基于深度学习的自然语言处理

RandomStar

2021 年 5 月 18 日

摘要

图机器学习入门，基于《统计学习方法》和斯坦福大学 CS224W 等课程。

目录

1 自然语言处理

1.1 自然语言处理简介

自然语言处理是对人类语言的分析和处理，人类的语言 (也就是自然语言) 是一种专门为传达意思而构建的系统，并不是任何物理设备生成的，因此自然语言和图像或者其他一些机器学习任务有很大的不同。不管是什么自然语言，总是由一系列单词组成的，而这些单词中的大部分仅仅代表的是一种语言以外的实体，也就是说单词是一种能指 (signifier，即单词本身) 到所指 (signified，单词所代表的含义) 的映射。

自然语言处理建模的对象可以包括单词，句子，文章，文档等一系列包含自然语言的对象。

1.1.1 自然语言处理的任务

自然语言处理中有很多种不能难度等级的任务，但总的来说自然语言处理的目标是让计算机可以“理解”人类的语言，这些任务可以根据难度分成这样几个等级：

- Easy：拼写检查，关键词检索，找同义词
- Medium：解析复杂文档，包括 web 文档和其他结构化的文档
- Hard：机器翻译，语义分析，引用推断 (Coreference) 和自动问答

后面的内容将主要围绕基于深度学习方法来完成自然语言处理的各项任务。

1.1.2 如何表示单词

所有的自然语言都是由若干的单词或者说词汇组成的，NLP 的建模最重要的一个任务就是对单词进行建模，采用一定的方式来表示各种单词，因为后面介绍的大部分 NLP 任务都将单词看成一种 atomic symbols，也就是原子符号，即不可拆分的最基本单位，我们的首要任务就是完成对单词的建模。

我们可以使用向量来表示一个单词，将单词的特征编码到向量的一系列维度中，这也就是词向量 (Word Vector)

2 词向量 Word Vector

词向量也就是用向量来表示一个单词，但是语言的单词数量可能是非常庞大的，比如英语就有各类单词词组约 1300 万个，我们需要将这些单词全部编码到一个 N 维度的向量中去，向量的每个维度可以编码一些语义或者单词的特征。词向量也可以叫做词嵌入 $WordEmbedding$

2.1 one-hot 向量

一种非常不负责任的编码方式就是 one-hot 向量，这种方法根据当前样本数据的词汇表的大小 $|V|$，用一系列维度为 $|V|$ 的 0-1 向量来代表单词，词汇表中出现的每个单词 m 有一个对应的维度 v_m，每个单词对应的词向量中，该维度的值是 1，其他的维度都是 0，这种方式非常简单粗暴，但是问题也很明显，这样的编码是非常稀疏的，词向量中的大部分内容都是无效信息 0，并且词向量的规模也非常大，是对计算资源的严重浪费。

2.2 基于奇异值分解 SVD 的词向量

2.2.1 奇异值分解 SVD

根据我们仅存的一点线性代数的知识，我们知道矩阵可以分解成一系列特征向量和特征值，但是除此之外矩阵还可以进行奇异值分解 (Singular Value Decomposition, SVD) 是将矩阵分解成一系列奇异向量和奇异值，这种方法可以使我们得到一些和特征分解类似的信息，**但是相比于特征分解，奇异分解适用范围更广，所有的实矩阵都有一个奇异值分解但不一定有特征分解 (特征分解必须要是方阵)**。在特征分解中我们可以求出一系列特征向量使其构成一个矩阵 V 和一系列特征值构成对角矩阵 $\text{diag}(\lambda)$，这样一来一个矩阵就可以分解成：

$$A = V\text{diag}(\lambda)V^{-1} \tag{1}$$

而在奇异值分解中我们可以类似地将矩阵分解成三个矩阵的乘积：

$$A = UDV^{-1} \tag{2}$$

这里我们可以假设 A 是一个 $m \times n$ 维的矩阵那么 U 是一个 $m \times m$ 的矩阵，D 是一个 $m \times n$ 的矩阵，V 是一个 $n \times n$ 的矩阵并且 U 和 V 都是正交矩阵，而 D 是一个对角矩阵。

对角矩阵 D 的对角线上的元素就是矩阵 A 的奇异值，U 和 V 分别被称为左右奇异向量，事实上这种分解方式可以理解为：D 就是 AA^T 特征值的平方根，而 U 和 V 分别是 AA^T 和 A^TA 的特征向

量，实际上就是我们将任意一个实矩阵通过一定的变换变成了一个方阵，然后对方阵进行奇异值分解反过来表示原本矩阵的一些特征。

因此我们可以适用奇异值分解的办法来提取一个单词的嵌入向量，但是我们首先需要将数据集中海量的单词汇总成一个矩阵 X 并对该矩阵进行奇异值分解，然后可以将分解得到的 U 作为所有单词的嵌入向量，而生成矩阵 X 的方法有多种选择。

2.2.2 单词文档矩阵 Word-Document Matrix

我们可以假设相关的单词总会在同一篇文档中出现，利用这一假设来建立一个单词文档矩阵，其中 $X_{ij} = 1$ 表示单词 w_i 出现在了文档 d_j 中，否则就是 0，这样一来矩阵的规模就是 $|V| \times M$，规模是非常庞大的。

2.2.3 基于窗口的共生矩阵 Co-occurrence Matrix

我们可以使用一个固定大小的滑动窗口来截取文档中的单词组合，然后对每一次采样得到的单词进行计数，如果单词 a 和单词 b 在一次采样结果中同时出现，就在矩阵 X 上给它们对应的位置的值 +1(相同的单词不考虑)，这样一来可以得到一个大小为 $|V| \times |V|$ 的矩阵并且 U 和 V 都是正交矩阵，而 D 是一个对角矩阵。

2.2.4 SVD 的使用

在获得的矩阵 X 上使用 SVD 的时候，最后得到的每个单词的嵌入向量是 $|V|$ 维的，我们可以进行截取，只保留 k 维：

$$\frac{\sum_{i=1}^{k} \sigma_i}{\sum_{i=1}^{|V|} \sigma_i} \tag{3}$$

然后用截取得到的 k 维向量作为每个单词的嵌入向量。

2.2.5 SVD 方法的缺陷与解决

SVD 方法还是存在一定的缺陷的，不管用什么方式生成矩阵 X，都会面临这样几个问题：

- 计算量较大，对于 $m \times n$ 的矩阵，SVD 的时间复杂度是 $O(mn^2)$，因此维数高了以后计算量会暴增
- 生成的矩阵要插入新单词或者新文档的内容时比较麻烦，矩阵的规模很容易变化
- 生成的矩阵往往比较稀疏，因为大部分单词和单词或者单词和文档之间可能没什么关系

面对这些问题，可以采取的解决措施有：

- 不考虑一些功能性的单词比如 a，an，the，it 等等
- 使用一个倾斜的窗口，也就是在生成共生矩阵的时候考虑单词相距的距离

2.3 Word2vec

2.3.1 基于迭代的模型

基于 SVD 的方法实际上考虑的都是单词的全局特征，因此计算代价和效果都不太好，因此我们可以尝试建立一个基于迭代的模型并根据单词在上下文中出现的概率来进行编码。

另一种思路是设计一个参数是词向量的模型，然后针对一个目标进行模型的训练，在每一次的迭代中计算错误率或者 loss 函数并据此来更新模型的参数，也就是词向量，最后就可以得到一系列结果作为单词的嵌入向量。常见的方法包括 bag-of-words (CBOW) 和 skip-gram，其中 CBOW 的目标是根据上下文来预测中心的单词，而 skip-gram 则是根据中心单词来预测上下文中的各种单词出现的概率。而这些模型常用的训练方法有：

- 负采样 negative sampling：判断两个单词是不是一对上下文与目标词，如果是一对，则是正样本，如果不是一对，则是负样本，而采样得到一个上下文词和一个目标词，生成一个正样本，用与正样本相同的上下文词，再在字典中随机选择一个单词，这就是负采样
- 层级化 (hierarchical) 的 softmax

2.3.2 语言模型 Language Model

语言模型可以给一系列单词序列指定一个概率来判断它是不是可以作为一个完整的句子输出，一般完整性强的，语义通顺的句子被赋予的概率会更大，而无法组成句子的一系列单词可能就会计算出非常小的概率，语言模型用公式来表示就是：

$$P\left(w_1, w_2, \cdots, w_n\right) \tag{4}$$

如果我们假设每个单词在每个位置出现的概率是独立的话，这个表达式就会变成：

$$P\left(w_1, w_2, \cdots, w_n\right) = \prod_{i=1}^{n} P(w_i) \tag{5}$$

但这显然是不可能的，因为单词的出现并不是不受约束的，而是和其周围的单词 (也就是上下文) 有一定的联系，这也是 NLP 中一个非常重要的观点，那就是**分布式语义** (Distributional semantics)：一

个词语的意思是由附近的单词决定的，一个单词的上下文就是"附近的单词"，可以通过一个定长的滑动窗口来捕捉每个单词对应的上下文和语义。

Bigram model 认为单词出现的概率取决于它的上一个单词，即：

$$P(w_1, w_2, \cdots, w_n) = \prod_{i=2}^{n} P(w_i|w_{i-1}) \tag{6}$$

但显然这也是一种非常 naive 的的模型，因此我们需要寻找更合适的语言模型。

2.3.3 连续词袋模型 (CBOW)

连续词袋模型 Continuous Bag of Words Model 是一种根据上下文单词来预测中心单词的模型，对于每一个单词我们需要训练出两个向量 u 和 v 分别代表该单词作为中心单词和作为上下文单词的时候的嵌入向量 (一种代表输出结果，一种代表输入)

因此可以定义两个矩阵 $\mathcal{V} \in \mathbb{R}^{n\times|V|}$ 和 $\mathcal{U} \in \mathbb{R}^{|V|\times n}$，其中 n 代表了嵌入空间的维度，矩阵 $\mathcal{V}$ 的每一列代表了词汇表中一个单词作为上下文单词时候的嵌入向量，用 v_i 表示，而矩阵 $\mathcal{U}$ 表示的是输出的结果，每一行代表了一个单词的嵌入向量，用 u_j 表示。

