循环神经网络（RNN）

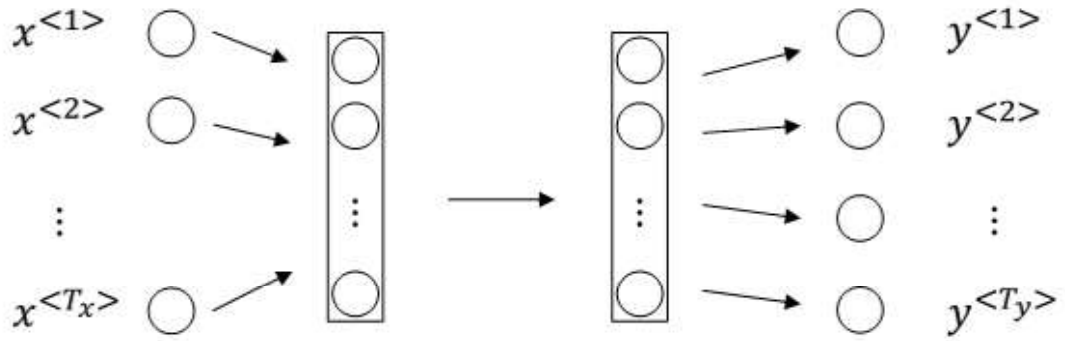
本文将介绍另一种功能强大、应用非常广泛的神经网络模型：循环神经网络（Recurrent Neural Networks，RNN）。

为什么使用 RNN

生活中，我们会遇到许多序列信号，例如一段语音、一段文字、一首音乐，等等。这些序列信号都有一个共同的特点：某一点的信号跟它之前或之后的某些信号是有关系的。比如，当我们在理解一句话意思时，孤立的理解这句话的每个词是不够的，我们需要处理这些词连接起来的整个序列；当我们处理视频的时候，我们也不能只单独的去分析每一帧，而要分析这些帧连接起来的整个序列。

那么，为了更好地解决序列信号问题，例如语音识别、机器翻译、情感分类、音乐发生器等，需要构建一种新的神经网络模型，RNN 就是这样的序列模型。

是否可以使用传统的神经网络模型呢？对于序列模型，如果使用标准的神经网络，其模型结构如下：



上图中，$x^{<1>}$、$x^{<2>} \cdots x^{<T_x>}$ 是序列模型的输入，即序列信号，T_x 表示输入信号的长度，例如一段文字的长度、一段语音包含的单词数目。$y^{<1>}$、$y^{<2>} \cdots y^{<T_y>}$ 是序列模型的输出，T_y 表示输出信号的长度。

