

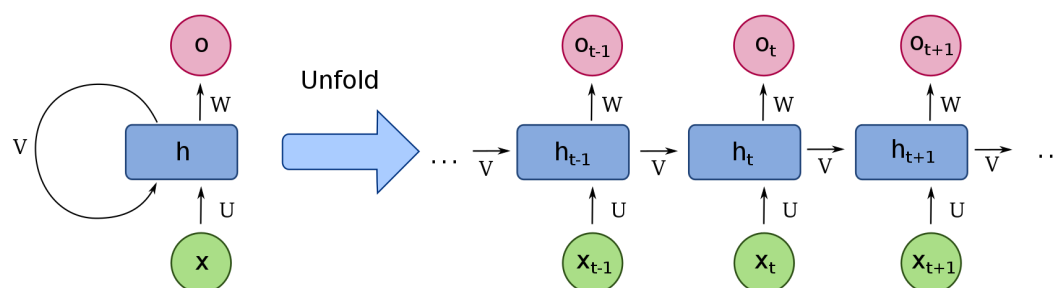
Chapter-9 RNNs and LSTMs

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



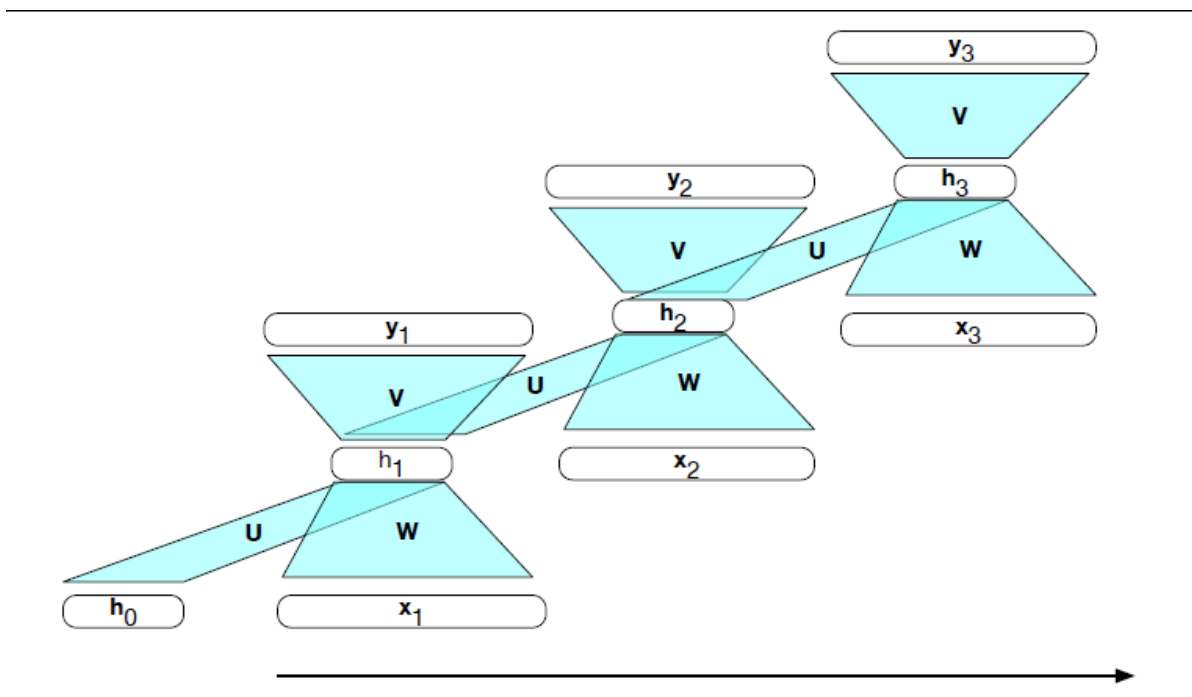
对于RNN网络来说，对于 $step = t$ 为了考虑context或者说之前时间步 x 所蕴含的信息来影响后续输出的 O_t ，网络中同时储存了 h_{t-1} 作为蕴含了前文信息的向量：

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

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

$$P(\text{fish}|\text{Thanks for all the})$$

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

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 $X = [x_1; \dots; x_t; \dots x_N]$ ，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。

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