CBOW 模型的工作原理

1. 首先根据输入的句子生成一系列 one-hot 向量，假设要预测的位置是 c，上下文的宽度各为 m，则生成的一系列向量可以表示为：

$$\left(x^{(c-m)}, \ldots, x^{(c-1)}, x^{(c+1)}, \ldots, x^{(c+m)} \in \mathbb{R}^{|V|}\right) \tag{7}$$

2. 计算得到上下文单词的嵌入向量：

$$v_j = \mathcal{V}x^{(j)} \tag{8}$$

3. 求出这些向量的均值：

$$\hat{v} = \frac{v_{c-m} + v_{c-m+1} + \ldots + v_{c+m}}{2m} \in \mathbb{R}^n \tag{9}$$

4. 生成一个分数向量 $z = \mathcal{U}\hat{v}$，当相似向量的点积较大时，会将相似的词推到一起，以获得较高的分数

5. 用 softmax 函数将分数向量转化成概率分布：$\hat{y} = \text{softmax}(z) \in \mathbb{R}^{|V|}$

6. 根据概率分布 $\hat{y}$ 得到最终的结果 y(是一个 one-hot 向量的形式)，也就是最有可能出现在中心位置的单词

因此现在的问题就变成了，我们应该如何学习到两个最关键的矩阵 $\mathcal{V}$ 和 $\mathcal{U}$，这也是 CBOW 最关键的一个问题，这里可以采用信息论中的交叉熵来定义一个目标函数：

$$H(\hat{y}, y) = -\sum_{j=1}^{|V|} y_j \log\left(\hat{y}_j\right) = -y_i \log\left(\hat{y}_i\right) \tag{10}$$

因为概率的真实值 y 是一个 one-hot 向量，只有一个维度是 1 其他的都是 0，因此可以进行这样的化简。

损失函数与优化方式 我们可以定义如下形式的损失函数并进行优化：

$$\begin{aligned}
\text{minimize } J &= -\log P\left(w_c \mid w_{c-m}, \ldots, w_{c-1}, w_{c+1}, \ldots, w_{c+m}\right) \\
&= -\log P\left(u_c \mid \hat{v}\right) \\
&= -\log \frac{\exp\left(u_c^T \hat{v}\right)}{\sum_{j=1}^{|V|} \exp\left(u_j^T \hat{v}\right)} \\
&= -u_c^T \hat{v} + \log \sum_{j=1}^{|V|} \exp\left(u_j^T \hat{v}\right)
\end{aligned} \tag{11}$$

这里我们可以采用随机梯度下降 SGD 的方式来进行模型的求解和参数的更新。

2.3.4 Skip-gram 模型

Skip-gram 模型是一种给定中心单词来预测上下文中各个位置的不同单词出现概率的模型，模型的工作原理可以用如下几个步骤概括：

1. 生成中心单词的 one-hot 向量 $x \in \mathbb{R}^{|V|}$ 作为输入
2. 得到输入单词的嵌入向量 $v_c = \mathcal{V}x$ 并计算分数向量 $z = \mathcal{U}v_c$
3. 将分数向量用 softmax 转化成对应的概率分布 $\hat{y} = \text{softmax}(z) \in \mathbb{R}^{|V|}$，则

   $$\left(\hat{y}^{(c-m)}, \ldots, \hat{y}^{(c-1)}, \hat{y}^{(c+1)}, \ldots, \hat{y}^{(c+m)} \in \mathbb{R}^{|V|}\right) \tag{12}$$

   是预测出的对应位置中出现的单词的概率分布，每个概率向量的维度都和词汇表的大小一样
4. 根据概率分布生成对应的 one-hot 向量，作为最终的预测结果，这里就是将概率最大的那个维度对应的单词作为结果。

参数的求解和模型的优化 与CBOW类似，可以采用类似的方法来优化模型的参数，首先Skip-gram模型是通过中心单词c来预测上下文单词o的出现概率，这一过程可以表示为：

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)} \tag{13}$$

而其损失函数可以表示为：

$$J = -\sum_{w \in V} y_w \log(\hat{y}_w) = -\log P(O = o|C = c) = -u_o^T v_c + \log(\sum_{w \in V} \exp(u_w^T v_c)) \tag{14}$$

这样一来损失函数J对于词向量 u, v 的导数分别可以用如下方式来计算，首先来看中心词向量v的导数：

$$\begin{aligned}
\frac{\partial J}{\partial v_c} &= -\frac{\partial \left(u_o^T v_c\right)}{\partial v_c} + \frac{\partial \left(\log \left(\sum_w \exp \left(u_w^T v_c\right)\right)\right)}{\partial v_c} \\
&= -u_o + \frac{1}{\sum_w \exp \left(u_w^T v_c\right)} \frac{\partial \left(\sum_w \exp \left(u_w^T v_c\right)\right)}{\partial v_c} \\
&= -u_o + \sum_w \frac{\exp \left(u_w^T v_c\right) u_w}{\sum_w \exp \left(u_w^T v_c\right)} \\
&= -u_o + \sum_w P(O = w \mid C = c) u_w \\
&= -u_o + \sum_w \hat{y}_w u_w \\
&= -u_o + u_{\text{new}} \\
&= U^T(\hat{y} - y)
\end{aligned} \tag{15}$$

我们首先要搞清楚真实的结果y是一个one-hot向量，所以上述步骤是可以成立的，而对于向量 u_w，则要分情况讨论，当w=o的时候，有

$$\begin{aligned}
\frac{\partial J}{\partial u_w} &= -\frac{\partial \left(u_o^T v_c\right)}{\partial u_w} + \frac{\partial \left(\log \left(\exp \left(u_w^T v_c\right)\right)\right)}{\partial u_w} \\
&= -v_c + \frac{1}{\sum_w \exp \left(u_w^T v_c\right)} \frac{\partial \left(\exp \left(u_w^T v_c\right)\right)}{\partial u_w} \\
&= -v_c + \frac{\exp \left(u_w^T v_c\right) v_c}{\sum_w \exp \left(u_w^T v_c\right)} \\
&= -v_c + p(O = w \mid C = c) v_c \\
&= -v_c + \hat{y}_{w=o} v_c
\end{aligned} \tag{16}$$

而当w是其他值的时候，有：

$$\frac{\partial J}{\partial u_w} = 0 + \sum_{w \neq o} P(O = w|C = c) v_c = \sum_{w \neq o} \hat{y}_w v_c \tag{17}$$

考虑到真实结果 y 是一个 one-hot 向量，上面两个式子又可以合并为：

$$\frac{\partial J}{\partial U} = v_c^T(\hat{y} - y) \tag{18}$$

2.3.5 负采样 Negative Sampling

上面提到的算法中，时间复杂度基本都是 $O(|V|)$ 的 (比如 softmax 计算中，会消耗大量的算力)，也就是词汇表的大小，而这个规模往往是比较大的，因此我们可以想办法逼近这个值。在每一次的迭代中，不是遍历整个词汇表而是进行一些负采样操作，我们可以从一个噪声分布中“采样”，其概率与词汇表频率的顺序匹配。

负采样可以理解为单词和短语及其构成的分布式表示，比如在 skip-gram 模型中，假设一对单词和上下文 (w, c)，我们用 $P(D|w, c), D \in \{0, 1\}$ 来表示**一对单词和上下文是否属于训练集中出现过的内容**，首先可以用 sigmoid 函数来定义这个概率：

$$P(D = 1 \mid w, c, \theta) = \sigma\left(v_c^T v_w\right) = \frac{1}{1 + e^{(-v_c^T v_w)}} \tag{19}$$

这里的 v 就是上面提到的嵌入向量，然后我们可以构建一个新的目标函数来试图最大化一个单词和上下文对属于训练文本的概率，也就是采用极大似然法来估计参数，这里我们将需要求的参数 $\mathcal{V}, \mathcal{U}$ 记作 θ，则极大似然估计的目标可以表示为：

$$\begin{aligned}
\theta &= \underset{\theta}{\operatorname{argmax}} \prod_{(w,c)\in D} P(D = 1 \mid w, c, \theta) \prod_{(w,c)\in \tilde{D}} P(D = 0 \mid w, c, \theta) \\
&= \underset{\theta}{\operatorname{argmax}} \prod_{(w,c)\in D} P(D = 1 \mid w, c, \theta) \prod_{(w,c)\in \tilde{D}} (1 - P(D = 1 \mid w, c, \theta)) \\
&= \underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} \log P(D = 1 \mid w, c, \theta) + \sum_{(w,c)\in \tilde{D}} \log(1 - P(D = 1 \mid w, c, \theta)) \\
&= \underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} \log \frac{1}{1 + \exp\left(-u_w^T v_c\right)} + \sum_{(w,c)\in \tilde{D}} \log\left(1 - \frac{1}{1 + \exp\left(-u_w^T v_c\right)}\right) \\
&= \underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} \log \frac{1}{1 + \exp\left(-u_w^T v_c\right)} + \sum_{(w,c)\in \tilde{D}} \log\left(\frac{1}{1 + \exp\left(u_w^T v_c\right)}\right)
\end{aligned} \tag{20}$$

因此可以损失函数可以等价地采用如下形式，相对的，我们需要求的就是损失函数的最小值：

$$J = -\sum_{(w,c)\in D} \log \frac{1}{1 + \exp\left(-u_w^T v_c\right)} - \sum_{(w,c)\in \tilde{D}} \log\left(\frac{1}{1 + \exp\left(u_w^T v_c\right)}\right) \tag{21}$$

这里的集合 $\tilde{D}$ 表示负采样集合，也就是一系列不属于训练文本的中心词和上下文的对构成的一个集合，我们可以通过根据训练文本随机生成的方式来得到负采样集合。这样一来 CBOW 的损失函数就

变成了：

$$J = -\log\sigma\left(u_c^T \cdot \hat{v}\right) - \sum_{k=1}^{K} \log\sigma\left(-\tilde{u}_k^T \cdot \hat{v}\right) \tag{22}$$

而 Skip-gram 的新损失函数就变成了：

$$J = -\log\sigma\left(u_o^T \cdot v_c\right) - \sum_{k=1}^{K} \log\sigma\left(-\tilde{u}_k^T \cdot v_c\right) \tag{23}$$

