5. 序列模型

`Deeplearning.ai深度学习课程`

---

1. 循环神经网络

1.1 循环神经网络应用领域

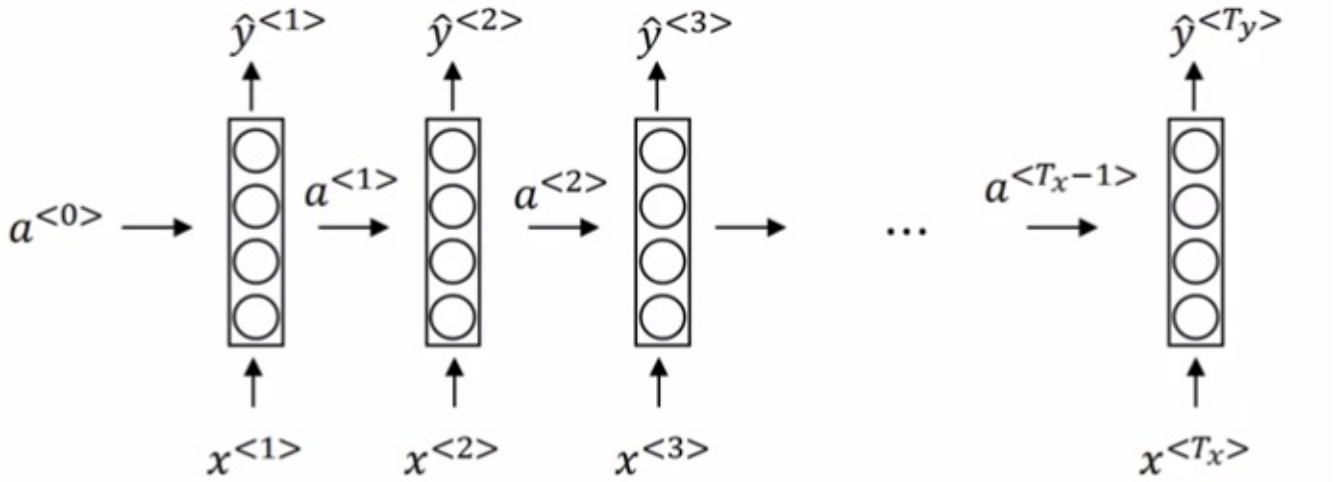
- 语音识别
- 音乐生成
- 文本情感分析
- DNA序列分析
- 机器翻译
- 录像动作识别
- 命名实体识别

1.2 循环神经网络

时间符号表示

对于一个特征序列x或者标签序列y，$x^{<t>}$ 或$y^{<t>}$ 表示序列中的第t个元素，T_x 和T_y 表示序列长度。如果是第i个样本数据的第t个元素，则用$x^{(i)<t>}$ 来表示。

前向传播



当到达隐藏时间节点t时，隐藏层不仅接收该节点的数据$x^{<t>}$，也包含上一层的激活结果$a^{<t-1>}$，同时每个时间节点也可能输出结果$y^{<t>}$。初始激活结果$a^{<0>}$可以为零向量。$x^{<t>}$、$a^{<t>}$、$y^{<t>}$所对应的权重矩阵分别为w_{ax}、w_{aa}和w_{ay}，在每一个时间节点处，权重矩阵都是相同的，也就是说参数是共享的。假设有n_x个特征，m个样本，则$x^{<t>}$的形式为(n_x, m)，对应的特征矩阵w_{ax}形式为(n_a, n_x)；$a^{<t-1>}$的形式为(n_a, m)，对应的特征矩阵w_{aa}形式为(n_a, n_a)；$y^{<t>}$的形式为(n_y, m)，对应的特征矩阵w_{ya}形式为(n_y, n_a)。