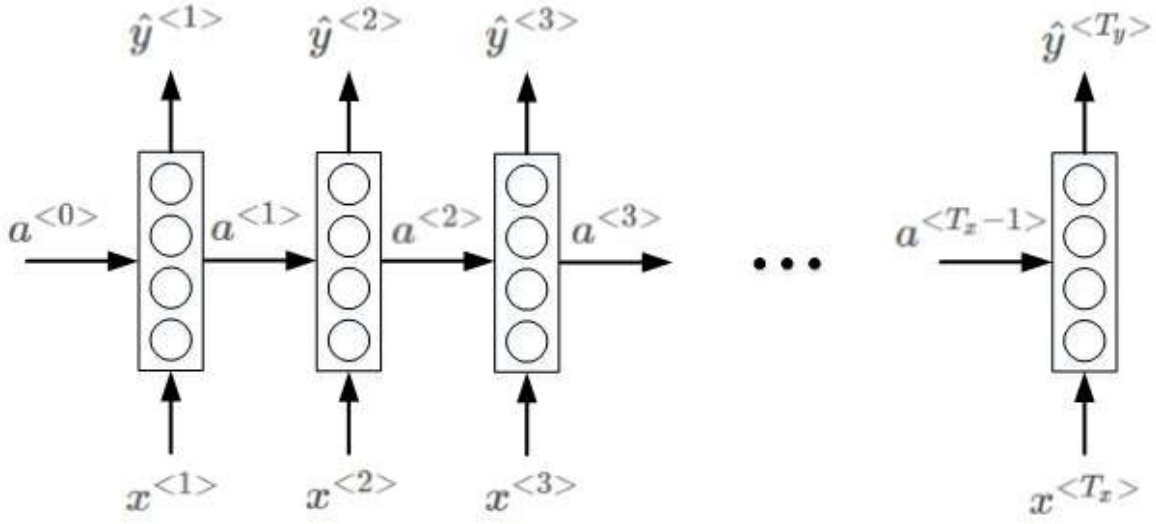
如果使用标准的神经网络结构来处理，存在两个问题。

第一，不同样本的输入序列长度或输出序列长度可能不同，即 $T_x^{(i)} \neq T_x^{(j)}$ ，$T_y^{(i)} \neq T_y^{(j)}$ ，比如无法确定两句话包含的单词数目一样，这就造成模型难以统一。可能的解决办法是设定一个最大序列长度，对每个输入和输出序列补零并统一到最大长度。但是这种做法的实际效果并不理想而且比较繁琐。

第二，也是最主要的，这种标准神经网络结构无法共享序列不同 $x^{<t>}$ 之间的特征。例如一句话中，如果某个 $x^{<t>}$ 是“李雷”，表示人名，那么句子其它位置出现了“李雷”，很可能也是人名。这是共享特征的结果，如同 CNN 的特点一样。但是，上图所示的标准神经网络不具备共享特征的能力。

基本的 RNN 模型

知道了标准神经网络处理序列信号问题的弊端之后，我们再来看看 RNN 的结构。



首先，我们需要知道的是 RNN 输入信号的编码问题。我们知道序列信号可能是一段文字，也可能是一段语音，文字如何量化成数字的信号呢？最常用的方式就是建立一个词汇表（例如 1000 个单词），每个单词使用 One-Hot 形式进行编码。这样，每个单词就由 1000x1 的向量组成。该向量对应词汇表顺序，相应单词对应位置为 1，其它位置为 0。

举个简单的例子，看下面这句话：

> **My name is Tom.**

对于这个输入序列信号，$x^{<1>}$=Tom，$x^{<2>}$=name，$x^{<3>}$=is，$x^{<4>}$=Tom，$T_x = 4$。

如果我们建立一个词汇表，词汇表大小是 10，上面这句话中的四个单词 “My”、“name”、“is”、“Tom” 分别位于词汇表的第 0、3、6、9 位置处。使用 One-Hot 编码，得到的各个输入信号为：

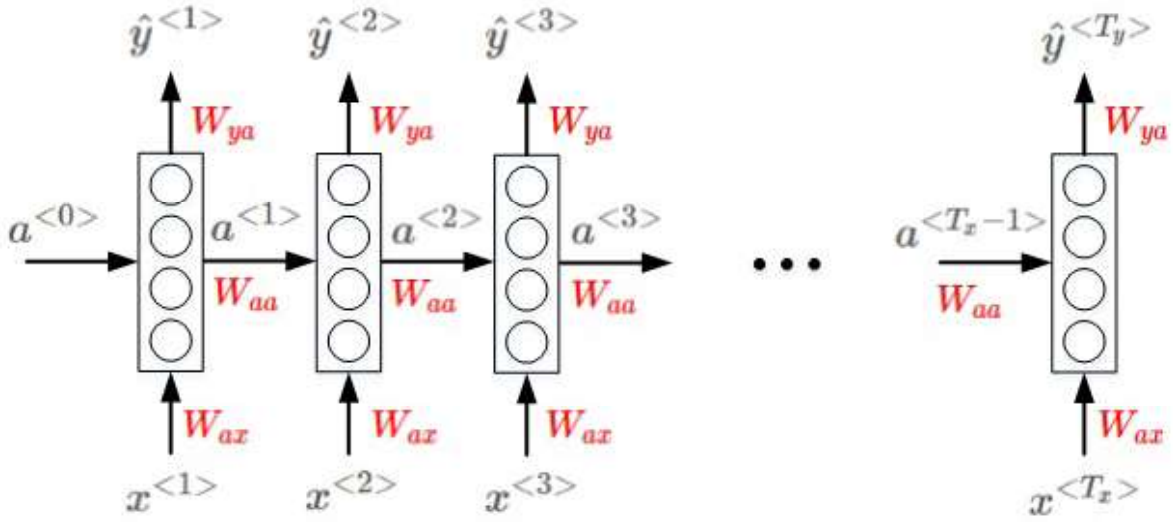
$$x^{<1>} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad x^{<2>} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad x^{<3>} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad x^{<4>} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