这里的 $\tilde{u_k}$ 是随机采样产生的 K 个不是当前中心词的上下文的单词，在 skip-gram 模型中，梯度可以表示为：

$$\begin{aligned}\frac{\partial J_{\text{neg-sample}}}{\partial v_c} &= \left(\sigma\left(u_o^T v_c\right) - 1\right) u_o + \sum_{k=1}^{K}\left(1 - \sigma\left(-u_k^T v_c\right)\right) u_k \\ &= \left(\sigma\left(u_o^T v_c\right) - 1\right) u_o + \sum_{k=1}^{K}\sigma\left(u_k^T v_c\right) u_k\end{aligned} \tag{24}$$

而对于 u_o 和负采样向量 $\tilde{u_k}$，其梯度是：

$$\frac{\partial J_{\text{neg-sample}}}{\partial u_o} = \left(\sigma\left(u_o^T v_c\right) - 1\right) v_c \tag{25}$$

$$\frac{\partial J_{\text{neg-sample}}}{\partial u_k} = \sum_{k=1}^{K}\left(1 - \sigma\left(-u_k^T v_c\right)\right) v_c = \sum_{k=1}^{K}\left(\sigma\left(u_k^T v_c\right)\right) v_c \tag{26}$$

2.4 单词表示的全局向量 GloVe

2.4.1 已有的单词表示方法

到现在为止已经介绍过的单词表示方法主要分为两种，一种是基于次数统计和矩阵分解的传统机器学习方法，这类方法可以获取全局的统计信息，并很好的捕捉单词之间的相似性，但是在单词类比之类的任务重表现较差，比较经典的有潜在语义分析 (LSA) 等等，另一类是上面提到的基于″浅窗口″(shallow window based) 的方法，比如 CBOW 和 Skip-gram，通过局部的上下文来进行上下文或者中心单词的预测，这样的方法可以捕捉到复杂的语言模式，但是对全局的统计信息知之甚少，缺少″大局观″，这也是这些模型的缺点。

而 GloVe 则使用一些全局的统计信息，比如共生矩阵，并使用最小二乘损失函数来预测一个单词出现在一段上下文中的概率，并且在单词类比的任务上取得了 SOTA 的效果。

2.4.2 GloVe 算法

首先用 X 来表示训练集的共生矩阵，而 X_{ij} 表示单词 i 的上下文中出现单词 j 的次数，用 X_i 来表示矩阵中第 i 行的和则这样一来：

$$P_{ij} = P(w_j|w_i) = \frac{X_{ij}}{X_i} \tag{27}$$

可以表示单词 j 在单词 i 的上下文中出现的概率。

我们在 Skip-gram 中，用 Softmax 来计算单词 j 出现在 i 的上下文中的概率：

$$Q_{ij} = \frac{\exp\left(\vec{u}_j^T \vec{v}_i\right)}{\sum_{w=1}^{W} \exp\left(\vec{u}_w^T \vec{v}_i\right)} \tag{28}$$

而训练的过程中用到的损失函数如下所示，又因为上下文的关系在样本中可能会多次出现，因此按照如下方式进行变形：

$$J = -\sum_{i\in \text{ corpus }(i)} \sum_{j\in \text{ context(i)}} \log Q_{ij} = \sum_{i=1}^{W} \sum_{j=1}^{W} X_{ij} \log Q_{ij} \tag{29}$$

交叉熵损失的一个显著缺点是它要求分布 Q 被适当地标准化，因此可以换一种方式，也就是使用基于最小二乘的目标函数来进行优化求解，就可以不用进行标准化了：

$$\hat{J} = \sum_{i=1}^{W} \sum_{j=1}^{W} X_i \left(\hat{P}_{ij} - \hat{Q}_{ij}\right)^2 = \sum_{i=1}^{W} \sum_{j=1}^{W} X_i \left(X_{ij} - \exp(\vec{u}_j^T)\vec{v}_i\right)^2 \tag{30}$$

但是这样子的目标函数又带来了一个新的问题，那就是 X_{ij} 往往是一个比较大的数字，这会对计算量造成很大的影响，一个有效的方法是取对数：

$$\begin{aligned} \hat{J} &= \sum_{i=1}^{W} \sum_{j=1}^{W} X_i \left(\log(\hat{P})_{ij} - \log\left(\hat{Q}_{ij}\right)\right)^2 \\ &= \sum_{i=1}^{W} \sum_{i=1}^{W} X_i \left(\vec{u}_j^T \vec{v}_i - \log X_{ij}\right)^2 \end{aligned} \tag{31}$$

2.4.3 结论

GloVe 模型通过只训练共生矩阵中的非零元素充分利用了训练样本的全局信息，并且提供了一个拥有比较有意义的子结构的向量空间，在单词类比的任务中表现优于传统的 Word2Vec

2.5 词向量的评估

目前已经学习了一系列将单词用词向量来表示的方法，接下来的这一部分主要探讨如何评估生成的词向量的质量的好坏。

2.5.1 内部评估

内部评价 (Intrinsic Evaluation) 是对生成的一系列词嵌入向量，用一些子任务来验证嵌入向量的性能优劣，并且一般这样的子任务是比较容易完成的，这样可以快速反馈词向量训练的结果。

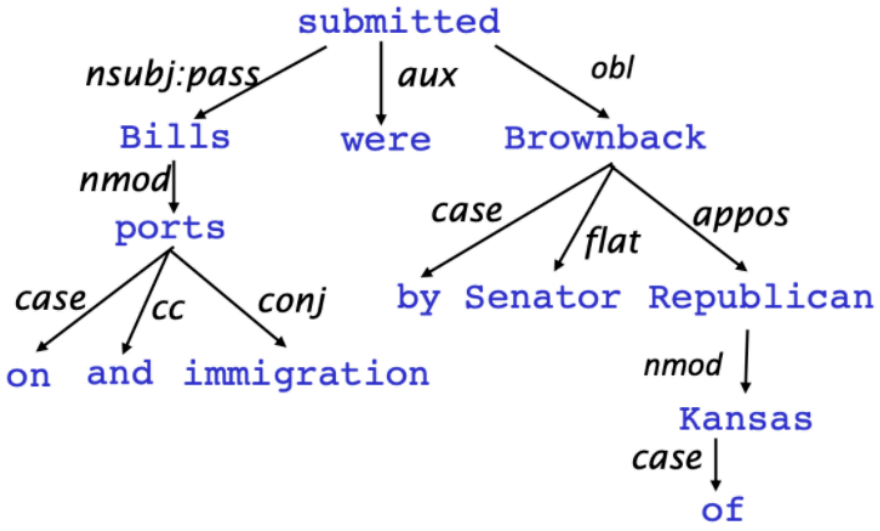
2.5.2 外部评估

外部评估是用一些实际中的任务来评估当前训练获得的词向量的性能，但是在优化一个表现不佳的外部评估系统的时候我们无法确定是哪个特定的子系统产生的错误，这就激发了对内在评估的需求。

3 依赖性解析 Dependency Parsing

3.1 依赖性关系

语言结构有两种视角来看待，第一种是短语结构，是结构化的语法 (上下文无关语法 Context-free Grammars)，短语结构将单词组成一系列嵌套的成分，从单个单词组成短语，再组成完整的句子。第二种是依存关系结构，这种结构表明一个单词依赖于哪些单词。自然语言处理需要研究句子的结构才能分析出句子的真正含义。依赖关系是一种二元的非对称关系，这种关系也可以用树形的结构来表示：



依赖性分析最终输出的就是类似于上面的依赖语法树的结构，通过对输入的句子 $s = w_0 w_1 w_2 \dots w_n$ 进行分析来得到依赖语法树 G（也叫做依赖图），而依赖性分析中还有两个子任务，分别是：

- 学习：通过给定的数据集 D 和给定的依赖图，生成一个依赖性的解析模型
- 分类：对于输入的测试句子，生成合适的依赖图

3.2 基于转换的依赖性解析

这种依赖性分析方式依靠一个定义了可能转移过程的有限状态机来建立输入句子到依赖树的映射，这种策略下的学习过程就是训练一个可以通过状态转变的历史来预测下一个状态转变的有限状态机，解析的过程就是构建出最优的依赖树。常见的基于贪心的确定性转换的依赖性解析，这种解析方式的核心依然是一个有限状态机。

下面有段看不懂的，感觉是很玄学的计算理论，就是有限状态机那一套东西，没学过不太懂。

3.3 基于神经网络的依赖性解析

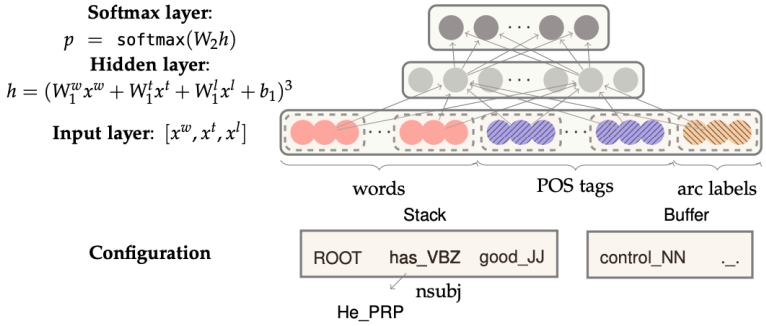
基于神经网络的依赖性解析相比于传统的模型，主要的区别在于依赖稠密而不是稀疏的特征表示。最终，该模型的目标是预测一个从初始配置 c 到终端配置的转换序列，其中对依赖关系解析树进行了编码，每次尝试基于当前的配置 $c = (\sigma, \beta, A)$ 中提取的特征预测一个转变过程 T，其中三个参数分别表示栈，缓冲区和一系列的依赖关系构成的集合。

- 这里的栈应该是经过语法分析后确定的单词解析顺序构成的栈，而缓冲区里存放了这些单词可能依赖的词

对于一个给定的句子 S，其特征包括以下内容的子集：

- word：句子 S 中的一些位于栈顶和 buffer 中的词向量
- tag：词性的标记
- label：弧线标记 S 中的一些单词，弧线标记集合包含了一些小规模的关系样本来描述特定的关系

3.4 一个简单的神经网络的例子



这里使用了 $f(x) = x^3$ 作为非线性的激活函数，并将 word，tag 和 label 三种特征作为输入，在隐层进行矩阵计算而在输出层进行 softmax

