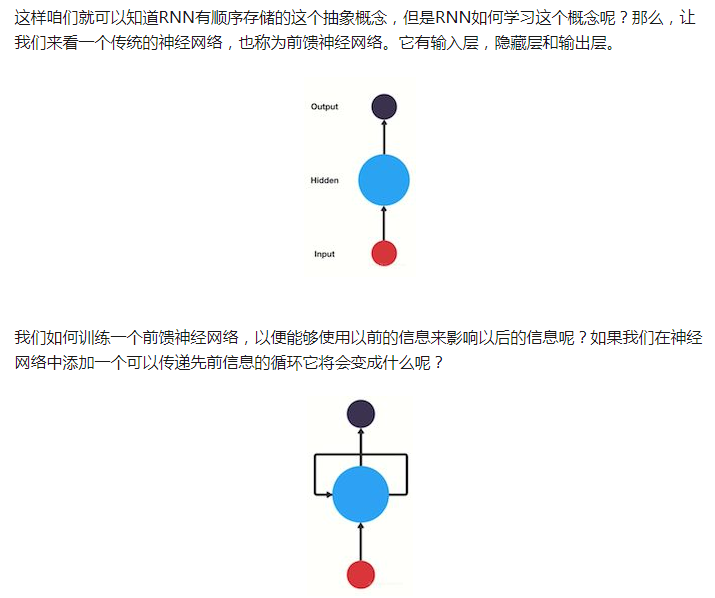
RNN

<https://zhuanlan.zhihu.com/p/45289691?utm_source=wechat_session&utm_medium=social&utm_oi=685930719541465088>

<https://zhuanlan.zhihu.com/p/28054589?utm_source=wechat_session&utm_medium=social&utm_oi=685930719541465088>

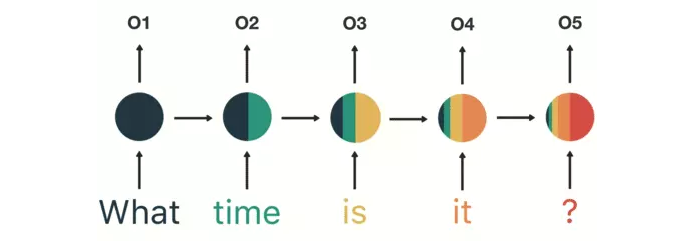
RNN是神经网络中的一种，它擅长对序列数据进行建模处理,音频是一种自然的序列，你可以将音频频谱图分成块并将其馈入RNN,文本也是一种形式的序列，你可以将文本分成一系列字符或一系列单词。

递归神经网络



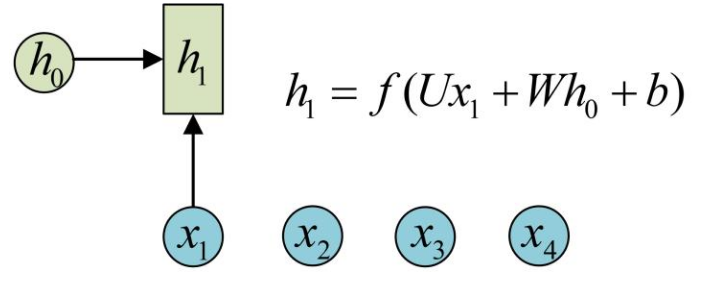
假设用户输入：what time is it？我们需要预测用户的意图，首先，我们将句子分解为单个单词。RNN按顺序工作，所以我们一次只能输入一个字。

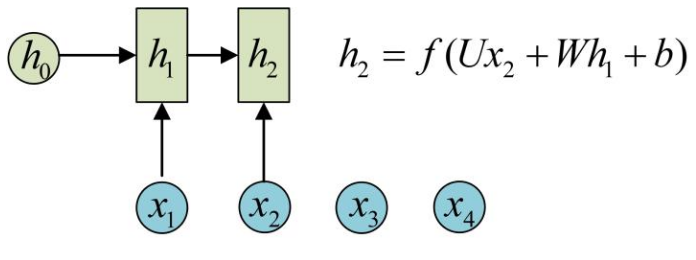
1. 第一步是将“What”输入RNN，RNN编码“what”并产生输出。
2. 对于下一步，我们提供单词“time”和上一步中的隐藏状态。RNN现在有关于“what”和“time”这两个词的信息。
3. 我们重复这个过程，直到最后一步。你可以通过最后一步看到RNN编码了前面步骤中所有单词的信息。