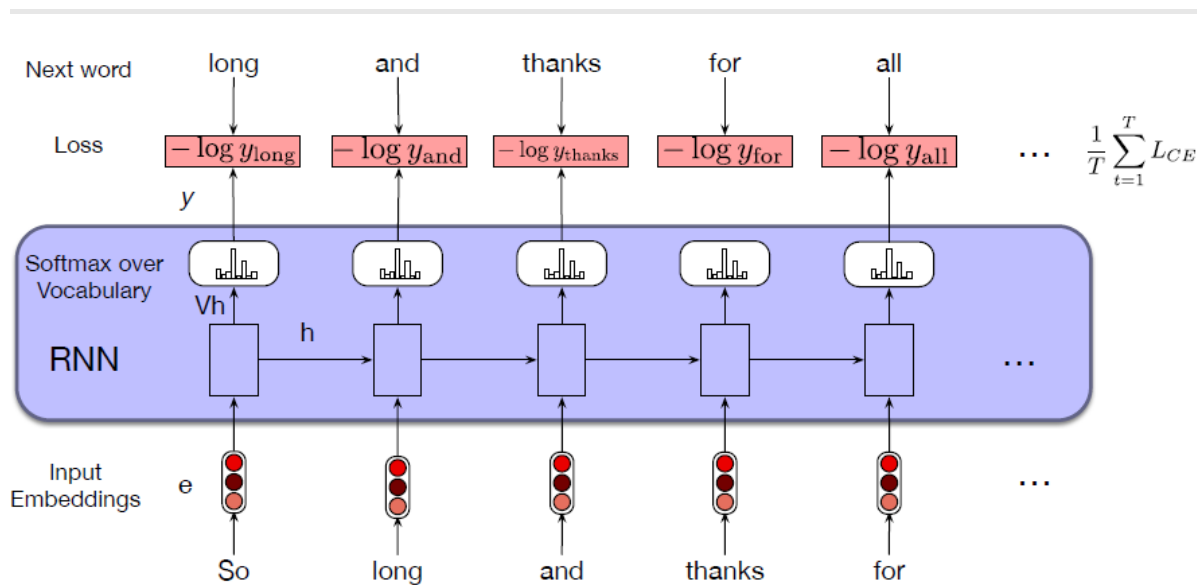
在RNNs modeling下:

$$P(w_{1:n}) = \prod y_i[w_i]$$

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：

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



```
1 RNN = torch.nn.RNN(input_size, output_size, layer_nums)
2 #input=[b, sq_len, vector_dim]
3 #h0 = [layer_num, sq_len, out_dim]
4 outp, hidden_states = RNN(input, h0)
5 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：

```
1 | <S> 我 非常 喜欢 吃 饺子 和 馄饨 </S>
```

在训练时：

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

2. 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

1	0. 输入	输出	Ground Truth
2	1. <S>	你	我
3	2. 你	咋	非常
4	3. 咋	?	喜欢
5	4. ?	?	吃
6	...		