4 语言模型，RNN 和 LSTM

4.1 语言模型

语言模型是一种计算在一个特定序列出现的概率的模型，对于 m 个单词 w_i，它们同时出现并且连成一句话的概率是 $P(w_1, \ldots, w_n)$，很明显单词之间出现的概率不是完全独立的，也就是说一些单词的出现会和句子中的其他单词有关，因此这个概率是比较难计算的，根据条件概率公式可以表示为：

$$P(x^{(1)}, \ldots, x^{(t)}) = P(x^{(1)}) \times P(x^{(2)}|x^{(1)}) \times \cdots \times P(x^{(t)}|x^{(1)}, \ldots, x^{(t-1)}) = \prod_{i=1}^{t} P(x^{(i)}|x^{(1)}, \ldots, x^{(i-1)}) \tag{32}$$

我们可以做出这样一个简化问题假设 (也叫做马尔可夫假设)：一个单词出现的概率依赖于出现在它前面的 n 个单词，这样一来这个概率可以表示为：

$$P\left(w_1, \ldots, w_m\right) = \prod_{i=1}^{i=m} P\left(w_i \mid w_1, \ldots, w_{i-1}\right) \approx \prod_{i=1}^{i=m} P\left(w_i \mid w_{i-n}, \ldots, w_{i-1}\right) \tag{33}$$

这个表达式在机器翻译中有比较大的作用，翻译系统在完成句子的翻译之后实际上会生成多种不同的结果，这个时候就可以使用语言模型来对每一种结果进行评估，一般来说语义不通顺的句子计算出的概率会比较低，而语义通顺的句子计算出的概率会比较高，这样一来就可以选择概率最高的一个作为翻译的结果。

4.1.1 n-gram 语言模型

因此要获得一个语言模型，我们需要对每一组连续 n 个单词进行出现频率的计算，这就是 n-gram 语言模型，比如说对于一个 2-gram 的语言模型，我们可以这样计算：

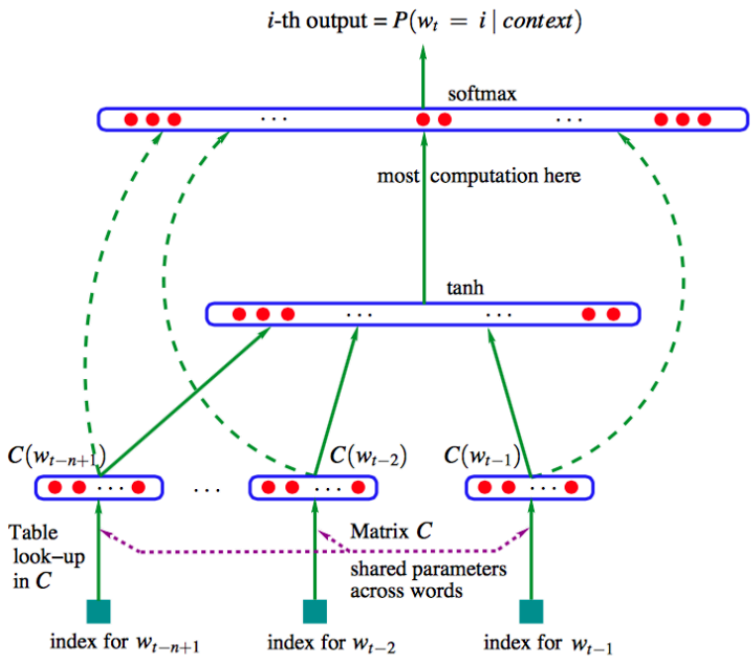
$$p(w_2|w_1) = \frac{\text{count}(w_1, w_2)}{\text{count}(w_1)} \tag{34}$$

通过大量文本获得了统计信息之后，n-gram 就可以根据前 n 个单词来预测出当前位置单词的出现概率分布，并选择出现概率最大的作为预测结果，这样一来就导致了语言模型存在的两个问题：

- n-gram 语言模型可能会非常稀疏，因为一些单词组合可能永远也不会在训练样本中出现，我们可以用 smoothing 的方法来解决这一问题，也就是说不管有没有出现过我们都算这个组合至少出现了 δ 次
- 随着 n 的增长，n-gram 语言模型需要的存储空间也会越来越大，并且这一复杂度是指数规模的

4.1.2 基于窗口的神经语言模型

基于窗口的神经网络语言模型可以学习出单词的分布式表示和计算序列出现概率的函数，它的基本结构如下所示：



这个模型可以用下面这个表达式来表示：

$$\hat{y} = \text{softmax}\left(W^{(2)} \tanh\left(W^{(1)}x + b^{(1)}\right) + W^{(3)}x + b^{(3)}\right) \tag{35}$$

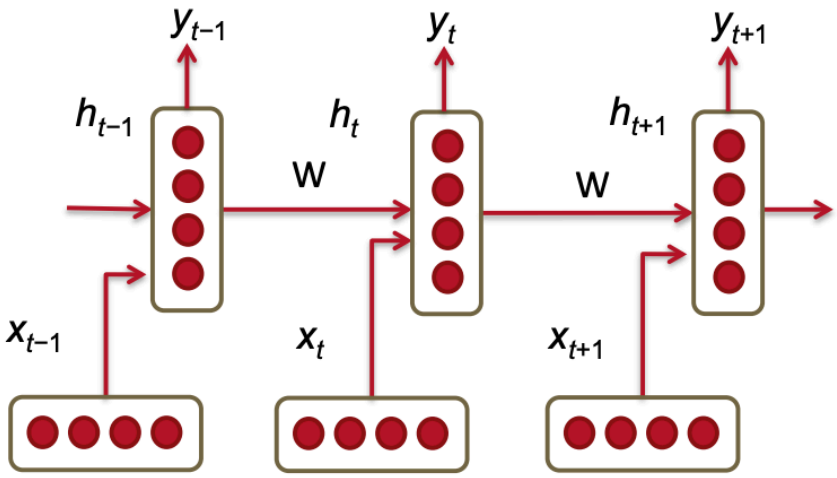
其中第一个矩阵作用在词向量上 (图中的绿色实线)，第二个矩阵作用在隐层，第三个矩阵作用在词向量上 (图中的绿色虚线)，这个模型虽然简单但是也能够起到比较好的 performance，上面提到的这些语言模型实际上只是**基于有限长度的滑动窗口的卷积模型**，但事实上，窗口比较小的时候模型预测的准确度也是无法保证的，而窗口大了又会导致参数的规模急剧增长，因此我们需要一个可以处理任意长度的输入，并且参数规模维持在一定范围内的新模型。

4.2 循环神经网络 RNN

4.2.1 RNN 的架构

和只能给予一个有限长度的滑动窗口不同，卷积翻译模型不同，循环神经网络 (Recurrent Neural Networks) 是可以根据语料库中所有之前出现过的额单词来调整模型，RNN 的基本组成单元如下图

所示：



每个神经网络单元 (也称为 timestep) 会对输入的词嵌入向量 x_t 进行一个矩阵运算，并且对上一个位置的单元传递下来的 h_{t-1} 进行运算以后相加得到 h_t 传递给下一个单元，然后对 h_t 使用非线性函数激活和 sotfmax 函数之后输出一个概率分布 $\hat{y}_t$ 作为当前单元的输出：

$$\begin{aligned} h_t &= \sigma\left(W^{(hh)}h_{t-1} + W^{(hx)}x_t\right) \\ \hat{y}_t &= \text{softmax}\left(W^{(S)}h_t\right) \end{aligned} \tag{36}$$

并且权重矩阵 $W^{(hh)}, W^{(hx)}$ 在不同的 timestep 之间时共享的，这样一来**一个 RNN 模型需要学习的参数数量是比较少的，并且和输入语料库的长度是没有无关的**，不仅避免了参数的维度爆炸，而且可以处理任意长度的输入结果。

4.2.2 RNN 的损失函数

RNN 的损失函数通常采用交叉熵来计算，对于第 t 个 timestep，其交叉熵可以表示为：

$$J^{(t)}(\theta) = -\sum_{j=1}^{|V|} y_{t,j} \times \log\left(\hat{y}_{t,j}\right) \tag{37}$$

这样一来总的交叉熵就可以表示为：

$$J(\theta) = \frac{1}{T}\sum_{i=1}^{T} J^{(t)}(\theta) = -\frac{1}{T}\sum_{i=1}^{T}\sum_{j=1}^{|V|} y_{t,j} \times \log\left(\hat{y}_{t,j}\right) \tag{38}$$

根据损失函数就可以对 RNN 中的权重矩阵进行优化，这里采用的仍然是梯度下降和反向传播的方法，但是要注意的是，RNN 的反向传播实际上是根据时间的反向传播，也就是按照 timestep 的顺序从后往前传播，具体的公式后面有空进一步推导。

4.2.3 RNN 的反向传播

我们可以对上面的 RNN 表达式进行一定的修改，并假设激活函数 $\sigma(x) = \tanh(x)$，这样一来 RNN 中的一系列计算可以分解成如下几个步骤：

$$\begin{aligned} s_t &= Uh_{t-1} + Wx_t \\ h_t &= \tanh(s_t) \\ z_t &= Vh_t \\ \hat{y}_t &= \text{softmax}(z_t) \end{aligned} \tag{39}$$

这里 x 是 d 维的词嵌入向量，而 h 是维度为 D_h 的隐藏状态，最终输出的结果 z 和 y 是维度为 $|V|$ 的列向量，损失函数采用的就是上面提出的 J，这样一来我们可以根据反向传播算法，通过 J 来更新 U, V, W, h_t 等等，推导的具体过程如下：

首先我们需要注意到，softmax 函数可以写成：

$$f = [f_1, f_2, \ldots, f_{|V|}], \quad f_i = \frac{\exp^{x_i}}{\sum_{j=1}^{|V|} \exp^{x_j}} \tag{40}$$

而 softmax 函数关于 x_j 的导数可以写成：

$$\begin{aligned} \frac{\partial f_i}{\partial x_j} &= -f_i f_j (j \neq i) \\ \frac{\partial f_i}{\partial x_j} &= f_i(1 - f_i)(j = i) \end{aligned} \tag{41}$$