如果出现词汇表之外的单词，可以使用 UNK 或其他字符串来表示。对于多样本，以上输入信号对应的命名规则可表示为：$X^{(i)<t>}$，$Y^{(i)<t>}$，$T_x^{(i)}$，$T_y^{(i)}$。其中，i 表示第 i 个样本。不同样本的 $T_x^{(i)}$ 或 $T_y^{(i)}$ 都有可能不同。

回过头来看 RNN 模型的基本结构，序列模型从左到右，依次传递，此例中，$T_x = T_y$。$x^{<t>}$ 到 $\hat{y}^{<t>}$ 之间是隐藏神经元（类似于标准神经网络中的隐藏层）。注意模型中的 $a^{<t>}$ 是记忆单元，它是当前模块的输出，同时也作为下一个模块的输入。这样就能实现将序列信号中单个信号的信息传递给后面的信号，实现信号的记忆和传递功能，建立序列信号前后联系。其中，$a^{<0>}$ 一般为零向量。

RNN 正向传播

RNN 模型包含三类权重系数，分别是 W_{ax}，W_{aa}，W_{ya}。且不同元素之间同一位置共享同一权重系数。这样做的优点是模型参数与序列信号长度无关。



上图展示了一个包含单隐藏层的 RNN 模型，正向传播（Forward Propagation）过程的表达式如下：

$$a^{<t>} = g(W_{aa} \cdot a^{<t-1>} + W_{ax} \cdot x^{<t>} + b_a)$$

$$\hat{y}^{<t>} = g(W_{ya} \cdot a^{<t>} + b_y)$$

其中，$g(\cdot)$ 表示激活函数，不同的问题需要使用不同的激活函数。b_a 表示输入层到隐藏层的常数项，b_y 表示隐藏层到输出层的常数项。

为了简化表达式，可以对 $a^{<t>}$ 项进行整合：

$$W_{aa} \cdot a^{<t-1>} + W_{ax} \cdot x^{<t>} = [W_{aa} \ W_{ax}] \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix}$$
$$\rightarrow W_a [a^{<t-1>}, x^{<t>}]$$

则正向传播可表示为：

$$a^{<t>} = g(W_a [a^{<t-1>}, x^{<t>}] + b_a)$$

$$\hat{y}^{<t>} = g(W_y \cdot a^{<t>} + b_y)$$

RNN 反向传播

RNN 模型的损失函数与其他机器学习模型类似。如果是分类问题，则可使用交叉熵损失；如果是回归问题，则可使用 MSE、MAE 或 Huber 损失。此处，以分类问题为例，RNN 模型中单个元素的损失函数为：

$$L^{<t>}(\hat{y}^{<t>}, y^{<t>}) = -y^{<t>} log\, \hat{y}^{<t>} - (1 - y^{<t>}) log\, (1 - \hat{y}^{<t>})$$

该样本序列信号所有元素的损失函数为：

$$L(\hat{y}, y) = \sum_{t=1}^{T_y} L^{<t>}(\hat{y}^{<t>}, y^{<t>})$$

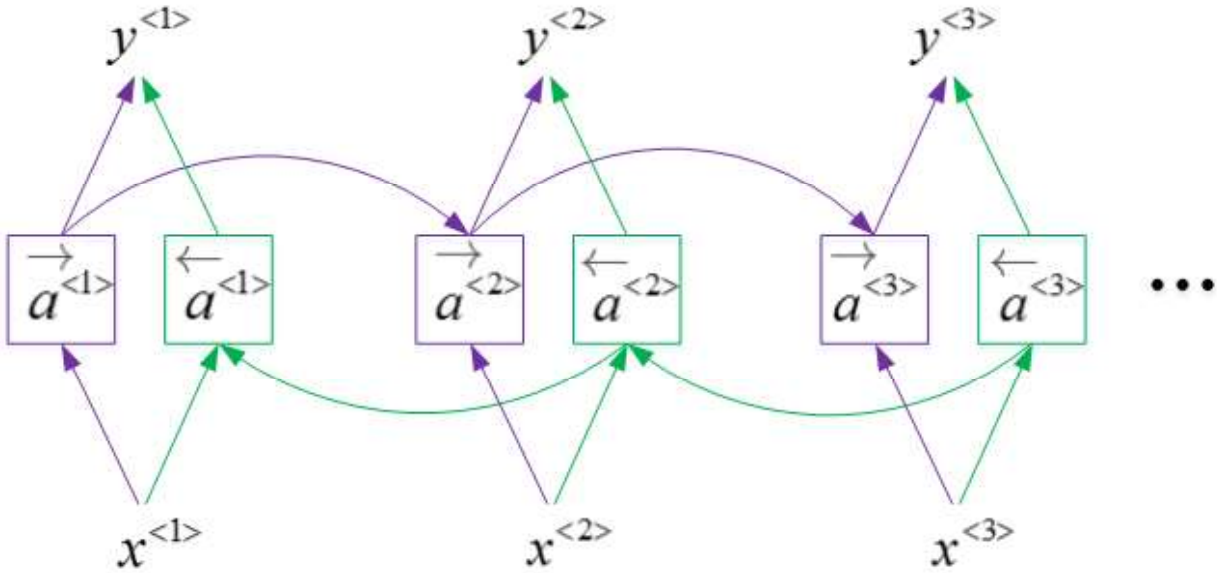
然后，与其他机器学习模型类似，使用梯度优化算法，对参数 W_a、W_y、b_a、b_y 分别计算偏导数，并更新，直到训练完成。