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

9.2.3 Weight Tying

对于之前的公式

$$h_t = g(Uh_{t-1} + Wx_t)$$
$$y_t = f(Vh_t)$$

我们发现在语言建模过程中理解为：

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

发现对于(1)(3)：

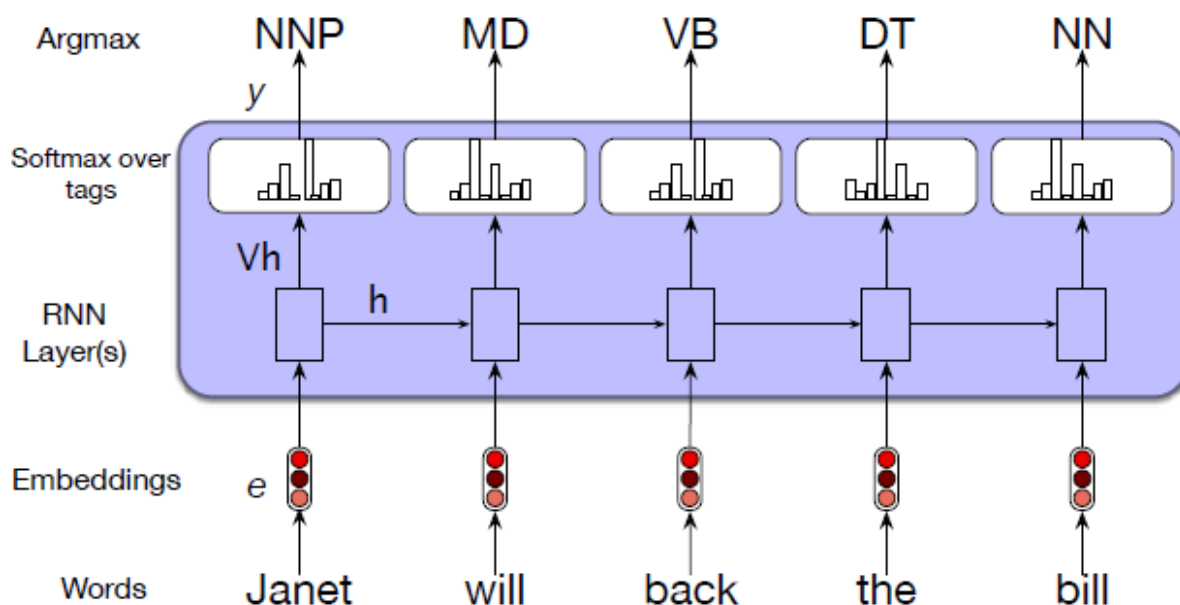
```
1 | E.dim=[d_r, |V|]
2 | e_t.dim=[d_r, 1]
3 | h_t.dim=[d_hidden, 1]
4 | y_hat -> similarity cal()
```

我们可以发现其实对 V 来说 h_t 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

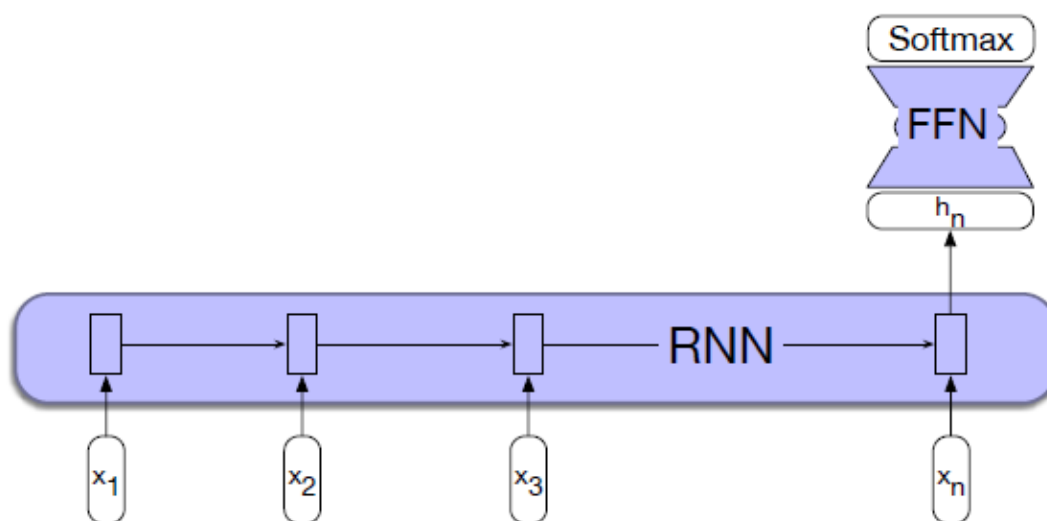
9.3 RNNs for other NLP tasks

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

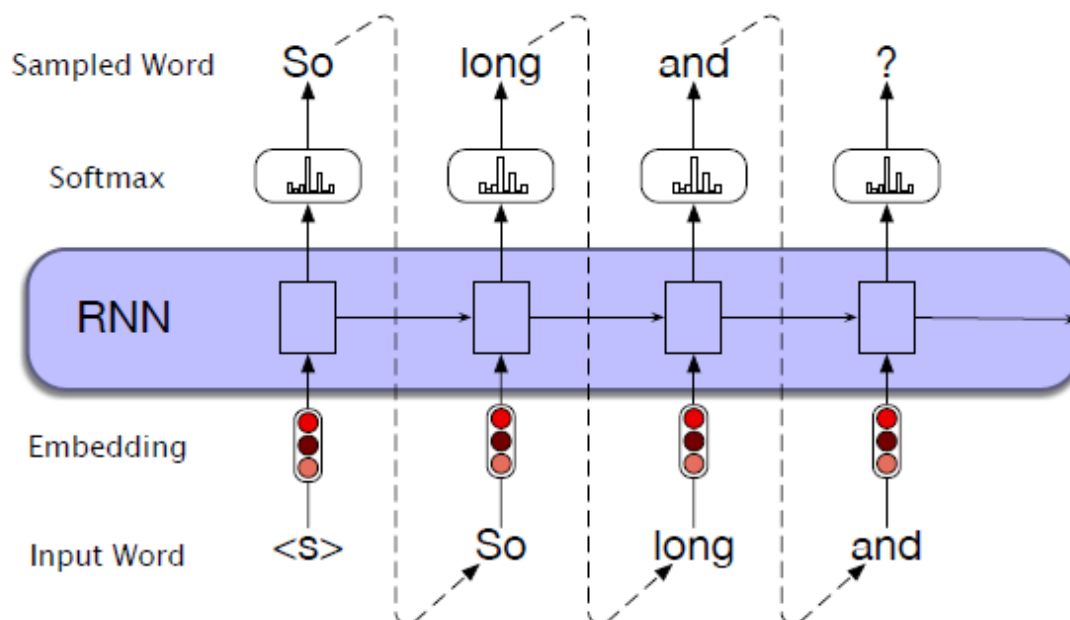