这样一来，可以先求出损失函数对于 z_t 的梯度：

$$\begin{aligned} \frac{\partial J(\theta)}{\partial z_i} &= \sum_{j=1}^{|V|} \frac{\partial J(\theta)}{\partial \hat{y}_{t,j}} \frac{\partial \hat{y}_{t,j}}{\partial z_i} \\ &= -\sum_{j=1}^{|V|} \frac{y_{t,j}}{\hat{y}_{t,j}} \frac{\partial \hat{y}_{t,j}}{\partial z_i} \\ &= -\sum_{j=1}^{|V|} \frac{y_{t,j}}{\hat{y}_{t,j}} (\hat{y}_{t,j}\hat{y}_{t,i}) + \frac{y_{t,i}}{\hat{y}_{t,i}} (\hat{y}_{t,j}\hat{y}_{t,i} + (1 - \hat{y}_{t,i})\hat{y}_{t,i}) \\ &= \hat{y}_{t,i} \sum_{j=1}^{|V|} y_{t,j} - y_{t,j} \\ &= \hat{y}_{t,i} - y_{t,i} \end{aligned} \tag{42}$$

这样一来 E 对于 z 的梯度就是一个 $|V|$ 维的向量并且每一项都是上面这样的形式 (随着下标的变化而变化)，这里 $\hat{y}_{t,i}$ 表示预测结果，是一个概率分布，而 $y_{t,i}$ 则是正确结果的 one-hot 向量，因此上述步

骤可以成立，这样一来，V 的梯度更新可以表示为：

$$\frac{\partial J(\theta)}{\partial V} = (\hat{y}_{t,i} - y_{t,i}) \otimes h_t \tag{43}$$

同样类似地可以得到：

$$\frac{\partial J(\theta)}{U} = \sum_{i=1}^{t} \delta_i \otimes h_{i-1} \tag{44}$$

$$\frac{\partial J(\theta)}{W} = \sum_{i=1}^{t} \delta_i \otimes x_i \tag{45}$$

其中 $\delta_k = (U^T(\hat{y}_t - y_t)) \times (1 - h_t^2)(k \leq t)$ 并且 $\delta_t = (V^T(\hat{y}_t - y_t)) \times (1 - h_t^2)(k \leq t)$

4.2.4 RNN 的优缺点

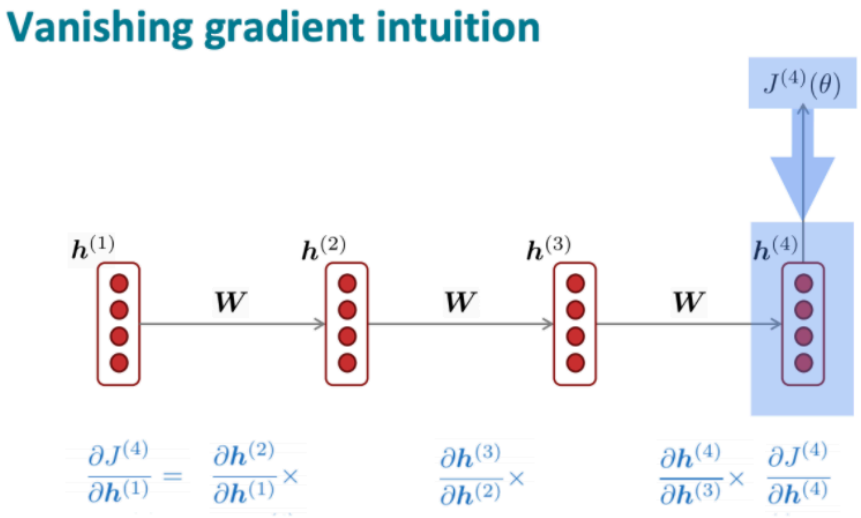
RNN 的优点主要有如下几个：

- RNN 可以处理任意长度的输入
- RNN 模型的 size 不会随着输入的语料库长度变化而变化
- 某个 timestep 计算得到的结果理论上可以在后面的很多 timestep 中被用到
- 因为每个 timestep 的权重是相同的，因此 RNN 模型具有一定的对称性

但是同时，RNN 的缺点也是很明显的：

- RNN 是一个序列化的模型，因此 RNN **不能进行并行的训练**
- 事实上随着 timestep 的推移，前面的信息越来越难以被保留下来，可能存在梯度消失和梯度爆炸的问题

4.2.5 梯度消失 Vanishing Gradient



在 RNN 的反向传播过程中，参数需要根据其梯度来更新，而梯度需要用链式求导法则来进行计算，如果其中的一部分梯度非常小，可能会导致最终要求的梯度累积越来越小，造成梯度的消失。

$$
\begin{aligned}
h^{(t)} &= \sigma(W_h h^{(t-1)} + W_e e^{(t)} + b_1) \\
\frac{\partial h^{(t)}}{\partial h^{(t-1)}} &= \mathrm{diag}(\sigma'(\mathrm{W_h h^{(t-1)}} + \mathrm{W_e e^{(t)}} + \mathrm{b_1}))W_h \\
&= IW_h = W_h \\
\frac{\partial J^{(i)}(\theta)}{\partial h^{(j)}} &= \frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} \prod_{j<t\leq i} \frac{\partial h^{(t)}}{\partial h^{(t-1)}} \\
&= \frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} \prod_{j<t\leq i} W_h = \frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} W_h^{i-j}
\end{aligned}
\tag{46}
$$

而我们发现根据上面的推导，如果矩阵 W_h 的所有特征值都比较小的话可能会导致最终计算的结果非常小，而这就导致了梯度消失。梯度消失会导致反向传播算法失效，参数无法再更新下去。对于 RNN 来说通过多个 timestep 以后学习到持续的信息是比较困难的，因此一种新的模型 LSTM 被提出，这种模型使得神经网络拥有短暂的记忆能力。

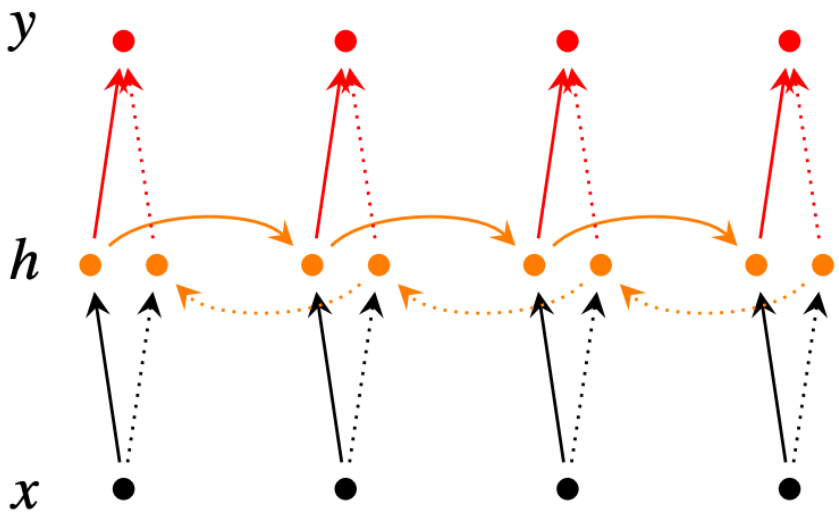
4.2.6 梯度爆炸 Exploding Gradient

同样的如果梯度过大也会使得参数的更新出现问题，因为梯度值过大导致反向传播更新的时候容易出现参数大幅度的更新，引起波动，这称之为 bad update，可能会导致最终的计算结果是 INF

或者 NAN，解决办法：Gradient Clipping，当梯度 g 的大小超过一个固定的阈值的时候，对 g 进行一定的压缩，防止梯度爆炸。因此 LSTM 被踢出用来解决梯度消失和梯度爆炸的问题。

4.2.7 双向 RNN

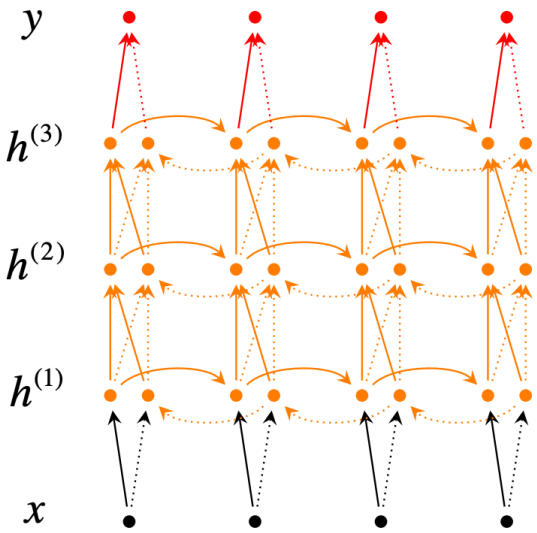
前面介绍的 RNN 模型实际上只能把前面的单词信息传递给后面，信息的传递是单向的，而双向 RNN 则可以进行双向的信息传递，实现的方式也很简单，就是通过在隐层增加一系列神经元来进行反向的信息传递，如下图所示：



而这一架构用公式可以表示为：

$$\begin{aligned} \vec{h}_t &= f\left(\vec{W}x_t + \vec{V}\vec{h}_{t-1} + \vec{b}\right) \\ \overleftarrow{h}_t &= f\left(\overleftarrow{W}x_t + \overleftarrow{V}\overleftarrow{h}_{t+1} + \overleftarrow{b}\right) \\ \hat{y}_t &= g\left(Uh_t + c\right) = g\left(U\left[\vec{h}_t; \overleftarrow{h}_t\right] + c\right) \end{aligned} \tag{47}$$

同时 RNN 也可以有多层架构，这样的架构下，RNN 的信息交互会更多，预测结果会更加准确。



4.3 长短期记忆模型 LSTM

4.3.1 门控循环单元

虽然理论上来说，RNN 可以提取长期的，距离较远的单词依赖关系，但是实际训练过程中其实是难以实现的，因为计算机数值计算的有限精度导致了 RNN 存在梯度消失和梯度爆炸等问题，而门控循环单元 (Gated recurrent units，GRU) 的目的就是实现更加长持久性的对长依赖关系的记忆，在这种门控循环单元的作用下，使用 $h_t = f(h_{t-1}, x_t)$ 的这一个过程可以表示为：

