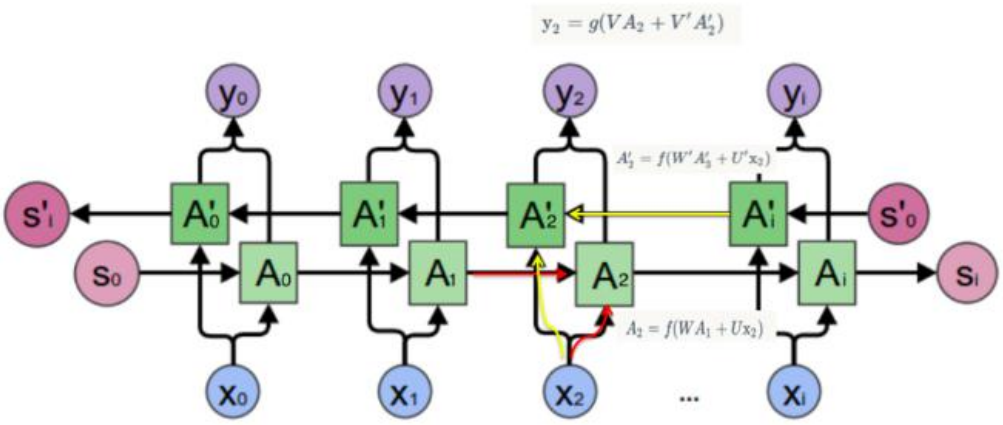
双向 LSTM

使用双向 LSTM 的原因：单向的 RNN，是根据前面的信息推出后面的，但有时候只看前面的词是不够的。

双向卷积神经网络的隐藏层要保存两个值， A 参与正向计算， A' 参与反向计算。最终的输出值 y_2 取决于 A_2 和 A_2'，正向计算时，隐藏层的 s_t 与 s_{t-1} 有关；反向计算时，隐藏层的 s_t 与 s_{t+1} 有关。



计算方法为：

$$y_2 = g(VA_2 + V'A_2')$$

A 和 A'的计算方法：

$$A_2 = f(WA_1 + Ux_2)$$
$$A_2' = f(W'A_3' + U'x_2)$$

最终的输出取决于正向和反向计算的加和

$$o_t = g(Vs_t + V's_t')$$
$$s_t = f(Ux_t + Ws_{t-1})$$
$$s_t' = f(U'x_t + W's_{t+1}')$$

正向计算和反向计算不共享权重，也就是说 U 和 U'、W 和 W'、V 和 V'都是不同的权重矩阵。

多层 LSTM

我们堆叠两个以上的隐藏层

$$
\begin{aligned}
o_t &= g(V^{(i)}s_t^{(i)} + V'^{(i)}s_t'^{(i)}) \\
s_t^{(i)} &= f(U^{(i)}s_t^{(i-1)} + W^{(i)}s_{t-1}) \\
s_t'^{(i)} &= f(U'^{(i)}s_t'^{(i-1)} + W'^{(i)}s'_{t+1}) \\
&\cdots \\
s_t^{(1)} &= f(U^{(1)}x_t + W^{(1)}s_{t-1}) \\
s_t'^{(1)} &= f(U'^{(1)}x_t + W'^{(1)}s'_{t+1})
\end{aligned}
$$

seq2seq

seq2seq 是一个 Encoder－Decoder 结构的网络，它的输入是一个序列，输出也是一个序列， Encoder 中将一个可变长度的信号序列变为固定长度的向量表达，Decoder 将这个固定长度的向量变成可变长度的目标的信号序列。

