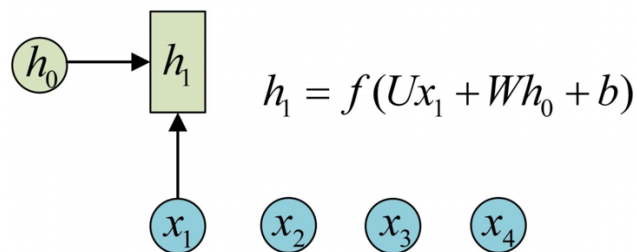


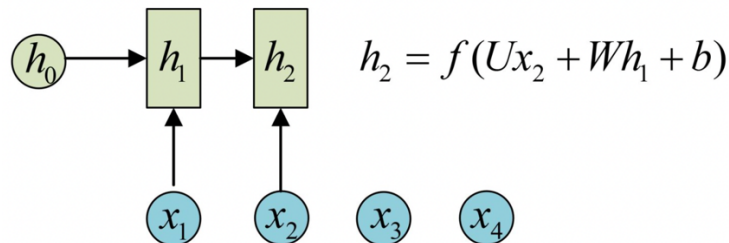
RNN_LSTM_GRU

1. 经典 RNN

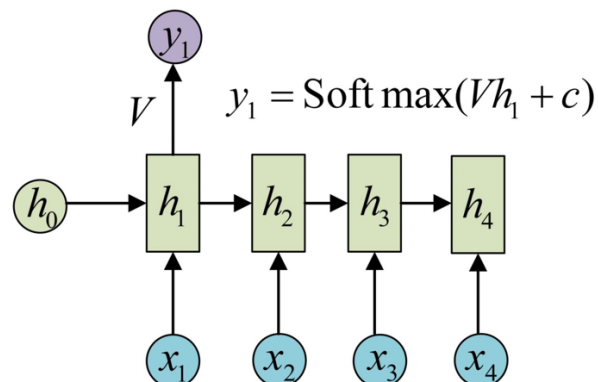
- 在实际应用中，会遇到很多序列形的数据，例如
 - a) 自然语言处理问题: x_1 可以看做是第一个单词, x_2 可以看做是第二个单词, 依次类推。
 - b) 语音处理: 此时, x_1, x_2, x_3, \dots 是每帧的声音信号。
 - c) 时间序列问题: 例如每天的股票价格等。
- 序列形的数据就不太好用原始的神经网络处理了。为了建模序列问题, RNN 引入了隐状态 h (hidden state) 的概念, h 可以对序列形的数据提取特征, 接着再转换为输出。(可以将 h 视为包含了 time-step 的信息)
 - i. 先从 h_1 的计算开始看:



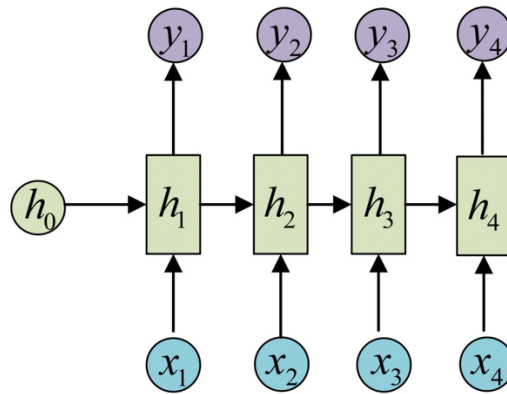
- ii. h_2 的计算和 h_1 类似。要注意的是, 在计算时, 每一步使用的参数 U, W, b 都是一样的, 也就是说每个步骤的参数都是共享的, 这是 **RNN 的重要特点**。
 - iii. 以下计算过程可以无限地持续下去。



- iv. 得到输出值的方式就是直接通过 h 进行计算



v. 完整结构

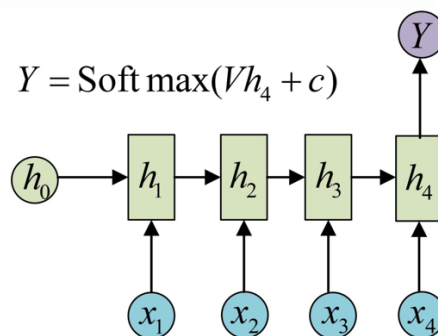


- 输入和输出序列必须要是等长的，即 $N - N$ 。
由于这个限制的存在，经典 RNN 的适用范围比较小，但也有一些问题适合用经典的 RNN 结构建模，如：
 - a) 计算视频中每一帧的分类标签。因为要对每一帧进行计算，因此输入和输出序列等长。
 - b) 输入为字符，输出为下一个字符的概率，即著名的 Char RNN。