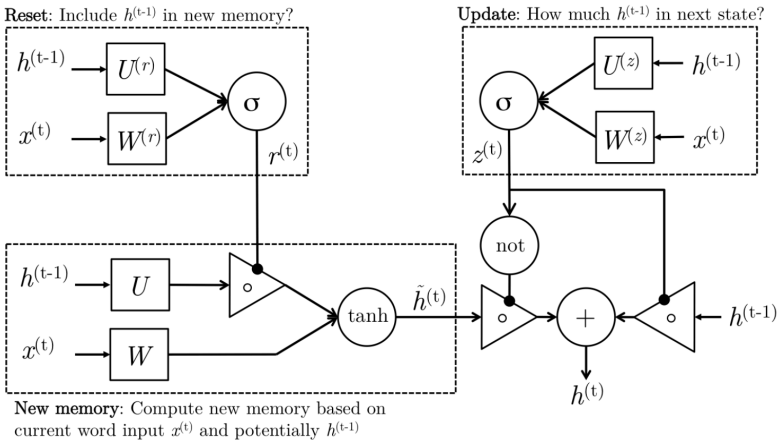
$$\begin{aligned} z_t &= \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right) && \text{(Update gate)} \\ r_t &= \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right) && \text{(Reset gate)} \\ \tilde{h}_t &= \tanh\left(r_t \circ Uh_{t-1} + Wx_t\right) && \text{(New memory)} \\ h_t &= (1 - z_t) \circ \tilde{h}_t + z_t \circ h_{t-1} && \text{(Hidden state)} \end{aligned} \tag{48}$$

以上四个公式就是 GRU 的四个基本步骤，它们有着比较直观的解释性并且可以让模型对持久性信息和长依赖关系的解析更加到位，这四个步骤可以分别理解为：

- 重置门：重置信号 r_t 可以决定 h_{t-1} 对于 $\tilde{h}_t$ 的重要性，如果发现没有啥关系，重置门有权利彻底消除过去的隐藏状态
- 更新门：更新信号 z_t 可以决定 h_{t-1} 对当前层究竟有多高的权重，以此来决定隐藏状态的最后结果相对于上一层的信息和这一层新的 meemory 的权重

- 新记忆生成：根据当前单元输入的 x_t, h_{t-1} 生成新的 $\tilde{h}_t$
- 隐藏状态：最终按一定权重将新的记忆和上个单元传递下来的 h_{t-1} 组合

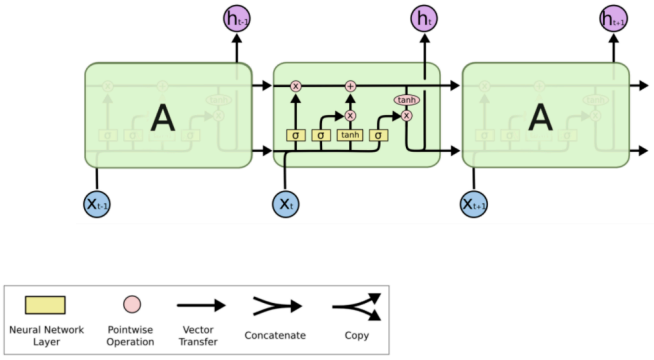
上面的这些过程可以用下面的这幅图来表示：

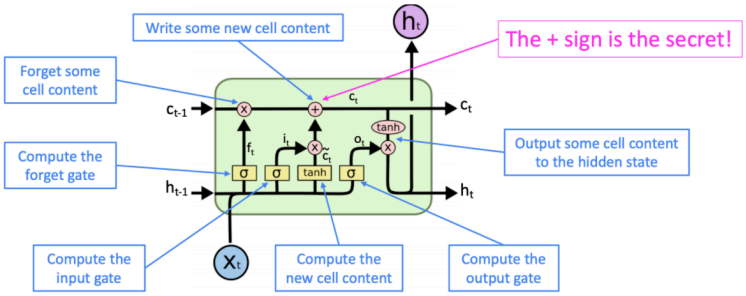


不得不说 CS224N 的插图对于 GRU 的讲解还是非常清楚的，而 GRU 中我们主要需要学习的参数有 6 个矩阵，同样的也是用反向传播的算法来进行训练。

4.3.2 长短期记忆 LSTM

长短期记忆模型 (Long-Short-Term-Memories,LSTM) 和门控循环单元略有区别，但也是用于保存长期记忆的一种复杂激活单元架构，如下图所示：





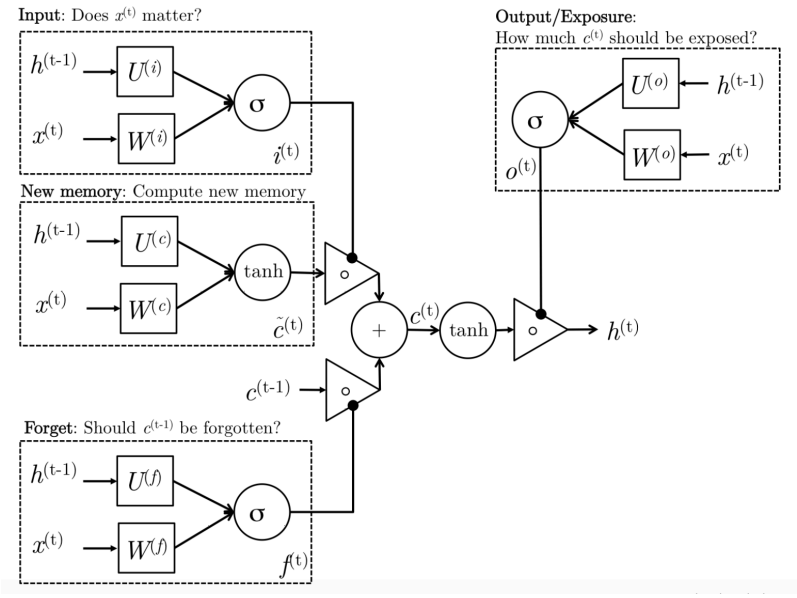
这种架构下，隐藏层中主要进行的数学运算有：

$$
\begin{aligned}
i_t &= \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right) \quad \text{(Input gate)} \\
f_t &= \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1}\right) \quad \text{(Forget gate)} \\
o_t &= \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1}\right) \quad \text{(Output/Exposure gate)} \\
\tilde{c}_t &= \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1}\right) \quad \text{(New memory cell)} \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \quad \text{(Final memory cell)} \\
h_t &= o_t \circ \tanh\left(c_t\right)
\end{aligned}
\tag{49}
$$

LSTM 中有若干个门，分别叫做输入门，输出门和遗忘门，起到一定的判断作用，用来判断 LSTM 中的一些信息是否值得保留，LSTM 的计算过程可以描述为：

- 生成新记忆：根据上一个单元传入的 h_{t-1} 和当前位置的词向量 x_t 计算出一个结果 $\tilde{c}_t$
- 输入门：根据上一个单元传入的 h_{t-1} 和当前位置的词向量 x_t 计算出一个结果 i_t，并评估当前位置的单词 x_t 是否有用，并有必要在之后的计算中使用
- 遗忘门：根据上一个单元传入的 h_{t-1} 和当前位置的词向量 x_t 计算出一个结果 f_t，并且评估上一个隐藏状态 h_{t-1} 是否有价值用在当前层的计算中
- 生成最终记忆：根据输入门和遗忘门的判断，并结合新的记忆单元和上一个传递过来的记忆单元的信息生成最终记忆
- 输出门：这是一个 GRU 中不存在的门，其目的是将最终的记忆单元和隐藏状态分开，因为最终生成的记忆单元有很多前文的信息是没有必要保存在当前隐藏状态中的，因此这一个单元决定了最终记忆 c_t 有多少内容需要暴露给 h_t，根据这个权重来产生最终的隐藏状态

事实上 LSTM 相比于前面提到的 GRU 主要多出了一个输出门用来控制信息暴露给当前隐藏状态的程度，可以用下面一张图来表示 LSTM 的各个门之间的关系：



4.3.3 LSTM 如何解决梯度问题

LSTM 的架构使得神经网络可以在许多 timestep 之后依然保存一些持久化的信息，可以通过改变门的开关达到保存信息的目的，但事实上，并没有彻底解决梯度消失和梯度爆炸的问题，但是让模型更容易学习到长距离的依赖关系，事实上梯度的消失和爆炸是所有神经网络架构都存在的问题，不仅仅是 RNN

5 机器翻译与 Seq2Seq 模型

词向量和 RNN 解决的主要是单词预测和命名实体识别 (Named Entity Recognization) 的问题，而这一节内容主要研究翻译、回答和概括等 NLP 子问题，并且重点介绍 Seq2Seq 模型。

5.1 传统翻译模型

机器翻译任务是将句子从一种语言翻译成另一种语言，1990 年代到二十一世纪初的机器翻译主要运用的还是基于统计学习的机器翻译方法 (SMT，Statical Machine Translation)，使用贝叶斯方法来构建模型，需要用大量的数据分别训练翻译模型和语言模型，为了将大量数据用于模型的并行训练，需要引入一个对齐变量，使得原文和译文要是对应的，否则可能导致翻译结果的不准确。alignment 是一种隐变量，没有显式地出现在数据中，因此**可以使用 EM 算法来进行优化**。

早起的翻译模型主要依赖于统计和概率的传统方法，这种模型分成两个部分，分别是一个翻译模型和一个语言模型，翻译模型用来预测文本中的一个句子最有可能翻译成什么，而语言模型来判断当前的句子是否对于全文而言是最合适的。

这种系统的构建主要基于单词或者短语，但是基于单词的翻译系统难以捕捉到语句中单词位置导致的差异，比如否定词的位置，句子中主语和动词的关系，而基于短语的翻译系统可以看成是 Seq2Seq 的前身，这种模型将输入和输出都看成是一系列短语并且可以构建更复杂的语法，但是这种模型下，长依赖关系依然难以提取，

5.2 Seq2Seq 模型

5.2.1 简介

Sequence-to-sequence(简称为 Seq2Seq，中文翻译是序列到序列) 是一种相对比较新的**端到端的翻译模型**，Seq2Seq 使用两个 RNN 模型构成的，分别是：