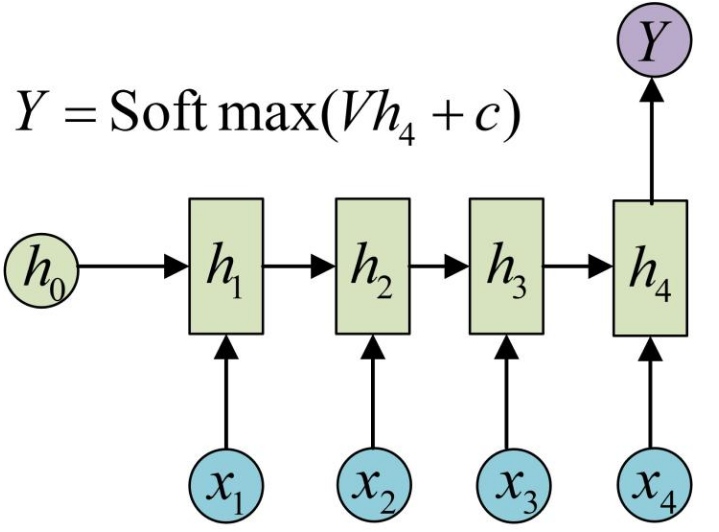
N vs 1 结构

对于x1, x2, x3, x4 四个单词来说





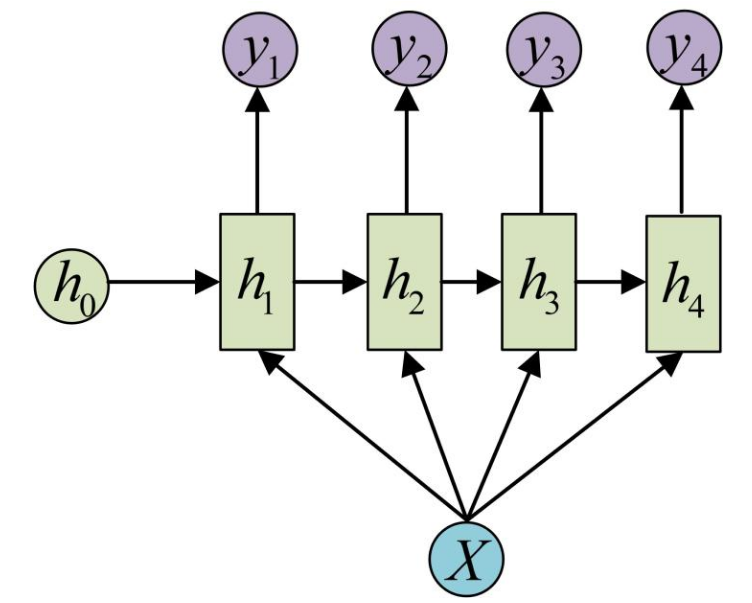
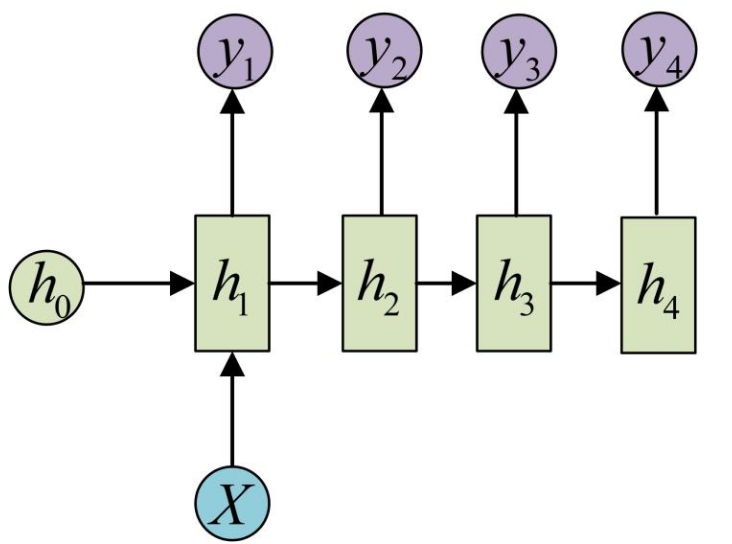
对于预测某个时间点的序列任务时，输出是一个单独的值而不是序列