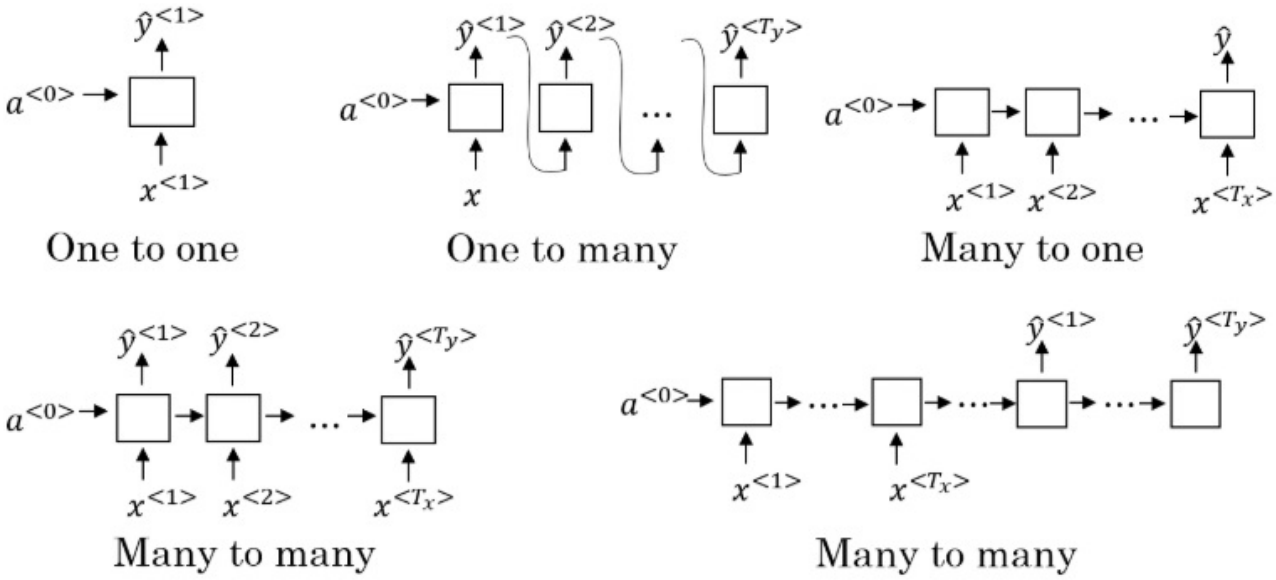
对于循环神经网络，前向传播的公式如下：

- $a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) = g_1(W_a[a^{<t-1>}, x^{<t>}] + b_a)$
- $\hat{y}^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$

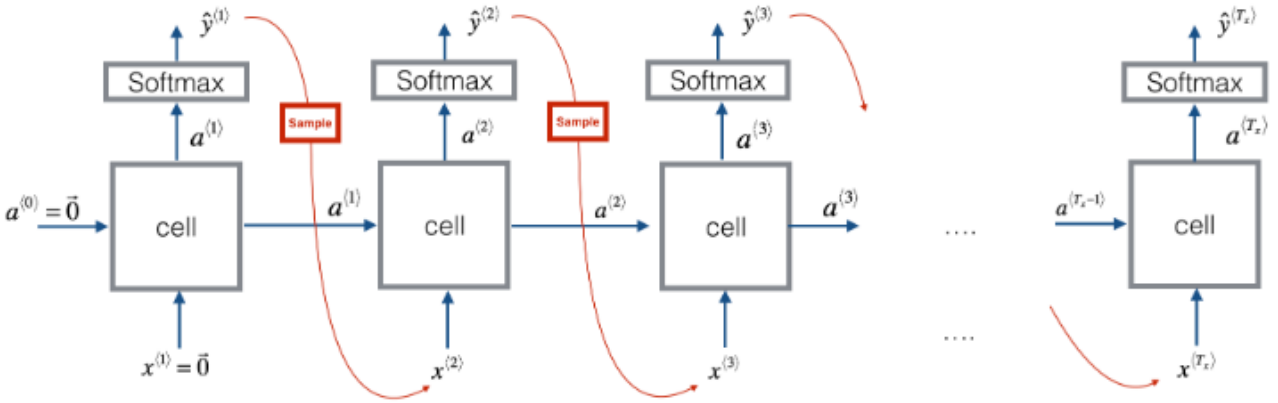
$W_a[a^{<t-1>}, x^{<t>}]$的形式是将两个矩阵堆叠起来，堆叠后的矩阵形式为$(n_x + n_a, m)$，对应的特征矩阵W_a形式为$(n_x + n_a, m)$。

不同类型的循环神经网络

- 一对一形式：相当于一般的神经网络
- 一对多：只在初始节点输入一次特征，之后生成多个y。应用：音乐生成、文本生成
- 多对一：每个节点都输入特征，只有最后一个节点输出结果。应用：文本分类
- 多对多(数量一样)：每个节点都输入特征，也生成结果。应用：命名实体识别
- 多对多(数量不一样)：前部分节点只输入，后部分节点只输出结果。应用：机器翻译



一种应用：文本生成



文本生成中应用到的循环神经网络是一对多形式，只需要向网络输入一个0向量，网络会计算出第一个节点的$\hat{y}^{<1>}$，并将其作为下一个节点的输入结果。重复此过程，可以生成一系列的y。

1.3 门控循环单元GRU

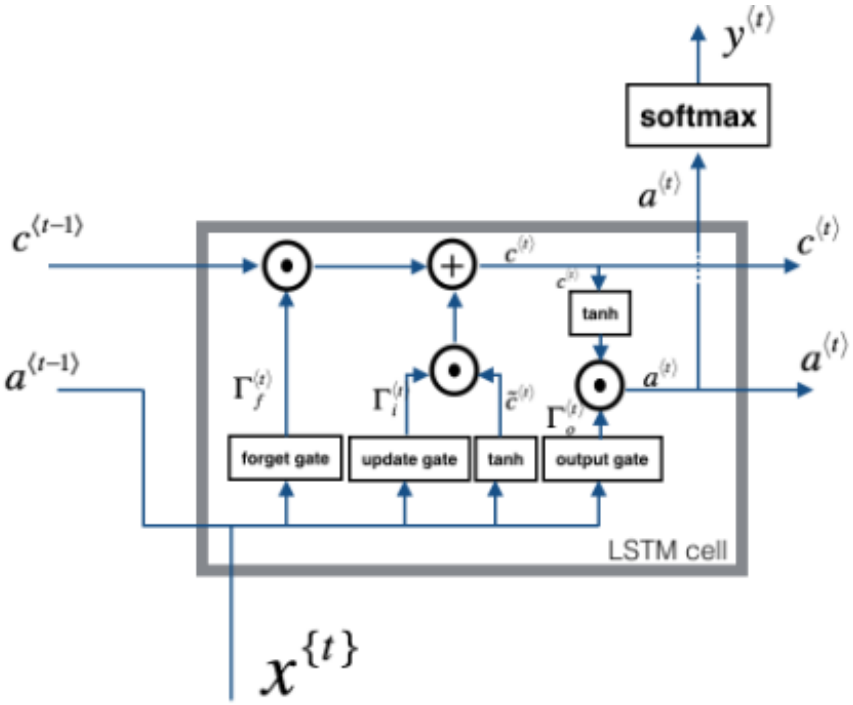
一般的RNN很难捕捉多个时间节点前的信息，主要是因传播过程中的梯度消失或梯度爆炸情况，因此模型也无法包含长期的记忆关系。