RNN 多种结构

上面介绍的基本 RNN 模型中 $T_x = T_y$，但在很多 RNN 模型中，T_x 是不等于 T_y 的。例如情感分类模型 $T_x \neq T_y$。一般根据 T_x 与 T_y 的关系，RNN 模型包含以下几个类型：

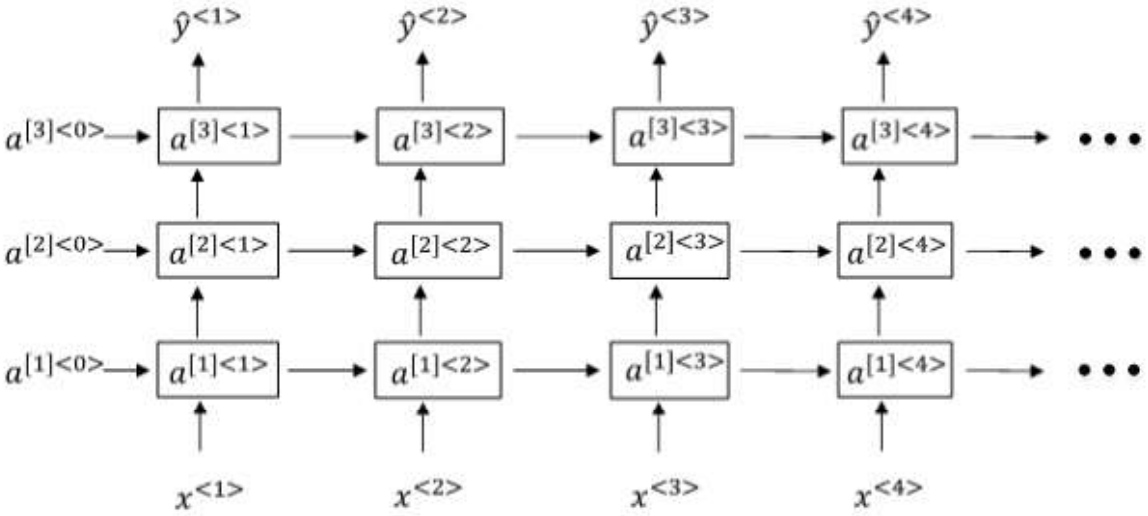
- Many to Many：$T_x = T_y$
- Many to Many：$T_x \neq T_y$
- Many to One：$T_x > 1, T_y = 1$
- One to Many：$T_x = 1, T_y > 1$
- One to One：$T_x = 1, T_y = 1$

上面这个模型是单向 RNN 结构，即按照从左到右 $\hat{y}^{<t>}$ 只与左边的模块有关。但是，有时候 $\hat{y}^{<t>}$ 也可能也与右边元素有关，于是就有了 Bidirectional RNN。BRNN 的基本结构如下图所示：