这种结构通常用来处理序列分类问题。如输入一段文字判别它所属的类别，输入一个句子判断其情感倾向，输入一段视频并判断它的类别,股票预测等等。

1 vs N 结构

输入不是序列而输出为序列的情况怎么处理？我们可以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：

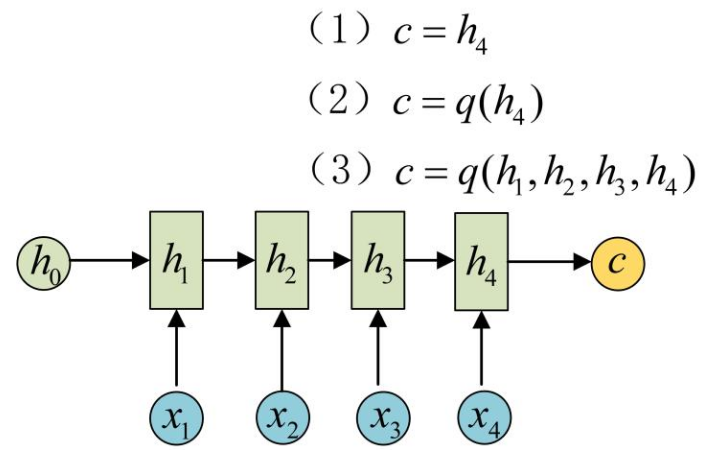
从图像生成文字（image caption），此时输入的X就是图像的特征，而输出的y序列就是一段句子

从类别生成语音或音乐等。

N vs M 结构

下面我们来介绍RNN最重要的一个变种：N vs M。这种结构又叫Encoder-Decoder模型，也可以称之为Seq2Seq模型。

原始的N vs N RNN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。为此，Encoder-Decoder结构先将输入数据编码成一个上下文向量c



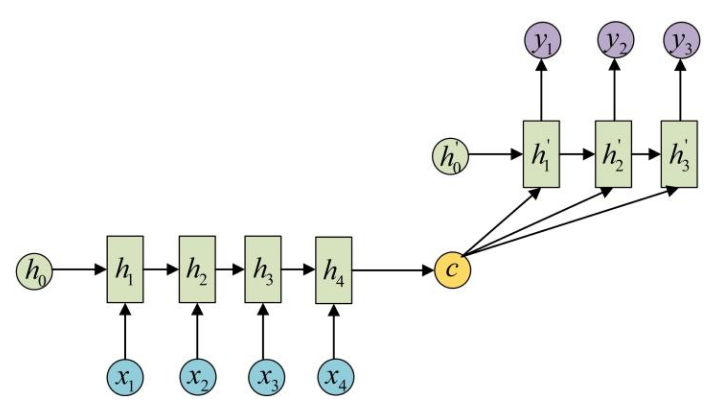
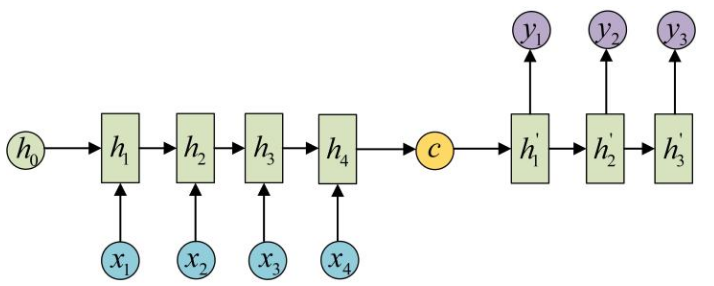
这一个过程就是Encoder，得到c有多种方式，