GRU单元一定程度上改进了这种情况。GRU中包含记忆细胞c，并通过更新门Γ_u来控制是否更新记忆细胞。c与RNN中的a比较类似，计算后的$c^{<t>}$也会赋给$a^{<t>}$，用来计算$y^{<t>}$等。

- 首先设置相关门：$\Gamma_r = \sigma(W_r[c^{<t-1>}, x^{<t>}] + b_r)$
- 记忆细胞的计算方法与RNN中的a相同：$\tilde{c}^{<t>} = tanh(W_c[\Gamma_r * c^{<t-1>}, x^{<t>}] + b_c)$
- 记忆门在0-1之间，决定了记忆强度：$\Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u)$

- 根据更新门的大小决定更新程度：$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$
- 激活后结果：$a^{<t>} = c^{<t>}$

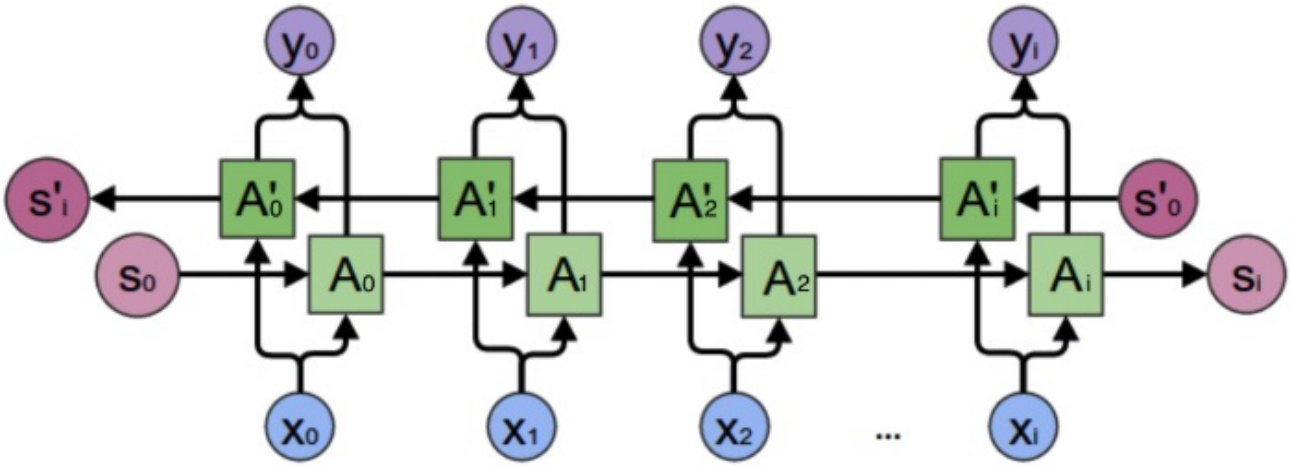
1.3 长短期记忆网络LSTM



LSTM也使用门来控制记忆强度，不过更新门和遗忘门是相互独立的。同时，LSTM中不直接把记忆细胞$c^{<t>}$赋予$a^{<t>}$，而是用另外的输出门来控制。另外，LSTM中没有相关门Γ_r。

- 更新门：$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$
- 遗忘门：$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$
- 新的记忆细胞：$\tilde{c}^{<t>} = tanh(W_c[c^{<t-1>}, x^{<t>}] + b_c)$
- 记忆强度由更新门和遗忘门共同控制：$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$
- 输出门：$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$
- 激活后结果：$a^{<t>} = \Gamma_o * tanh(c^{<t>})$