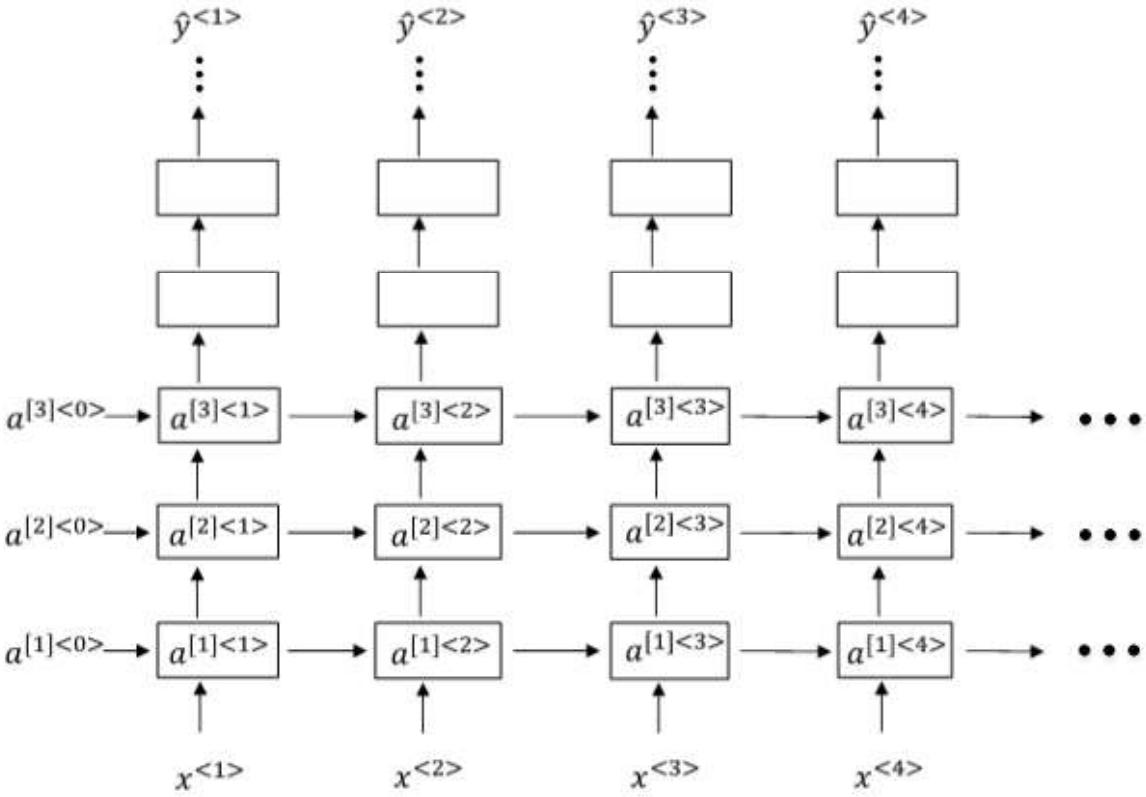


BRNN 能够同时对序列进行双向处理，性能大大提高。但缺点是计算量较大，且在处理实时语音时，需要等到完整的一句话结束时才能进行分析。

除了单隐藏层的 RNN 之外，还有多隐藏层的 Deep RNNs。Deep RNNs 由多层 RNN 组成，其结构如下图所示：



另外一种 Deep RNNs 结构中，每个输出层上还有一些垂直单元，如下图所示：



我们知道 DNN 层数可能超过 100，而 Deep RNNs 一般没有那么多层，三 层 RNNs 已经很复杂了。单层的 RNN 也有比较不错的效果。

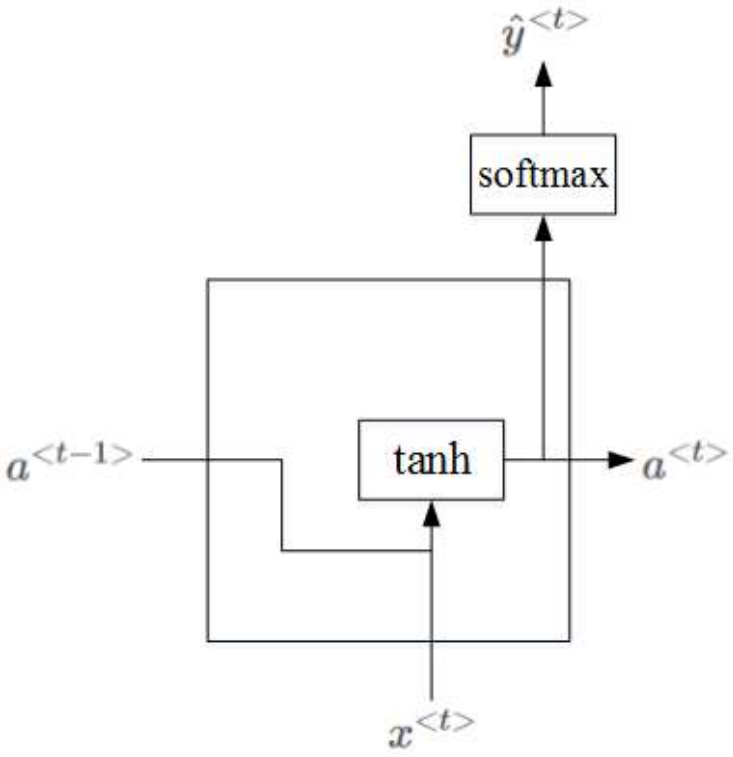
Gated Recurrent Unit（GRU）

序列信号中，可能存在跨度很大的词性依赖关系。例如下面这个例子：

> **The child, which already ate candy, was happy.**
>
> **The children, which already ate candy, were happy.**