对于最后的 h_n 获取存在两种方式：

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

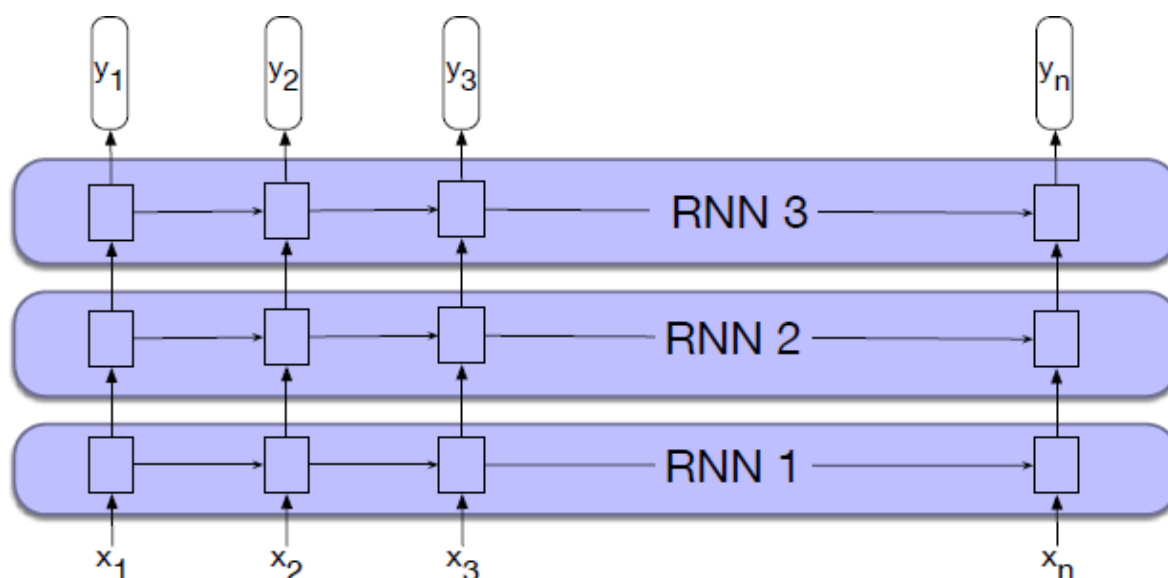
对于RNN式生成模型：

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 $\langle s \rangle$ 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)

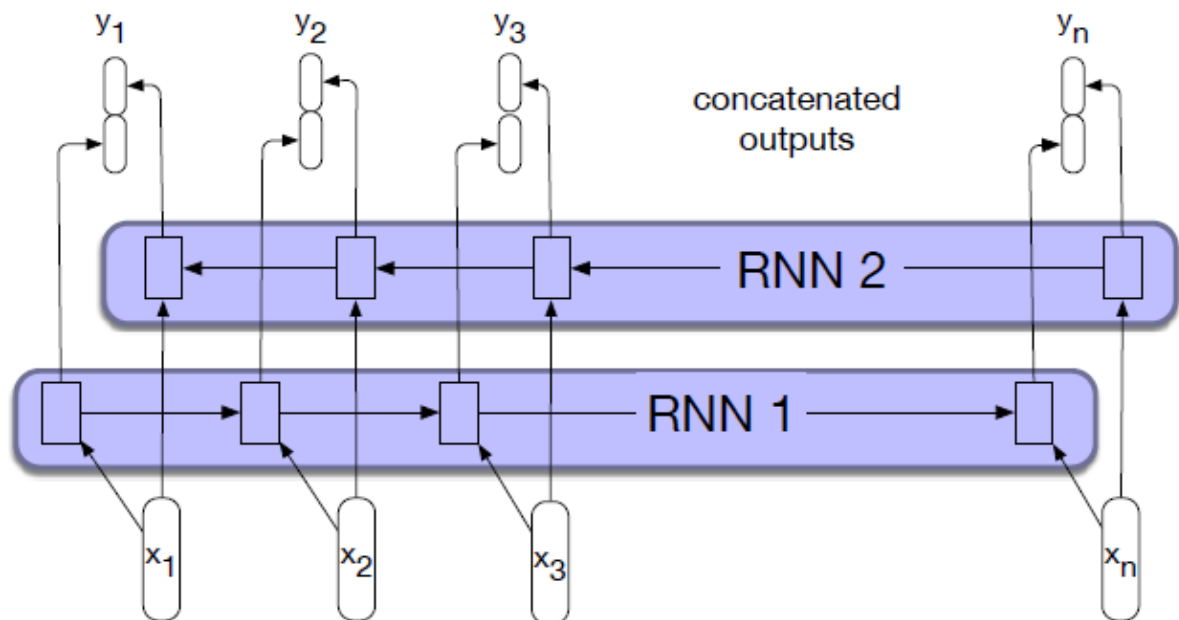
从直觉来看stacked RNNs outperform better的原因是想深层卷积网络一样可以获取时序信号中更加抽象的时序信息，但是对于RNNs的层数还是要具体情况具体分析；能力的提升也带来的计算开销和时间开销。

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 $token_{n \rightarrow t}$ 来获得新的逆向时序信号。

$$\begin{aligned}
 h_t^f &= \text{RNN}_f(x_1, \dots, x_t) \\