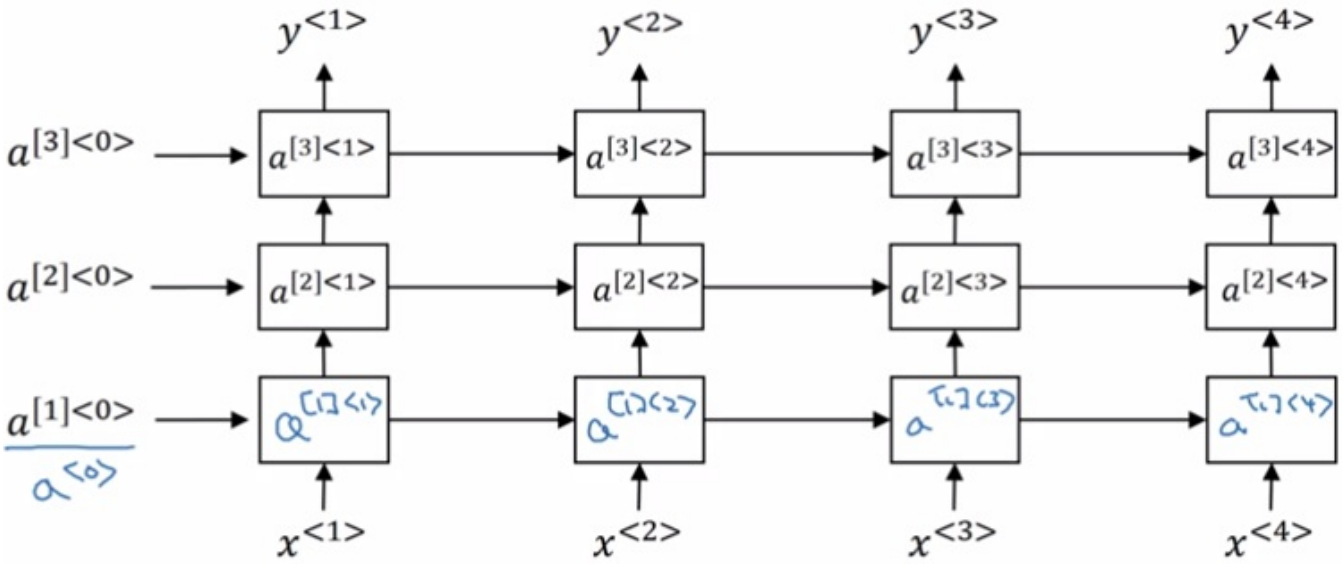
1.4 双向循环神经网络BRNN



$y^{<t>} = g(W_y[\overleftarrow{a}^{<t>}, \overrightarrow{a}^{<t>}] + b_y)$。

相比于只应用了之前时间节点信息的RNN，BRNN还可以应用到后面时间节点的信息。BRNN必须对完整的输入序列使用，例如音乐生成任务就是不行的。另外，BRNN的计算速度很慢。

1.5 深度循环神经网络DRNN



循环神经网络也可以设置为多层结构，此时除第一个隐藏层之外，其他层每个时间节点t的输入就不使用$x^{<t>}$，而是上一个隐藏层的计算结果$a^{[l-1]<t>}$。

对于某层l的第t个时间节点：$a^{[l]<t>} = g(W_a^{[2]}[a^{[l]<t-1>}, a^{[l-1]<t>}] + b_a^{[2]})$，也就是本层前一个时间节点的输入和前一层同一时间节点的输入。值得注意的是，不同层l的权重$W^{[l]}$是不同的。

深度神经网络不仅支持基本RNN单元，也可以使用LSTM单元、GRU单元或双向RNN。在实际应用中，DRNN的深度一般也只有几层，因为时间节点比较多，仅几层就会使计算变得很复杂

了。

2. 自然语言处理与词嵌入

2.1 词汇表示与词嵌入

最常见的词汇表示形式是One-Hot，但这种表示方法有不足之处：一是矩阵的维度会非常高，二是每个词向量之间是正交的，没有体现出词之间的相似性。
词嵌入是另一种词汇表示方法，可以把某个语义特征作为一个维度。如下图，假设有性别、皇室、年龄、事物等每个词在每个语义维度下都有一个数值，语义越接近的词，其词向量的相似性也就越高。

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97

2.2 词向量的类比推理

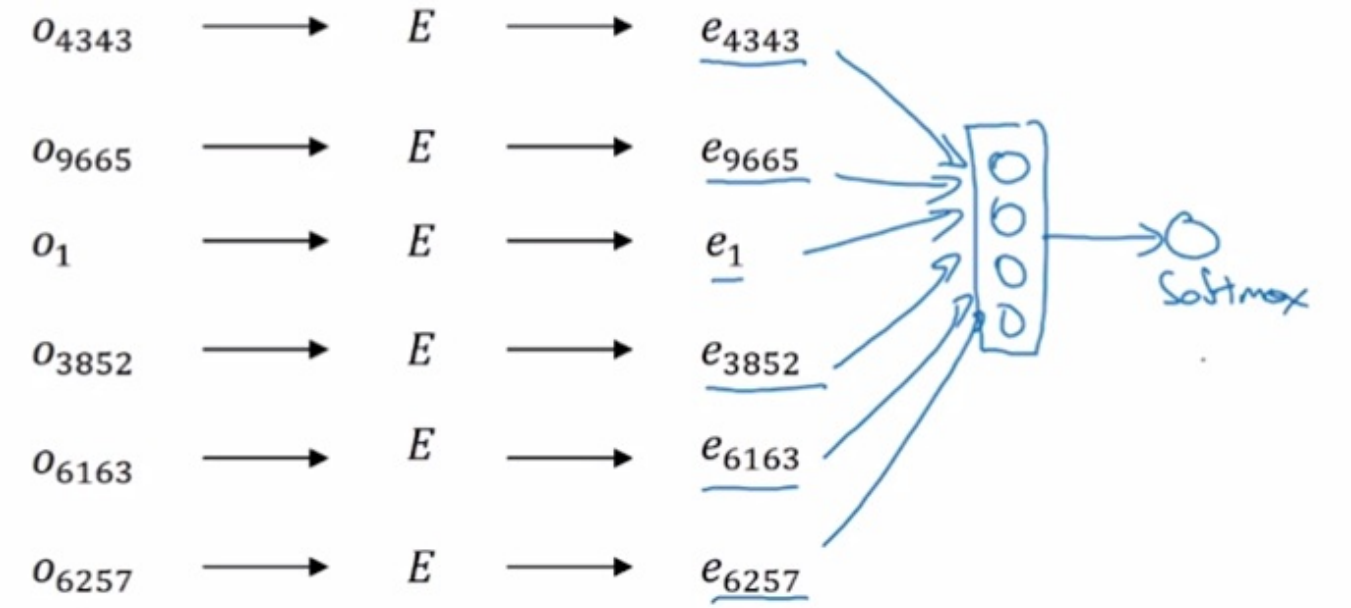
词向量的相似度一般用余弦相似性来衡量：$sim(\mu, v) = \frac{\mu^T v}{||\mu||_2 ||v||_2}$。
利用相似性可以进行类比推理，例如“男性”相对“女性”等于“国王”相对什么，即为$e_{man} - e_{woman} \approx e_{king} - e_w$，需要找到一个词向量$e_w$使得$sim(e_w, e_{king} - e_{man} + e_{woman})$最大。