最简单的方法就是把Encoder的最后一个隐状态赋值给c，c=h4

还可以对最后的隐状态做一个变换得到c，c=q(h4)

也可以对所有的隐状态做变换。 c=q(h1,h2,h3,h4)

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h0输入到Decoder中，或者将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，比如：

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

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。

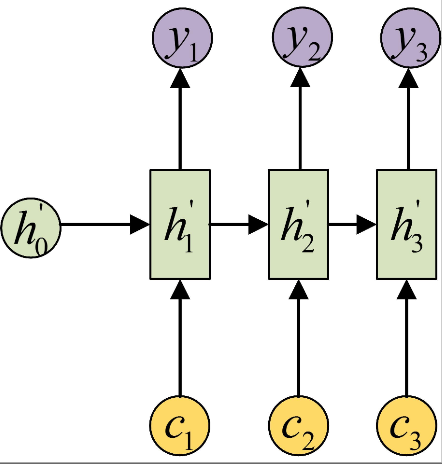
阅读理解。将输入的文章和问题分别编码，再对其进行解码得到问题的答案。

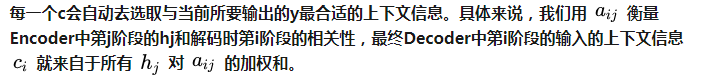
语音识别。输入是语音信号序列，输出是文字序列。

### Attention机制

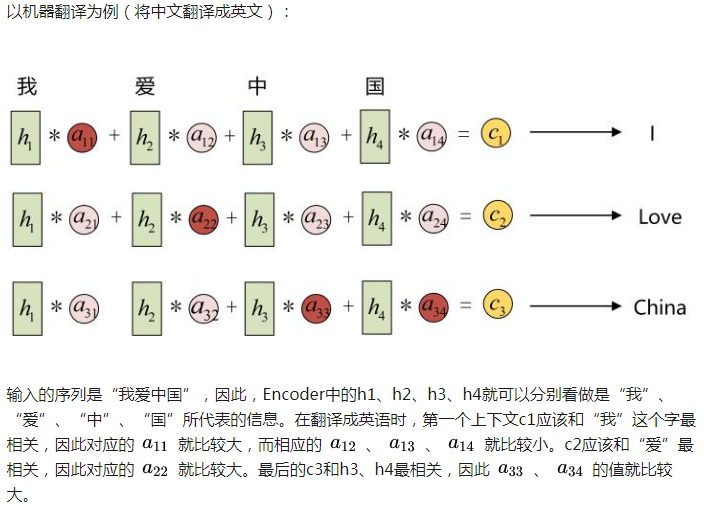
在Encoder-Decoder结构中，Encoder把所有的输入序列都编码成一个统一的语义特征c再解码，因此， c中必须包含原始序列中的所有信息，它的长度就成了限制模型性能的瓶颈。如机器翻译问题，当要翻译的句子较长时，一个c可能存不下那么多信息，就会造成翻译精度的下降。