 h_t^b &= \text{RNN}_b(x_t, \dots, x_n) \\
 h_t &= \text{concat}[h_t^f; h_t^b]
 \end{aligned}$$

这就是双向RNN网络, *Bidirectional RNNs*



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

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~

综上所述提出的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

$$\mathbf{f}_t = \sigma(\mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{W}_f \mathbf{x}_t)$$

$$\mathbf{k}_t = \mathbf{c}_{t-1} \odot \mathbf{f}_t$$

2. needed information 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

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

$$\mathbf{j}_t = \mathbf{g}_t \odot \mathbf{i}_t$$

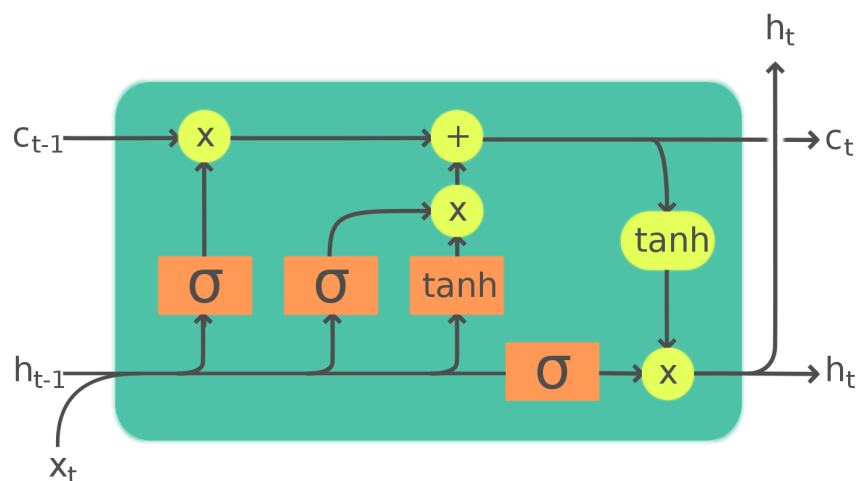
4.merge infomation 本轮cell所暂留的信息

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

5.最后获得隐藏层输出 信息

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

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

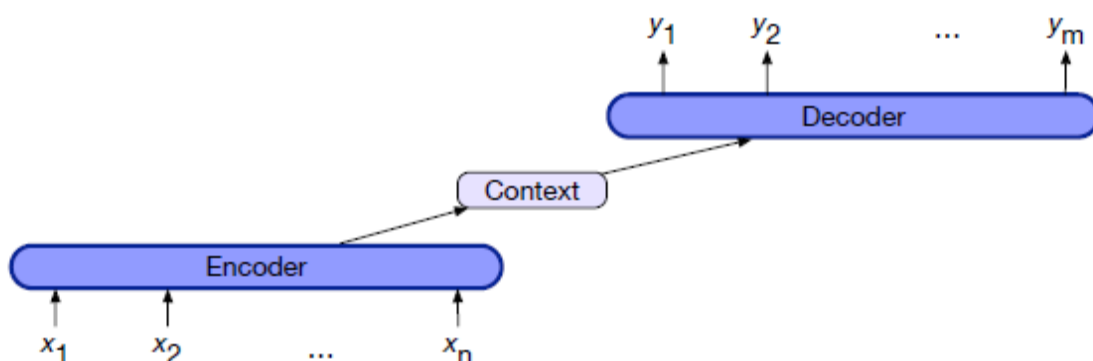


Legend:

Layer	Componentwise	Copy	Concatenate

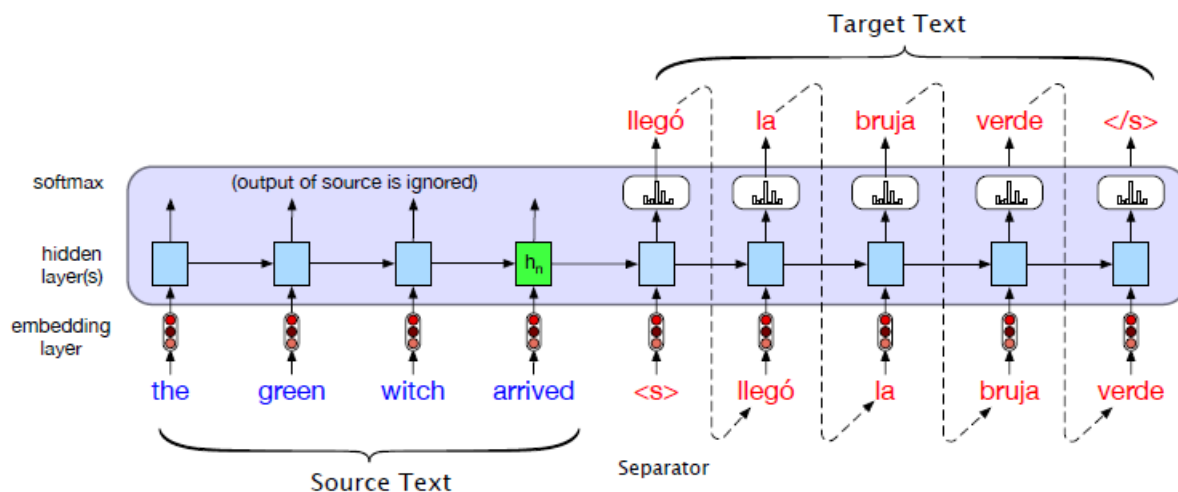
9.6 Summary: Common RNN NLP Architectures

RNN in encoder & decoder: machine translation & sequence2sequence



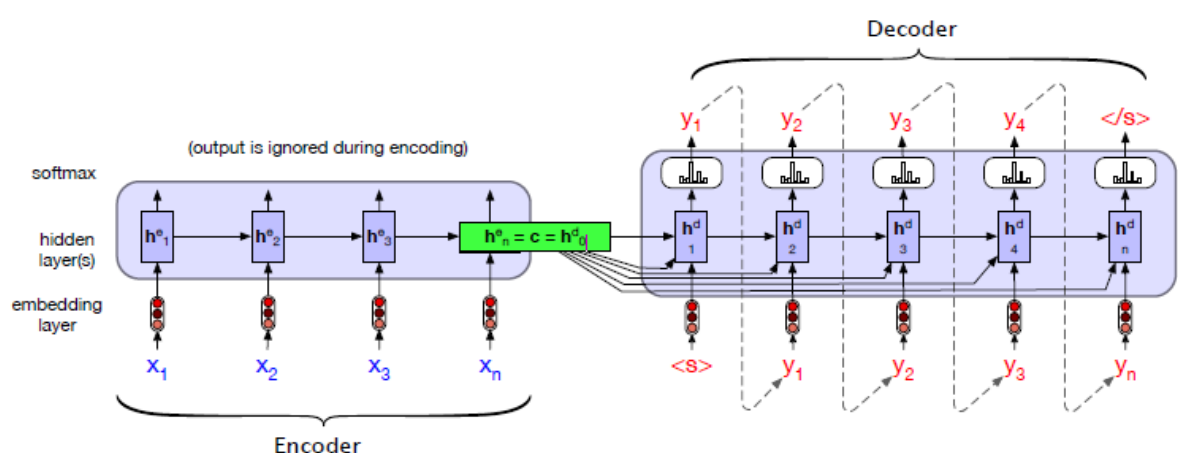
公式定义：