2.3 词嵌入矩阵

假设有n*n的One-Hot矩阵O，其中有n个词；需要训练一个m*n的权重矩阵，m表示语义特征数。对于矩阵n中的每一个one-hot词向量o_i，对应的词向量$e_i = E \cdot o_i$。

2.4 学习词嵌入

一般来说，词向量的权重是通过学习一个语言模型来得到的。如下图，神经网络是希望训练一个语言模型来预测句子的下一个单词，而过程中的E就是学习到的词向量。当然，也可以使用其他的任务来学习词向量，比如已知上下文来预测目标词、根据前一个词预测下一个词等。



Word2Vec

Word2Vec是一种学习词向量的方法，其中包含Skip-Gram和CBOW模型(本课程只讲到了Skip-Gram)。

Skip-Gram模型是使用上下文内容去预测目标词。设某个词为c为上下文内容，在其一定词距内选取匹配的目标词t，以c为输入值，t为输出值建立神经网络，学习词向量权重。

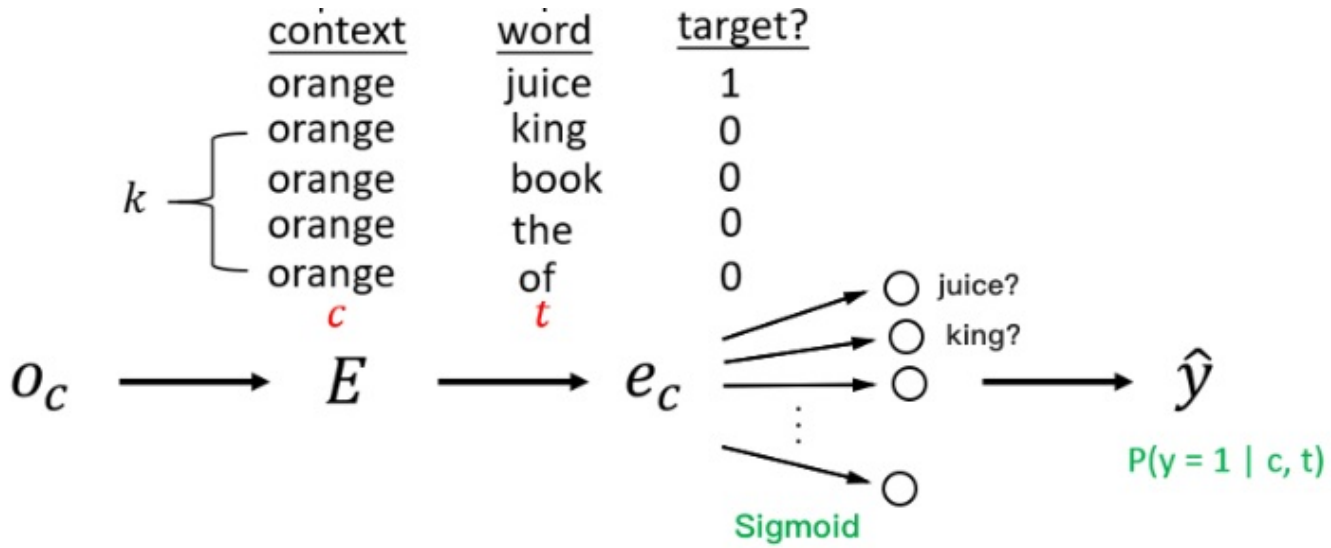
最终输出结果可以看作多分类问题：$p(t|c) = \frac{exp(\theta_t^T e_c)}{\sum_{j=1}^{m} exp(\theta_j^T e_c)}$；损失函数使用交叉熵：$L(\hat{y}, y) = -\sum_{i=1}^{m} y_i log\hat{y}_i$。

因为文本库中的词语一般比较多，多分类问题的类别也会很大，使计算速度很慢。解决的办法：1）使用分级的softmax分类器；2）负采样。

负采样

为了减少类别，可以将上下文与目标词设置为一个词对，而目标为0/1，表示是否匹配。一般会选取一对匹配词对作为正样本，再随机选取k个负样本。此时，模型由一个多分类任务，变成了k+1个二分类任务。输出为多个sigmoid单元格，对于每个单元格：

$P(y = 1|c, t) = \sigma(\theta_t^T e_c)$，$\theta_t$和$e_c$ 分别代表目标词和上下文的词向量。



在选取负样本的过程中，不能直接随机抽取，因为这样会导致the、a等常见词出现概率偏高。

对于每个词的选择概率，可以使用$p(w_i) = \frac{f(w_i)^{\frac{3}{4}}}{\sum_{j=0}^{m} f(w_j)^{\frac{3}{4}}}$来计算，其中$f(w_i)$表示每个词出现的频率。