第一句话中，was 受 child 影响；第二句话中，were 受 children 影响，它们之间的间隔较远。一般的 RNN 模型中，每个元素受其周围附近的影响较大，难以建立跨度较大的依赖性。上面两句话的这种依赖关系，由于跨度很大，普通的 RNN 模型就容易出现梯度消失，捕捉不到它们之间的依赖，造成语法错误。关于梯度消失和梯度爆炸我们在之前介绍神经网络的时候已经介绍过了，此处不再赘述。

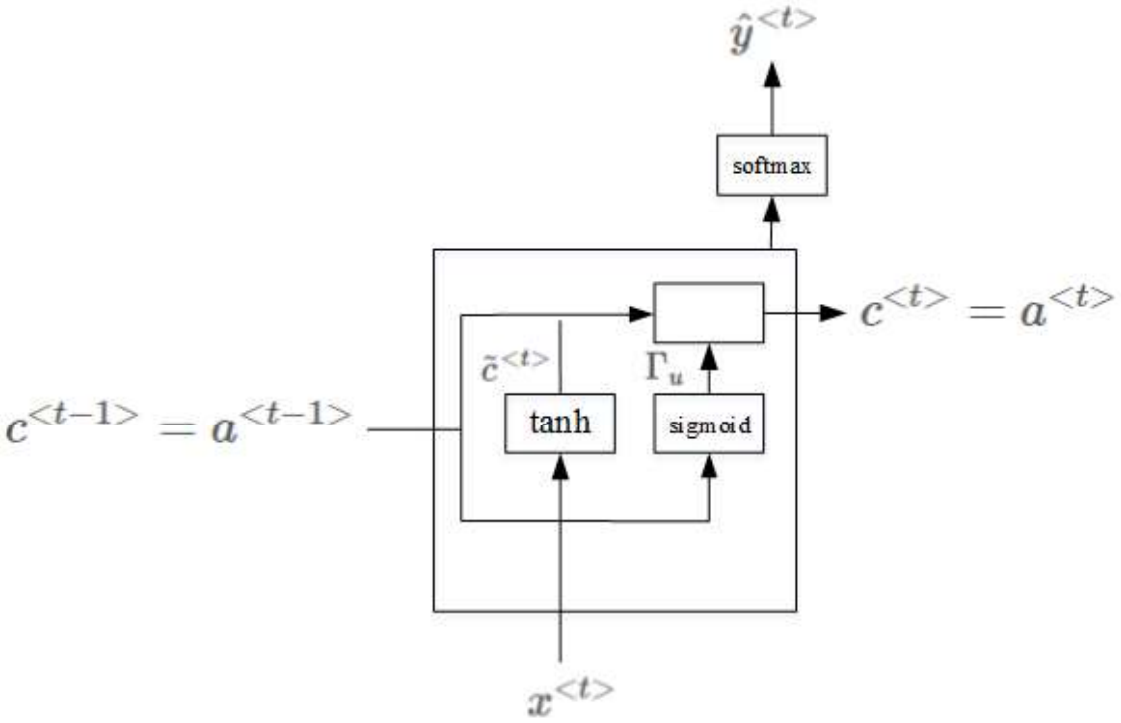
为了方便理解，我们首先细化下一般 RNN 隐藏层的结构：



其中，$a^{<t>}$ 的表达式为：

$$a^{<t>} = tanh(W_a[a^{<t-1>}, x^{<t>}] + b_a)$$

这个结构的缺点是阻断了 $a^{<t>}$ 与 $a^{<t-1>}$ 的直接性联系，相当于少了一个记忆单元，记住 $a^{<t-1>}$，这是造成梯度消失的主要原因。因此，为了解决梯度消失问题，对上述单元进行修改，添加了记忆单元。这种结构称之为 Gated Recurrent Unit（GRU），如下图所示：