2. RNN 变种(其他类型)

2.1 N - 1

- 要处理的问题输入是一个序列，输出是一个单独的值而不是序列
-> 只在最后一个 h 上进行输出变换就可以了

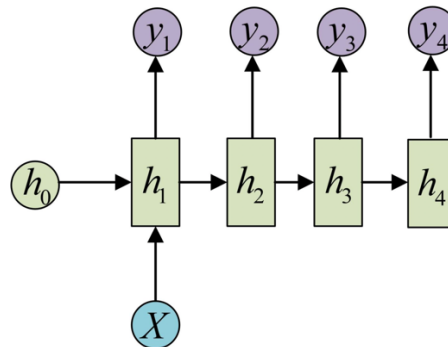


- 这种结构通常用来处理序列分类问题。
 - a) 输入一段文字判别它所属的类别。
 - b) 输入一个句子判断其情感倾向。
 - c) 输入一段视频并判断它的类别。

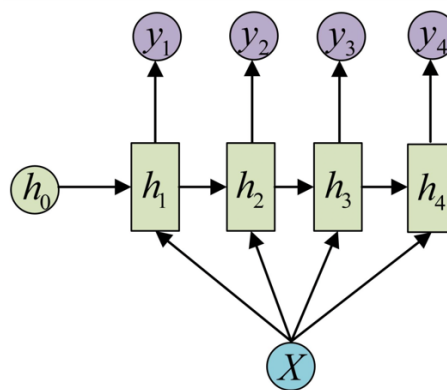
.....

2.2 1 - N

- 输入不是序列而输出为序列的情况 -> 只在序列开始进行输入计算



- 还有一种结构是把输入信息 X 作为每个阶段的输入



- 这种 1 VS N 的结构可以处理的问题有
 - a) 从图像生成文字(image caption), 此时输入的 X 就是图像的特征, 而输出的 y 序列就是一段句子。
 - b) 从类别生成语音或音乐等。

2.3 N - M

- RNN 最重要的一个变种 : N - M。**

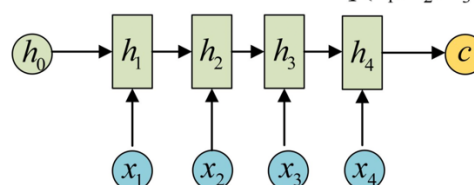
这种结构又叫 **Encoder-Decoder 模型**, 也可以称之为 **Seq2Seq 模型**。

- 原始的 RNN (N-N)要求序列等长, 然而我们遇到的大部分问题序列都是不等长的, 如机器翻译中, 源语言和目标语言的句子往往并没有相同的长度。
 - i. 为此, Encoder-Decoder 结构先将输入数据编码成一个上下文向量 c 得到 c 有多种方式, 最简单的方法就是把 Encoder 的最后一个隐状态赋值给 c , 还可以对最后的隐状态做一个变换得到 c , 也可以对所有的隐状态做变换。

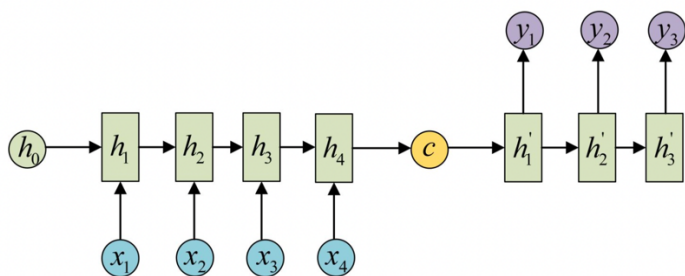
$$(1) c = h_4$$

$$(2) c = q(h_4)$$

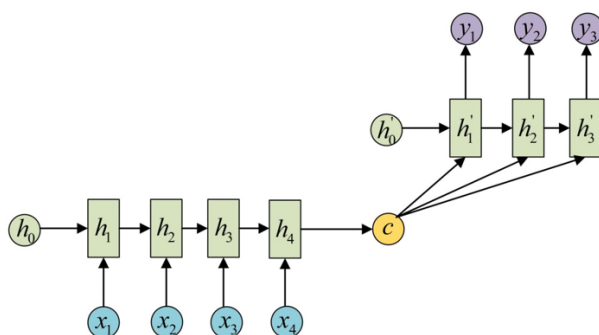
$$(3) c = q(h_1, h_2, h_3, h_4)$$



- ii. 拿到 c 之后，就用另一个 RNN 网络对其进行解码，这部分 RNN 网络被称为 Decoder。具体做法就是将 c 当做之前的初始状态 h_0 输入到 Decoder 中。



还有一种做法是将 c 当做每一步的输入。



- 由于这种 Encoder-Decoder 结构不限制输入和输出的序列长度，因此应用的范围非常广泛。
 - a) 机器翻译: Encoder-Decoder 的最经典应用，事实上这一结构就是在机器翻译领域最先提出的。
 - b) 文本摘要: 输入是一段文本序列，输出是这段文本序列的摘要序列。
 - c) 阅读理解: 将输入的文章和问题分别编码，再对其进行解码得到问题的答案。
 - d) 语音识别: 输入是语音信号序列，输出是文字序列。
-