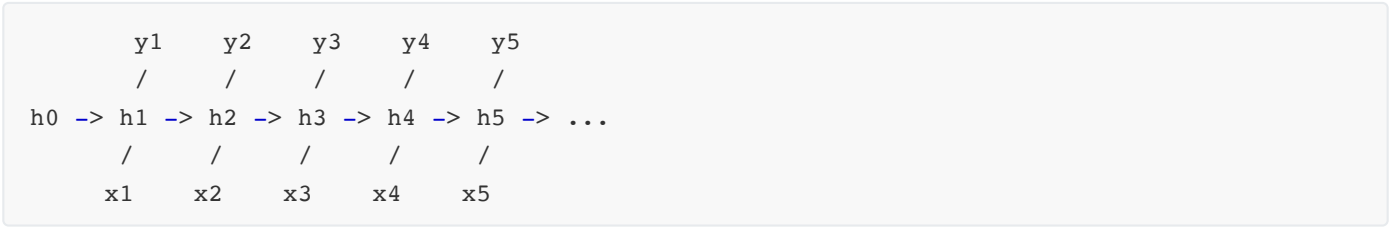
循环神经网络

常见的神经网络结构有多层感知机（MLP）、卷积神经网络（CNN）和循环神经网络（RNN）。MLP 和 CNN 是相对简单的，而 RNN 在处理序列数据时有独特的优势、也是比较难理解的模型。本文将介绍 RNN 的基本概念和应用。

RNN 的定义

假设我们现在有一个序列数据，比如一句话“I want to eat an apple”，我们把每个词看作一个时间点，那么这个序列就可以看作一个长度为 6 的时间序列，为了能够输入到神经网络中，我们还要把每个词转换成数字，然后通过 embedding 转化成向量，假设现在我们已经有了一个文本库，这里可能会包含很多不同的词，我们先通过分词统计出所有的词，比如说 {, , I, want, to, eat, an, apple, banana, time, event, scalar}，特别的 和 表示一段文本的开始和结束，现在我们有了所有的词，而后给它们分配一个类似于 id 的数字，比如说 {0: , 1: , 2: I, 3: want, 4: to, 5: eat, 6: an, 7: apple, 8: banana, 9: time, 10: event, 11: scalar}，这样我们就可以把文本转换成数字序列了，比如 “I want to eat an apple” 就可以转换成 $(2, 3, 4, 5, 6, 7)$，需要注意，这句话的每个词的数字都介于 0 到 11 之间，也就是从 0 到所有词的数量减一，这样我们就可以把文本转换成数字序列了，这个过程叫做词嵌入（word embedding）。对于一个长度为 T 的序列，通过前面说的方法，我们就可以得到一个长度为 T 的数字序列 $(x_1, x_2, \cdots, x_T)$，其中 $x_t \in \{0, 1, \cdots, V-1\}$，$V$ 是词库的大小。

但是一个词只有一个数字 x_t 来表示是不够的，我们希望将它转换成一个向量 $\mathbf{x}_t \in \mathbb{R}^D$，这样我们就可以把一个 scaler 序列 $(x_1, x_2, \cdots, x_T)$ 转换成一个词向量序列 $(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_T)$（我们始终用粗体表示列向量），其中 $\mathbf{x}_t \in \mathbb{R}^D$，$D$ 是词向量的维度，这个过程叫做词嵌入（word embedding）。词嵌入的方法有很多，比如 word2vec、GloVe 等，我们选择一个很简单的方法，假设有个 $V \times D$ 的矩阵 W，这是一个有待通过反向传播训练的矩阵，我们把每个词的 id 作为索引，然后从 W 中取出对应的行向量，比如 $\mathbf{x}_t = W[x_t]$，这样我们就得到了一个词向量的表示。这样我们就得到了词向量序列 $(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_T)$。现在我们已经有了 RNN 的输入，在每一个时间点 t，RNN 的输出 $\mathbb{y}_t \in \mathbb{R}^D$ 是由当前时间点的输入 $\mathbf{x}_t$ 和上一个时间点的隐藏状态 $\mathbf{h}_{t-1} \in \mathbb{R}^H$ 共同决定的，画图出来就是这样的：



图中 `h0` 是初始隐藏状态，`x1, x2, ..., x5` 是输入，`y1, y2, ..., y5` 是输出，对于每一个 RNN 单元，接受 $\mathbf{x}_t$ 和 $\mathbf{h}_{t-1}$，输出 $\mathbf{y}_t$ 和新的隐藏状态 $\mathbf{h}_t$，这个过程可以用下面的公式表示：

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1}) = \tanh\left(W_x \mathbf{x}_t + W_h \mathbf{h}_{t-1} + \mathbf{b}_h\right), \tag{1}$$

这里 $W_x \in \mathbb{R}^{H \times D}$ 和 $W_h \in \mathbb{R}^{H \times H}$ 是权重矩阵，$\mathbf{b}_h \in \mathbb{R}^H$ 是偏置向量，f 是激活函数，常用的有 `tanh`、`ReLU` 等，我们这里选择了 `tanh` 来激活。新的隐藏状态 $\mathbf{h}_t$ 会被输入到下一个时刻的 RNN 单元中，同时也会被用来计算当前时刻的输出 $\mathbf{y}_t$：

$$\mathbf{y}_t = W_y \mathbf{h}_t + \mathbf{b}_y, \tag{2}$$

其中 $W_y \in \mathbb{R}^{D\times H}$ 把 $\mathbf{h}_t$ 投影到输出空间 $\mathbb{R}^D$，然后我们再通过损失函数来计算预测值和真实值之间的差距，也就是我们希望 y_t 能够尽可能接近真实值，比如在预测下一个词的问题中，当我们输入了前面的几个词，我们希期模型能够预测出下一个词，在此任务中，我们希望每个 $\mathbf{y}_t$ 能够接近真实的 $\mathbf{x}_{t+1}$，而在下面的小节图片标题生成中，我们希望输出序列 $(\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_T)$ 能够接近真实的标题。损失函数的选择很多，比如均方误差（MSE）、softmax 交叉熵等，我们在这里介绍一个常用的损失函数——softmax 交叉熵损失函数，它的定义如下：

$$\mathcal{L} = -\sum_{t=1}^{T} \log \frac{e^{\mathbf{y}_{t,x_{t+1}}}}{\sum_{j=1}^{V} e^{\mathbf{y}_{t,j}}}, \tag{3}$$

softmax 的想法是希望 t 时刻的输出 $\mathbf{y}_t$（请牢记 $\mathbf{y}_t$ 的维度为 V 词库长度）能够接近真实的词 x_{t+1}，也就是 $\mathbf{y}_t$ 的第 x_{t+1} 个元素的值越大越好，而其他元素的值越小越好，这样我们就可以通过最小化交叉熵损失函数来训练 RNN 模型。

RNN 的构建、训练与预测

长短期记忆网络

应用——图片标题生成