GloVe

Glove是基于语料统计了词之间的共现矩阵X，矩阵中的每一个元素x_{ij}表示词i和词j组成上下文-目标词对的次数。

之后，用梯度下降法最小化损失函数：

$J = \sum_{i=1}^{N} \sum_{j=1}^{N} f(X_{ij})(\theta_i^t e_j + b_i + b_j - log(X_{ij}))^2$ 。其中，θ_t和e_c 分别代表目标词和上下文的词向量；f函数是一个加权项，用来避免 $X_{ij} = 0$时，$log(X_{ij})$为负无穷大(当$X_{ij} = 0, f(X_{ij}) = 0, 0 * log0 = 0$)，同时调整停止词或低频词的权重，使它们不至于过大或过小。

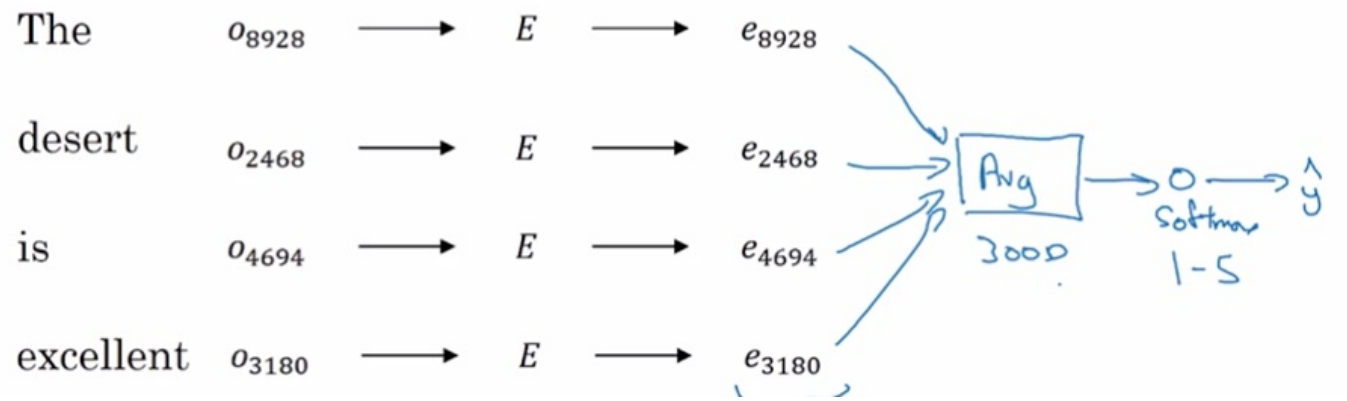
另外，θ_i 和 e_j是完全对称的。因此，在训练后，可以取θ_w和e_w的平均值来作为词向量的最终结果。

2.5 应用：情感分类

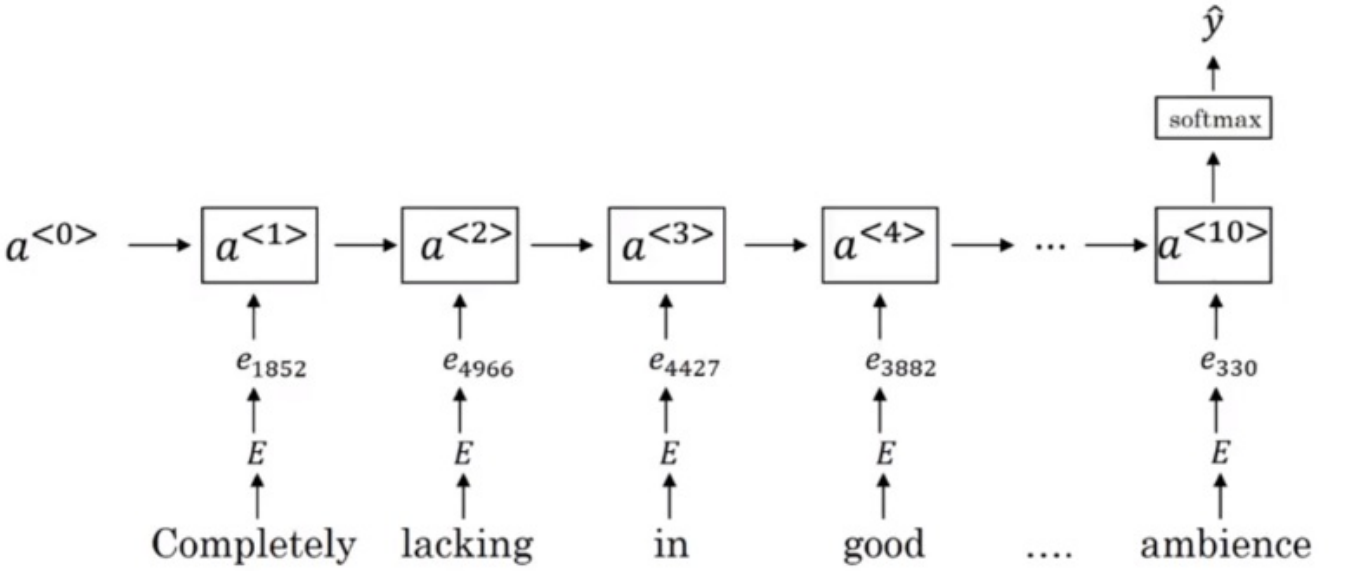
情感分类是自然语言处理技术一个应用场景，用来判断文本对某事物的态度是正面还是负面的(二分类)，或是对某事物的评分(多分类)。情感分类问题一般需要大量的数据才能完成，但有了词向量，我们使用相对小规模的数据，就可以达到不错的效果。