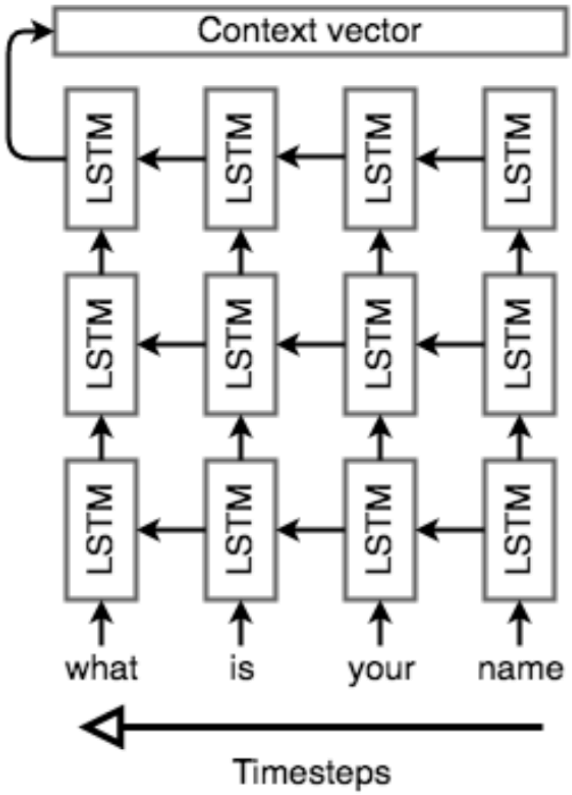
- 编码器：将输入序列编码成一个上下文向量 (context vector)
- 解码器：根据编码器生成的上下文向量生成对应的输出序列

因此 Seq2Seq 也被称为“编码器-解码器”模型。无论是编码器还是解码器，都需要先用一个嵌入层将输入的单词序列转化成词向量作为编码器和解码器的输入。

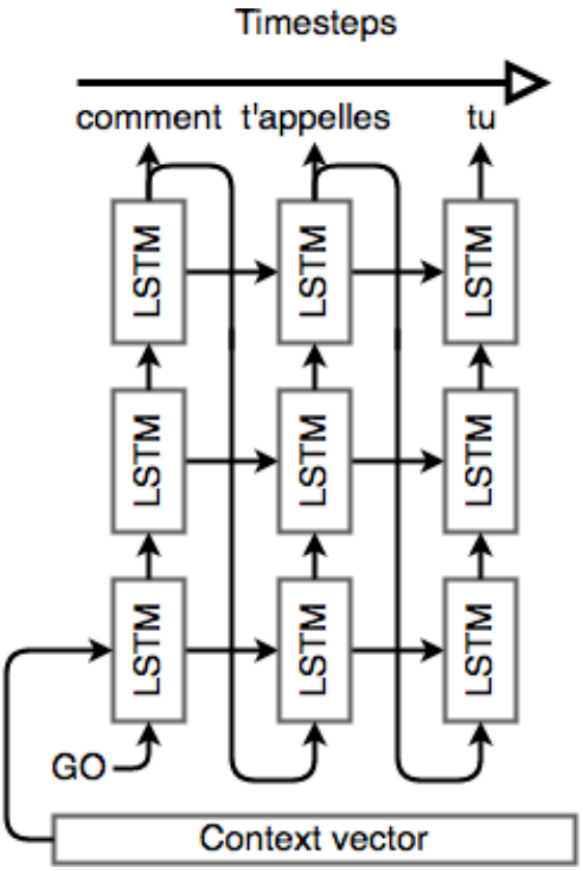
5.2.2 编码器

Seq2Seq 中的编码器用于将输入的单词序列编码成一个定长的上下文向量，因此编码器的组成单元通常是用 LSTM 的 RNN，这样的解码器在提取长依赖关系的时候效果更好，最终的隐藏状态就是 C，但将任意长度的输入压缩到一个固定长度的向量中是比较困难的，并且解码器往往有若干个 LSTM 层，称为 LSTM 栈 (stacked LSTM)，是一层接一层的 LSTM, 也可以叫做 LSTM 网络，每一层的输入是上一层的输出，最上层是隐藏状态层，**输出的结果就是定长的向量上下文向量 C**

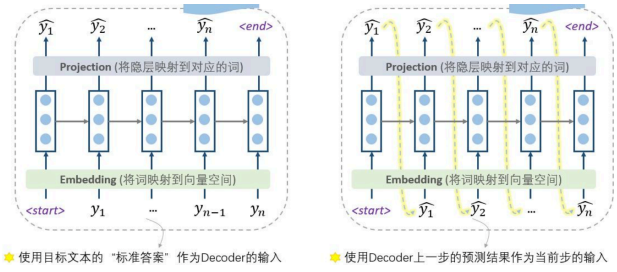
Seq2Seq 会对输入的训练进行反向的处理，也就是从序列的末尾从后向前处理，这样一来编码器读取到的最后一个单词将会对应输出结果中的第一个单词，这样可以方便解码器更快生成准确度更高的前面一部分译文，而译文最前面的一小部分的正确性对于译文整体的正确性的影响是非常大的，下面的图可以用来表示解码器的工作过程：



5.2.3 解码器



Seq2Seq 的解码器也是一个 LSTM 网络，当时相比于编码器的网络架构更复杂，首先要明确，**解码器的输入不是编码器中生成的上下文向量** C，解码器的输入依然是一系列由序列转化过来的单词向量，并且在训练和测试的时候的内容是不同的，而上下文向量 C 是一个用于预测结果的参数，而不是解码器的输入。



- 在训练阶段，解码器中输入的是**真实的目标文本**，并且第一步使用一个特殊的 <start> 标记表明这是句子的开头，每一步根据当前正确的输出词，上一步的隐状态来预测下一步的输出

- 在训练的时候，因为没有“参考答案”了，所以只能把上一步的输出作为下一步的输入，一步步慢慢预测出最终结果

这样两种训练方式，分别称为 teacher forcing 和 free running，事实上训练阶段也可以 free running，但是这样训练出来的模型的 performance 非常糟糕，容易出现误差爆炸的问题，使用“参考答案”(实际上也可以看成是当前输入序列的一个标签) 可以提高准确度。同时训练的时候可以采用更好的办法，也就是计划采样 (Scheduled Sampling)，

当我们获得了一个输出序列的时候，我们可以使用同样的学习策略，定义损失函数 (比如使用交叉熵)，然后用梯度下降的算法来反向传播优化参数，编码器和解码器是同时训练的，因此它们会学习到同样的上下文向量表示。

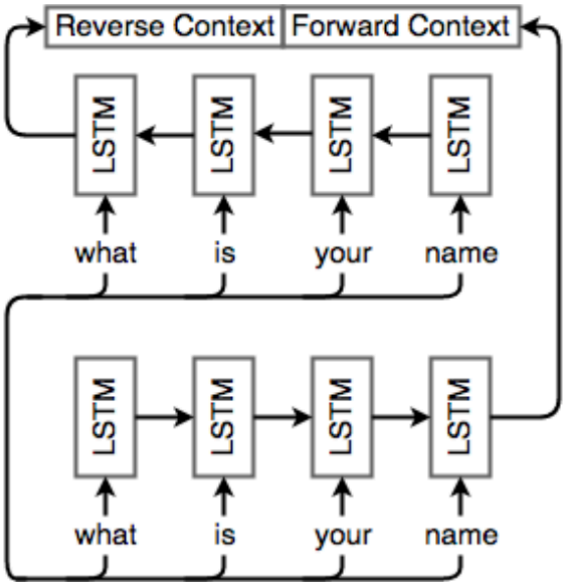
5.2.4 Seq2Seq 的损失函数

如果说词汇表 V 的容量是 $|V|$，那么显而易见，Seq2Seq 模型输出的结果就是一个 V 维度的向量代表单词作为可能结果的一个概率分布，Seq2Seq 最后的映射过程就是找出出现概率最高的单词作为最后的输出，我们在计算损失的时候可以使用交叉熵作为损失函数，假设解码器运行了 T 个步骤，那么最后的损失函数可以表示为：

$$J = -\frac{1}{T}\sum_{i=1}^{T}\log(P(\hat{y}_i)) \tag{50}$$

5.2.5 双向 RNN

我们知道单词之间的依赖关系可能是双向的，目前所提到的 Seq2Seq 模型只能提取单个方向的依赖关系，而双向 RNN 的提出解决了这一问题，这种模型将输入的序列分别按正向和反向分别处理了一次，最后生成两个上下文向量分别对应正序处理和逆序处理的结果，将这两个向量组合到一起来表示预测的结果。双向 RNN 可以用下面的图来表示：



5.3 注意力机制

5.3.1 什么是注意力机制

事实上当我们听到或者读写一个句子的时候，我们并不是平等的看待每一个单词，而是会把注意力集中在一些单词上，同样的，Seq2Seq 模型中，不同的部分输入也可以有不同级别的重要性，而不同部分的输入也可以用不一样的重要程度来看待不同部分的输入，比如在翻译问题中，输出的第一个单词往往是基于前几个输入单词，但是最后一个输出的单词往往取决于最后几个输入的单词。

注意力机制 (Attention Mechanisms) 可以给解码器网络在每个解码步骤中**提供查看整个输入序列的机会**，然后解码器可以决定哪部分输入单词是重要的，使得解码器可以了解到的原文信息不再局限于一个上下文向量。

5.3.2 注意力机制的一种 approach

我们假设输入的序列是 $(x_1, x_2, \ldots, x_n)$，目标结果 (也就是正确答案) 是 $(y_1, y_2, \ldots, y_m)$，这样一来，实现注意力机制需要如下几个步骤:

编码器 假设隐层生成的隐状态向量是 $(h_1, h_2, \ldots, h_n)$，其中 h_t 表示解码器在第 t 个编码步骤中的隐藏状态，按照传统的 Seq2Seq，最后输出的上下文向量实际上就是 h_n，并且编码器使用一个双向的

LSTM 并且提取了输入的每一个单词的上下文特征并用向量表示。

解码器 在解码器中，计算下一个隐藏状态 s_i 的过程可以表示为：

$$s_i = f(s_{i-1}, y_i, c_i) \tag{51}$$

其中 c_i 表示一个**上下文向量**，这个向量提取了第 i 步解码过程中的相关信息 (经典的 Seq2Seq 中只有一个上下文向量，而注意力机制中有若干个，实际上 Seq2Seq 模型中的**注意力机制就是为解码器中每个特定的位置，使用解码器中生成的所有隐状态向量生成了一个特定的上下文向量**)，我们需要计算的就是这个 c_i，首先需要对于每一个隐藏向量，计算一个" 分数"：

$$e_{i,j} = a(s_{i-1}, h_j) \tag{52}$$

这样一来就得到了一个分数序列，然后将其通过 softmax 函数进行标准化：

$$\alpha_{i,j} = \frac{\exp(e_{i,j})}{\sum_{k=1}^{n} \exp(e_{i,k})} \tag{53}$$

这里计算出的向量 α_i 被称为是注意力向量，然后可以计算新的上下文向量 c_i，用隐藏向量的加权和来表示：

$$c_i = \sum_{j=1}^{n} \alpha_{i,j} h_j \tag{54}$$

这样的向量 c_i 就从原文中提取了上下文的相关信息。注意力机制相当于保留了编码器在每一个 timestep 之后产生的上下文向量，并在解码器中通过加权的方式对原本每个 timestep 中产生的上下文向量进行一个加权求和，达到"集中注意力"的目的。