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



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。

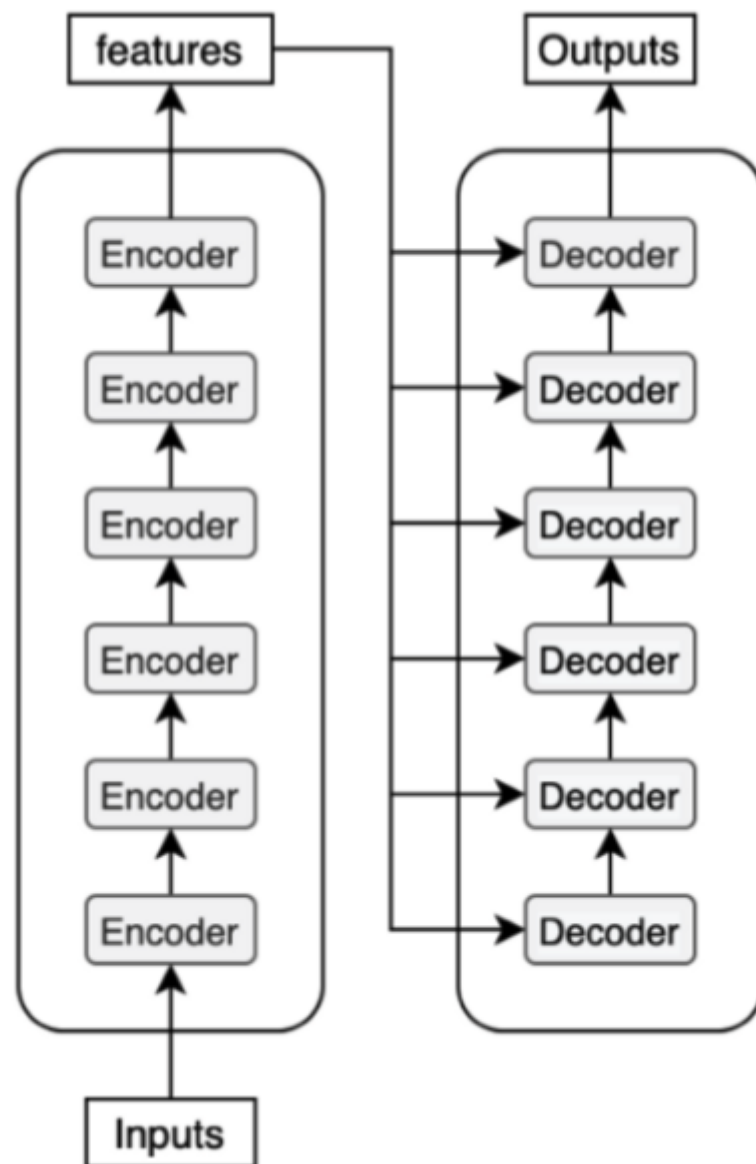
下图是更加正确的图像结构：



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

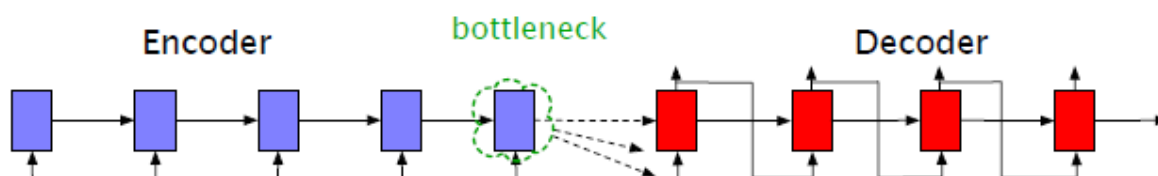
公式理解：

$$\begin{aligned} c &= h_n^e \\ h_0^d &= c \\ h_t^d &= g(\hat{y}_{t-1}, h_{t-1}^d, c) \\ z_t &= f(h_t^d) \\ y_t &= \text{softmax}(z_t) \end{aligned}$$



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 ， $c = f(h_1, \dots, h_n)$

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，

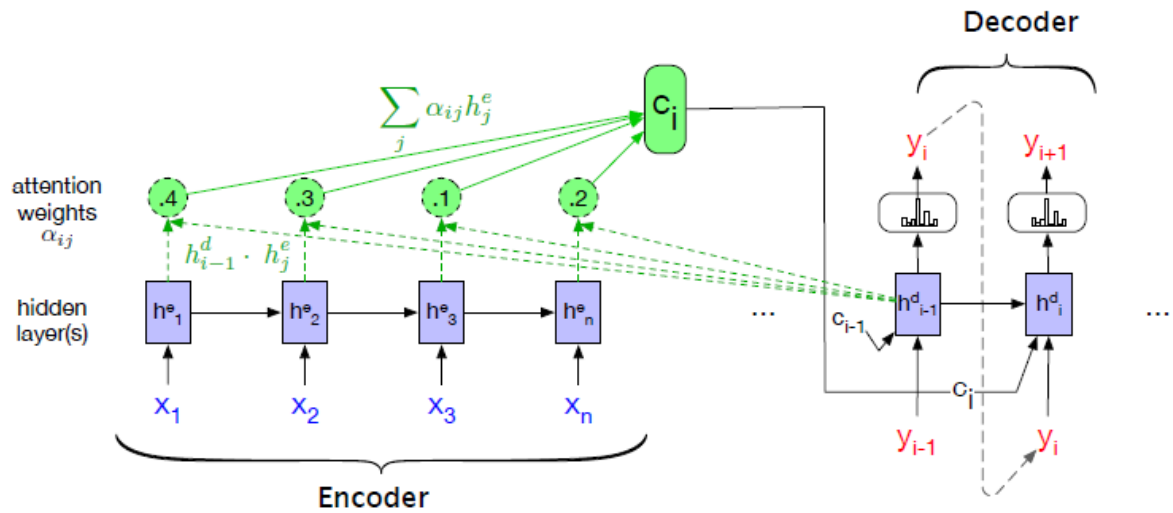
如果 h_{j-1}^d and h_i^e 非常相似，说明本轮的生成的 h_j^d 和encoder过程中的第 i 步非常相近，于是更应该考虑此处的上下文含义即为 c_i ，对于相似属性自然而然的想到的是向量点击，最后对所有encoder过程中的步骤都做如下操作，最后取softmax计算注意力权重：

$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e) \quad \forall j \in e)$$

$$= \frac{\exp(\text{score}(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e))}{\sum_k \exp(\text{score}(\mathbf{h}_{i-1}^d, \mathbf{h}_k^e))}$$

于是

$$c_t = \sum_j \alpha_{tj} h_j^e$$



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

$$\text{score}(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e) = \mathbf{h}_{i-1}^d W_s \mathbf{h}_j^e$$