这个结构最重要的地方在于输入序列和输出序列的长度是可变的。

encoder-decoder 模型

Encoder-Decoder 不是一种模型，而是一种框架，一种处理问题的思路，最早应用于机器翻译领域，输入一个序列，输出另外一个序列。

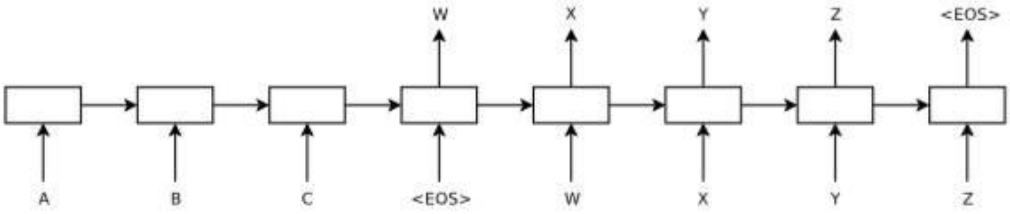
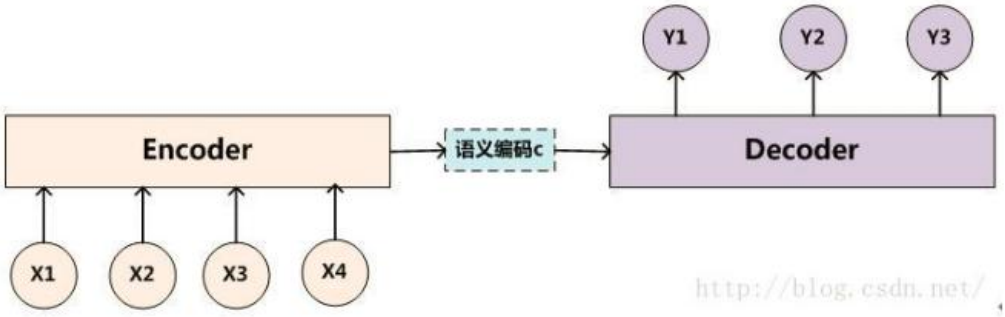


Figure 1: Our model reads an input sentence "ABC" and produces "WXYZ" as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.



encoder 部分是将输入序列表示成一个带有语义的向量。

decoder 部分是以 encoder 生成的 hidden state vector 作为输入“解码”出目标文本

序列，本质上是一个语言模型。

在 RNN Encoder-Decoder 的工作当中，我们用一个 RNN 去模拟大脑的读入动作，用一个特定长度的特征向量去模拟我们的记忆，然后再用另外一个 RNN 去模拟大脑思考得到答案的动作，将三者组织起来利用就成了一个可以实现 Sequence2Sequence 工作的“模拟大脑”了。对于句子对<X,Y>，我们的目标是给定输入句子 X，期待通过 Encoder-Decoder 框架来生成目标句子 Y。X 和 Y 可以是同一种语言，也可以是两种不同的语言。而 X 和 Y 分别由各自的单词序列构成：

$$\mathbf{X} = \langle \mathbf{x_1}, \mathbf{x_2} \dots \mathbf{x_m} \rangle$$

$$\mathbf{Y} = \langle \mathbf{y_1}, \mathbf{y_2} \dots \mathbf{y_n} \rangle$$

Encoder 顾名思义就是对输入句子 X 进行编码，将输入句子通过非线性变换转化为中间语义表示 C：

$$\mathbf{C} = \mathcal{F}(\mathbf{x_1}, \mathbf{x_2} \dots \mathbf{x_m})$$

对于解码器 Decoder 来说，其任务是根据句子 X 的中间语义表示 C 和之前已经生成的历史信息 $y_1, y_2 \cdots y_{i-1}$ 来生成 i 时刻要生成的单词 y_i

$$\mathbf{y_i} = \mathcal{G}(\mathbf{C}, \mathbf{y_1}, \mathbf{y_2} \dots \mathbf{y_{i-1}})$$

每个 y_i 都依次这么产生，那么看起来就是整个系统根据输入句子 X 生成了目标句子 Y。

在 RNN 中，上式又可以简化成

$$y_t = g(y_{t-1}, s_t, C)$$

其中是输出 RNN 中的隐藏层，C 代表语义向量，表示上个时间段的输出，反过来作为这个时间段的输入。g 则可以是一个非线性的多层的神经网络，产生词典中各个词语属于的概率。

encoder-decoder 模型局限性很大。最大的局限性就在于编码和解码之间的唯一联系就是一个固定长度的语义向量 C，编码器要将整个序列的信息压缩进一个固定长度的向量中去。这样做有两个弊端，一是语义向量无法完全表示整个序列的信息，二是先输入的内容携带的信息会被后输入的信息稀释掉。输入序列越长，这个现象就越严重。