一种方法是将句子的One-Hot表示与词嵌入矩阵相乘，得到词向量，再将词向量求平均值，最后通过一个全连接层得到输出结果。这种方法很简单，且适用于任何长度的文本，但忽略了词

的顺序和上下文关系。



另一种方法是再得到词向量后，输入到循环神经网络层，将循环神经网络最后一个时间节点的结果输入全连接层进行预测。这种方法一般来说效果更好，但速度会慢一些。

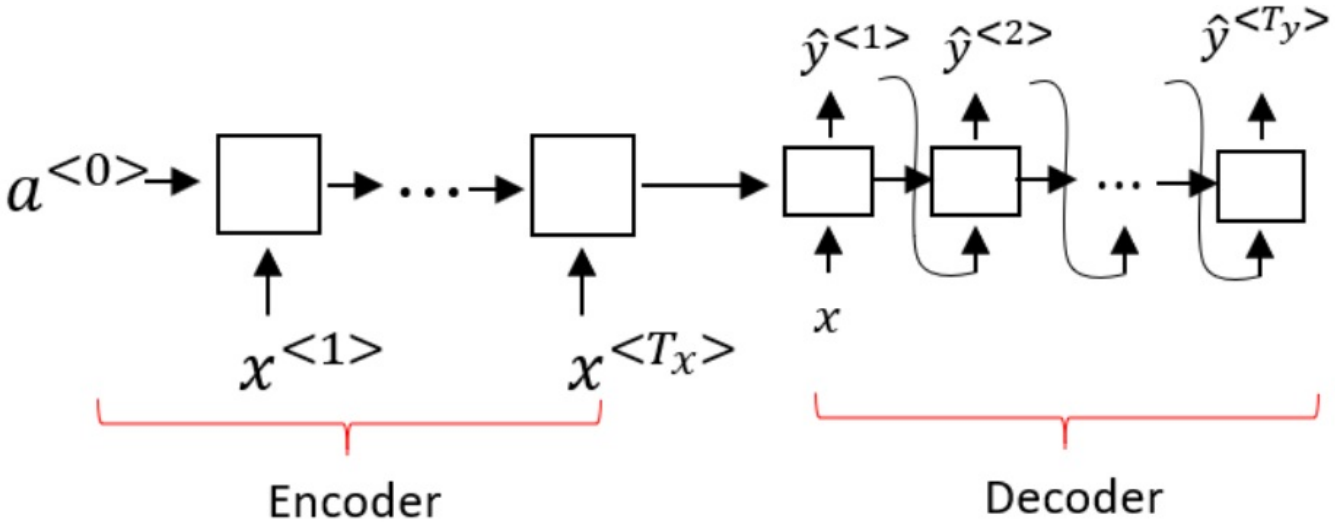


另外，当文本长度不同时，RNN的时间节点数难以统一。一种可行的方法是以最长的文本长度为标准设置时间节点，其他文本长度不足的部分设置为空，用0向量来代表其词向量。

3. 序列模型和注意力机制

3.1 seq2seq模型

seq2seq模型包括编码器和解码器两个部分，一般来说两者都是RNN模型。编码器用于处理输入数据，并将输入的编码结果传到解码器中，用解码器处理再进行输出。以seq2seq的常见应用机器翻译为例：编码器作用于被翻译的文本，处理结果输入解码器，解码器再输出翻译结果。



3.2 集束搜索

seq2seq模型也是希望能够使条件概率最大化的。以机器翻译为例，在给定输入文本序列x的条件下，我们希望能得到概率最大的翻译结果，即：argmax $P(y^{<1>},y^{<2>},\dots,y^{<T_y>}|x^{<1>},x^{<2>},\dots,x^{<T_x>})$。

集束搜索

求解序列最大条件概率时，最直接的方法是贪心算法，也就是首先求$P(y^{<1>}|X)$，再求$P(y^{<2>}|X,y^{<1>})$，直至最后得到$P(y^{<T_y>}|X,y^{<1>},\dots,y^{<T_{y-1}>})$。

集束搜索是在每个时间点时，不只选择概率最大的选择，而是同时考虑多个选择，个数叫做集束宽B。例如当B=3时，在每个时间点时保留概率最大的三个候选单词。贪心搜索其实是集束搜索的一个特殊情况，此时B=1。

长度标准化

对于argmax $\prod_{t=1}^{T_y}P(\hat{y}^t|x,\hat{y}^{<1>},\dots,\hat{y}^{<t-1>})$。因为公式涉及到多个小数相乘，因此最后的值会很小，造成数值下溢，也就是一个电脑不能表示出的极小数字。

因此，我们对公式进行log标准化：argmax $\frac{1}{T_y^\alpha}\sum_{t=1}^{T_y}logP(\hat{y}^t|x,\hat{y}^{<1>},\dots,\hat{y}^{<t-1>})$。

其中$\frac{1}{T_y}$是对较短的文本进行惩罚，α用来调整惩罚的强度。

对于集束宽B，B越大，搜索结果约准确，但计算成本也越大，通常B的取值在10以下，B越大边际作用就越小了。另外，集束搜索不一定能得到最准确的结果，但它的运行速度比BFS/DFS等精确搜索算法快很多。

误差分析

由于集束搜索不是完全准确，因此当我们产生误差时有两种可能：1）RNN的效果不好；2）集束搜索的效果不好。
为了分析误差原因，我们一般会人工翻译一次句子。假设人工翻译结果为$y*$，机器翻译结果为$\hat{y}$，我们将两者分别输入seq2seq模型进行比较。

