作 $\theta^{(0)}=(\pi^{(0)},p^{(0)},q^{(0)})$，然后通过下面的步骤迭代计算参数的估计值，直至收敛为止。第 i 次迭代参数的估计值为 $\theta^{(i)}=(\pi^{(i)},p^{(i)},q^{(i)})$。EM算法的第 $i+1$ 次迭代如下。

E 步：计算在模型参数 $\pi^{(i)}$，$p^{(i)}$，$q^{(i)}$ 下观测数据 y_j 来自掷硬币 B 的概率

$$\mu_j^{(i+1)}=\frac{\pi^{(i)}(p^{(i)})^{y_j}(1-p^{(i)})^{1-y_j}}{\pi^{(i)}(p^{(i)})^{y_j}(1-p^{(i)})^{1-y_j}+(1-\pi^{(i)})(q^{(i)})^{y_j}(1-q^{(i)})^{1-y_j}} \tag{9.5}$$

M 步：计算模型参数的新估计值

$$\pi^{(i+1)}=\frac{1}{n}\sum_{j=1}^{n}\mu_j^{(i+1)} \tag{9.6}$$

$$p^{(i+1)}=\frac{\sum_{j=1}^{n}\mu_j^{(i+1)}y_j}{\sum_{j=1}^{n}\mu_j^{(i+1)}} \tag{9.7}$$

$$q^{(i+1)}=\frac{\sum_{j=1}^{n}(1-\mu_j^{(i+1)})y_j}{\sum_{j=1}^{n}(1-\mu_j^{(i+1)})} \tag{9.8}$$

进行数值计算。假设模型参数的初值取为

$$\pi^{(0)}=0.5,\quad p^{(0)}=0.5,\quad q^{(0)}=0.5$$

由式 (9.5)，对 $y_j=1$ 与 $y_j=0$ 均有 $\mu_j^{(1)}=0.5$。

利用迭代公式 (9.6)～公式 (9.8)，得到

$$\pi^{(1)}=0.5,\quad p^{(1)}=0.6,\quad q^{(1)}=0.6$$

由式 (9.5)，

$$\mu_j^{(2)}=0.5,\quad j=1,2,\cdots,10$$

继续迭代，得

$$\pi^{(2)}=0.5,\quad p^{(2)}=0.6,\quad q^{(2)}=0.6$$

于是得到模型参数 θ 的极大似然估计：

$$\hat{\pi}=0.5,\quad \hat{p}=0.6,\quad \hat{q}=0.6$$

$\pi=0.5$ 表示硬币 A 是均匀的，这一结果容易理解。

如果取初值 $\pi^{(0)}=0.4$，$p^{(0)}=0.6$，$q^{(0)}=0.7$，那么得到的模型参数的极大似然估计是 $\hat{\pi}=0.4064$，$\hat{p}=0.5368$，$\hat{q}=0.6432$。这就是说，EM算法与初值的选择有关，选择不同的初值可能得到不同的参数估计值。■

通俗的解释：

图 9.1 给出 EM 算法的直观解释。图中上方曲线为 $L(\theta)$，下方曲线为 $B(\theta, \theta^{(i)})$。由式 (9.14)，$B(\theta, \theta^{(i)})$ 为对数似然函数 $L(\theta)$ 的下界。由式 (9.15)，两个函数在点 $\theta = \theta^{(i)}$

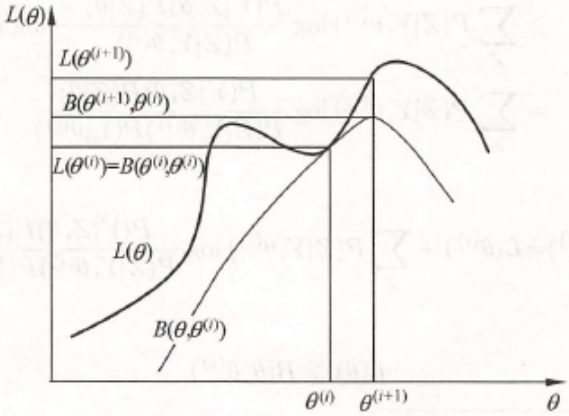


图 9.1　EM 算法的解释

处相等。由式 (9.16) 和式 (9.17)，EM 算法找到下一个点 $\theta^{(i+1)}$ 使函数 $B(\theta, \theta^{(i)})$ 极大化，也使函数 $Q(\theta, \theta^{(i)})$ 极大化。这时由于 $L(\theta) \geqslant B(\theta, \theta^{(i)})$，函数 $B(\theta, \theta^{(i)})$ 的增加，保证对数似然函数 $L(\theta)$ 在每次迭代中也是增加的。EM 算法在点 $\theta^{(i+1)})$ 重新计算 Q 函数值，进行下一次迭代。在这个过程中，对数似然函数 $L(\theta)$ 不断增大。从图可以推断出 EM 算法不能保证找到全局最优值。

9.1.3　EM 算法在无监督学习中的应用

监督学习是由训练数据 $\{(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)\}$ 学习条件概率分布 $P(Y|X)$ 或决策函数 $Y = f(X)$ 作为模型，用于分类、回归、标注等任务。这时训练数据中的每个样本点由输入和输出对组成。

有时训练数据只有输入没有对应的输出 $\{(x_1, \cdot), (x_2, \cdot), \cdots, (x_N, \cdot)\}$，从这样的数据学习模型称为无监督学习问题。EM 算法可以用于生成模型的无监督学习。生成模型由联合概率分布 $P(X, Y)$ 表示，可以认为无监督学习训练数据是联合概率分布产生的数据。X 为观测数据，Y 为未观测数据。