相应的表达式为：

$$\tilde{c}^{<t>} = tanh(W_c[c^{<t-1>}, x^{<t>}] + b_c)$$

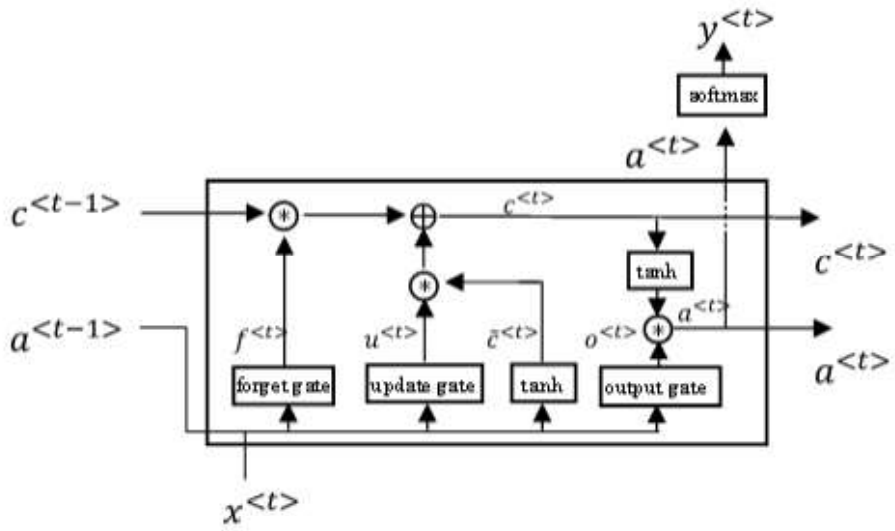
$$\Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$

其中，Γ_u 是一种 Gate，记忆单元。Γ_u 的取值范围在 [0,1] 之间。当 $\Gamma_u = 1$ 时，代表更新，$c^{<t>} = \tilde{c}^{<t>}$；当 $\Gamma_u = 0$ 时，代表记忆，$c^{<t>} = c^{<t-1>}$，保留之前的模块输出。所以，根据 Γ_u 的值，能够让模型自己决定是选择更新还是记忆。Γ_u 能够保证 RNN 模型中跨度很大的依赖关系不受影响，消除梯度消失问题。

Long Short Term Memory（LSTM）

比 GRU 更强大的模型是 LSTM。LSTM 称为长短期记忆模型，能有效处理 RNN 模型中“长期依赖”的问题。它的结构如下图所示：



LSTM 有 3 个 Gate，分别是 Forget Gate、Update Gate、Output Gate。$x^{<t>}$ 是输入，$c^{<t-1>}$ 和 $a^{<t-1>}$ 来自上一模块的输出。该 LSTM 单元相应的表达式为：

$$\tilde{c}^{<t>} = tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$$

$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$$

$$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$

$$a^{<t>} = \Gamma_o * c^{<t>}$$

其中，Γ_u、Γ_f、Γ_o 分别表示 Update Gate、Forget Gate、Output Gate 因子，取值范围均在 [0,1] 之间。这几个参数会在 RNN 模型训练过程中自适应调整记忆与更新的权重因子，已取得最佳的模型训练效果。

如果考虑 $c^{<t-1>}$ 对 Γ_u、Γ_f、Γ_o 的影响，可对 LSTM 的表达式进行修改：

$$\tilde{c}^{<t>} = tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}, c^{<t-1>}] + b_u)$$

$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}, c^{<t-1>}] + b_f)$$

$$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}, c^{<t-1>}] + b_o)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$

$$a^{<t>} = \Gamma_o * c^{<t>}$$

LSTM 是 RNN 的一个优秀的变种模型，继承了大部分 RNN 模型的特性，同时解决了神经网络反向传播过程由于逐步缩减而产生的梯度消失问题。具体到语言处理任务中，LSTM 非常适合用于处理与时间序列高度相关的问题，例如机器翻译、对话生成、编解码等。

总结

本文主要介绍了基本的循环神经网络 RNN 的基本结构。RNN 在处理序列信号方面具有更好的性能。然后讲解了 RNN 中非常重要的两种结构 GRU 和 LSTM，重点分析了这两种结构的特点。在机器翻译、对话生成等方面，LSTM 具有非常好的效果。

[help me with MathJax]