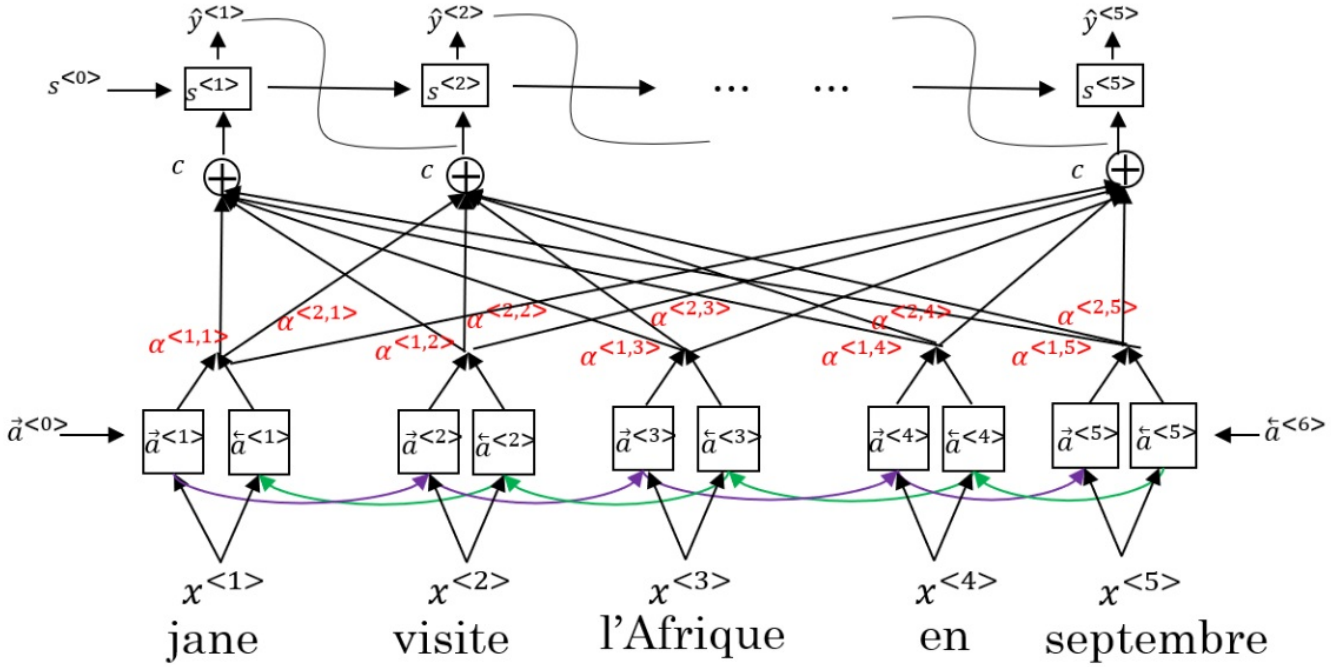
- 如果$P(y*|x) > P(\hat{y}|x)$，说明是集束搜索算法出现了问题，没有选到概率最大的词，需要增大集束宽
- 如果$P(y*|x) \leq P(\hat{y}|x)$，说明是RNN效果不佳，需要改进RNN模型设计

3.3 Bleu得分

Bleu是用于评估机器翻译结果的方法。Bleu得分的计算方法是：机器翻译的单词数目做分母；对于分子，用每个单词是否在标准翻译(一般为人工翻译结果)中出现来判断，如果单词在标准翻译中出现了n词，则机器翻译结果最终也只能计数n次。
上述方法是针对一元组的，当然也可以对句子中的n元组进行计算。多元Bleu得分计算为：$P_n = \frac{\sum_{n-gram \in \hat{y}} count_{clip}(n-gram)}{\sum_{n-gram \in \hat{y}} count(n-gram)}$。另外，需要对短结果进行惩罚：当机器翻译长度(MT)小于标准翻译长度(BM)时，$BP = exp(1 - \frac{MT}{BM})$，否则BP=1。
最终的Bleu得分为：$Bleu = BP * exp(\frac{1}{N} \sum_{i=1}^{N} log^{P_n})$

3.4 注意力模型

注意力机制模拟了人脑的翻译工作，因为我们在人工翻译过程中，不会一直同时关注整个句子，而是关注被翻译句子的局部。注意力机制是希望解码器在运行时不是同等对待编码序列的每一个时间点，而是把注意力放在某几个序列节点。



将编码器的节点用t'表示，解码器用t表示，我们希望对解码器中的每个时间节点t得到其对编码其中每个节点的“关注”权重$\alpha^{<t,t'>}$，α满足$\sum_{t'}\alpha^{<t,t'>}=1$。在获得权重$\alpha$的过程中，我们首先需要利用编码器得输出结果$a^{t'}$和上一个时间节点的解码器输出结果$s^{t-1}$来得到重$e^{<t,t'>}$，并利用softmax函数得到$\alpha^{<t,t'>}=\frac{exp(e^{<t,t'>})}{\sum_{t'=1}^{T_x}exp(e^{<t,t'>})}$。

再利用权重与编码器的输出结果$a^{t'}$相乘得到$c^{<t>}=\sum_{t'}\alpha^{<t',t>}a^{t'}$，即为输入给解码器的最终结果。

解码器在得到$c^{<t>}$后会计算出$s^{<t>}$，它不仅要作为输出结果，也要输入给编码器，协助计算出下一个节点的$e^{<t,t'>}$。