Attention机制通过在每个时间输入不同的c来解决这个问题，下图是带有Attention机制的Decoder：

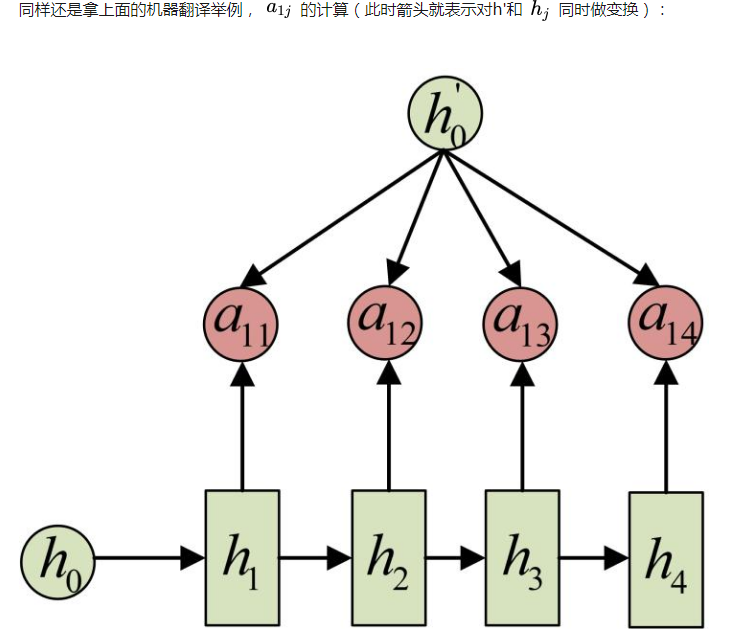




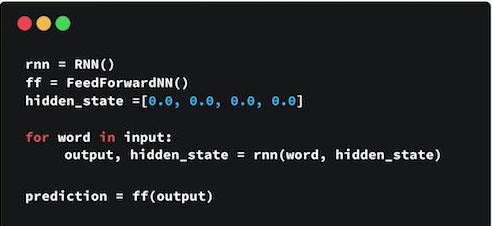
也就是说，在decode层输入Ci不在是全局信息而是与该节点最相关的信息。



事实上， aij 同样是从模型中学出的，它实际和Decoder的第i-1阶段的隐状态、Encoder第j个阶段的隐状态有关。



使用python展示了控制流程应该是最好的方式。



梯度消失

你可能已经注意到隐藏状态中奇怪的颜色分布。这是为了说明RNN被称为短期记忆的问题。



短期记忆问题是由臭名昭着的梯度消失问题引起的，这在其他神经网络架构中也很普遍。由于RNN处理很多步骤，因此难以保留先前步骤中的信息。正如你所看到的，在最后的时间步骤中，“what”和“time”这个词的信息几乎不存在。短期记忆和梯度消失是由于反向传播的性质引起的，反向传播是用于训练和优化神经网络的算法。为了理解这是为什么，让我们来看看反向传播对深度前馈神经网络的影响。

训练神经网络有三个主要步骤。

首先，它进行前向传递并进行预测。

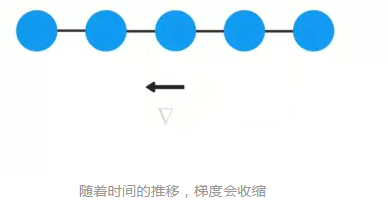
其次，它使用损失函数将预测与基础事实进行比较。损失函数输出一个错误值，该错误值是对网络执行得有多糟糕的估计。

最后，它使用该误差值进行反向传播，计算网络中每个节点的梯度。

梯度是用于调整网络内部权重的值从而更新整个网络。梯度越大，调整越大，反之亦然，这也就是问题所在。在进行反向传播时，图层中的每个节点都会根据渐变效果计算它在其前面的图层中的渐变。因此，如果在它之前对层的调整很小，那么对当前层的调整将更小。

这会导致渐变在向后传播时呈指数级收缩。由于梯度极小，内部权重几乎没有调整，因此较早的层无法进行任何学习。这就是消失的梯度问题。

让我们看看这如何适用于递归神经网络。你可以将循环神经网络中的每个时间步骤视为一个层，也就是说时间序列数据中，每一个样本都创建了一个神经元。为了训练一个递归神经网络，你使用了一种称为通过时间反向传播的方法。这样梯度值在每个时间步长传播时将呈指数级收缩。



由于梯度消失，RNN不会跨时间步骤学习远程依赖性。这意味着在尝试预测用户的意图时，有可能不考虑“what”和“time”这两个词。然后网络就可能作出的猜测是“is it？”。这很模糊，即使是人类也很难辨认这到底是什么意思。因此，无法在较早的时间步骤上学习会导致网络具有短期记忆。