5.3.3 对齐 Alignment

基于注意力机制的模型可以学习到不同部分的重要性，这在机器翻译问题中可以理解成是一种对齐 (alignment)，注意力分数 $\alpha_{i,j}$ 起到了将原句子中的单词与目标句子中的单词 I 对齐的作用，这样一来我们可以用注意力分数 $\alpha_{i,j}$ 来构建一个对齐表，来表明原句子和翻译得到的句子的单词之间的对应关系。

值得一提的是，基于注意力机制的 Seq2Seq 模型在长句子和文章的翻译中表现效果相比于传统统计翻译模型和无注意力的 Seq2Seq 有了很大的提高。

5.4 序列模型解码器

除了 Seq2Seq 之外，还有一种从统计翻译方法演变而来的**序列模型解码器方法**，这种模型的通过最大化 $\mathbb{P}(\hat{s}|s)$ 来获得对于原句子 s 而言最好的翻译结果 $\hat{s}$，即：

$$\bar{s}* = \text{argmax}_{\bar{s}}(\mathbb{P}(\bar{s} \mid s)) \tag{55}$$

为此需要进行一系列单词空间中的搜索，找出使得概率最大化的结果，常见的搜索方法有：

- 贪婪解码 (Greedy Decoding) 的方式 (其实就是极大似然的思想，每一步选择出最有可能的单词)，但是这种编码方式是又问题的，贪婪解码没有办法撤回已经作出的决策，也就是说后面的可能会被前面的带偏，概率最高的也不一定就是最好的
- 穷举搜索 (Exhaustive search)：计算所有可能的翻译结果选择概率最大的作为翻译结果，而这样的方法计算量是非常大的，是一个 NP-Complete 的问题
- 定向搜索解码 (Beam Search Decoding)：一种 trade-off 的解决方式，对于每一步，记录下 k 个最有可能的翻译结果 (叫做 hypotheses 假设)，这里的 k 称为定向长度，一般是 5-10

5.5 机器翻译的效果评估

到这里为止我们已经对机器翻译系统的构建有了一定的了解，但在实际的生产环境中，需要翻译的样本可能充满了噪声 (比如语法错误)，这会大大降低翻译结果的精确程度，因此我们需要对机器翻译的效果来作出评估，帮助我们改进翻译系统。

5.5.1 子任务评估

评估翻译的效果的常见方法包括通过人类进行评估和使用其他子任务进行评估。但通过人类直接对翻译结果进行评估是非常低效而且高成本的，一般不会使用，我们可以使用一些常见的其他任务来评估一个翻译系统的翻译质量，比如使用一个对话系统的数据来评估翻译系统的表现。这样可以提高系统的泛化性能。

5.5.2 BLEU 算法

Bilingual Evaluation Understudy 简称为 BLEU 算法，这种算法将一个训练好的机器翻译模型和人类的翻译结果进行比较，生成一个 precision score，使用 n-gram 模型来评估机器翻译结果和人工翻译结果的相似度，也就是判断机器翻译结果的 n-gram 有没有出现在人工翻译的结果中，以此来衡

量结果的好坏。而 precision score 就可以定义为匹配的 n-gram 和 n-gram 总数的比值 (当然这里的 n 是需要预先设定的)，这样一来可以定义：

$$p_n = \frac{\text{\# matched n-grams}}{\text{\# total n-grams}} \tag{56}$$

并且可以定义惩罚函数：

$$\beta = e^{\min\left(0, 1-\frac{\text{len}_{\text{ref}}}{\text{lenMT}}\right)} \tag{57}$$

两个 len 代表了机器翻译生成的句子和人工翻译结果的句子长度，并且规定几何权重 $w = \frac{1}{2^n}$，这样一来总的 score 可以写成是 K 个测试样本的 score 之和，即：

$$\text{BLEU} = \beta \prod_{i=1}^{k} p_n^{w_n} \tag{58}$$

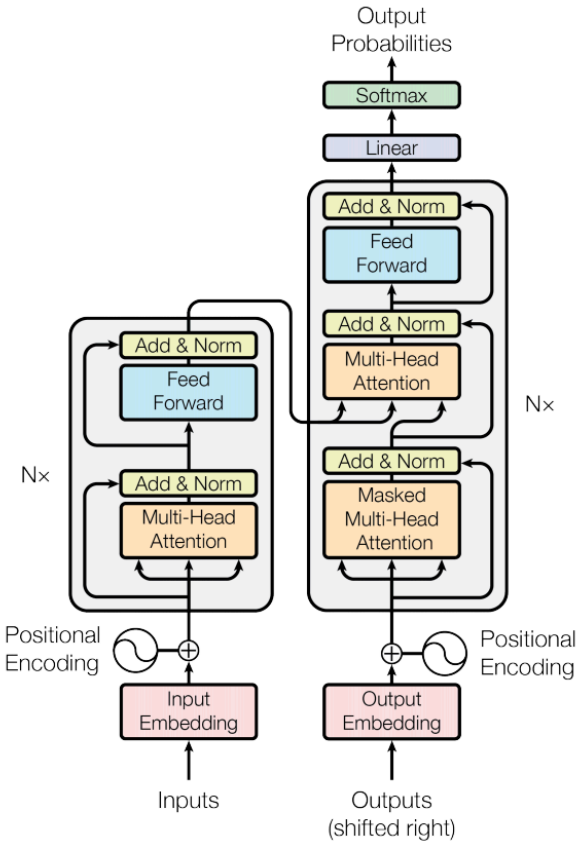
BLEU 算法通过用人类翻译评估机器翻译的方式取得了非常好的效果，成为了机器翻译评估标准的一种基准，但也存在着很多问题，比如这种评估方式只在语料库层面表现很好，而对于单个句子，如果一旦出现了非常不准确的翻译就可能导致 score 为 0，并且这种评估方式受人的影响很大，因为它只将翻译结果和一条人类翻译结果进行对比，这个过程的 bias 是非常大的。

6 Transformer 模型

RNN 虽然可以处理任意长度的输入，但是必须严格按照顺序来处理语句，并且需要 $O(N)$ 级别 (N 为输入语料的长度) 的步骤使相距较远的词语发生信息的交互，因此 RNN 存在如下三个问题：

- RNN 难以学习长距离的单词依赖性关系 (可能会有梯度消失或者梯度爆炸的问题)
- 必须以线性的方式来理解句子，但是句子往往不是以线性的方式可以正确理解的
- 因为需要按照输入顺序进行线性的处理，RNN 的可并行性差，不能并行计算

Transformer 模型是 2017 年的一篇论文《Attention is All You Need》提出的一种模型架构，这篇论文里只针对机器翻译这一种场景做了实验，全面击败了当时的 SOTA，并且由于 encoder 端是并行计算的，训练的时间被大大缩短了。Transformer 模型的结构如下图所示：



6.1 Transformer 模型架构

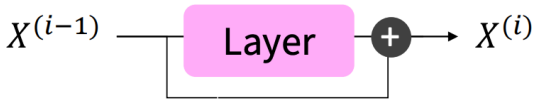
Transformer 模型也使用了编码器-解码器的架构，其中的架构细节包括：

6.1.1 编码器栈 Stacked Encoder

Transformer 模型的解码器使用了一系列编码器层 (论文中提出的是 6 层)，并且每个解码器层有 2 个子层构成，第一层是一个多端自注意力 (Multi-head self-attention) 层而第二层是一个简单的全连接前向传播层，并且子层的输出使用了 Residual Connection 的 trick(中文翻译应该是残差连接)，也就是说对每一个子层的输出进行如下操作：

$$y = \text{LayerNormolization}(x + \text{SubLayer}(x)) \tag{59}$$

这里的 x 表示当前层的输入，而 SubLayer(x) 表示这一层的输出，将二者相加之后进行标准化作为当前子层的最终输出。并且模型中的嵌入向量和所有子层的输出都是 512 维的向量，可以统一向量的 size 方便计算。标准化的过程可以用如下图表示：



6.1.2 解码器栈 Stacked Decoder

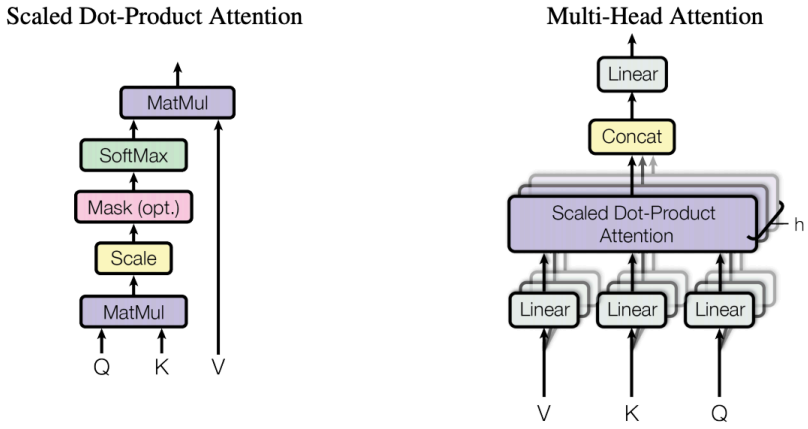
Transformer 模型的解码器也是由 6 层堆叠的解码器栈构成的，而每一层的结构和编码器相比多了一层多端注意力层用来对编码器栈中的输出进行注意力的计算，和编码器一样使用了标准化。同时也对自注意力层进行了一定的改变，避免了预测时对后文单词的依赖，这一操作保证了输出结果中的每个单词只依赖于在预测它之前已知的单词。

6.2 自注意力机制

一个注意力函数可以描述为将一系列查询 (query) 和一系列键值 (key-value) 对映射到一个输出结果上，并且 query，key，value 和 output 都是向量，可以理解为 output 就是一系列 values 的加权和，并且权重是取决于一系列 query 和 key 的。在自注意力机制中，query，key 和 value 是从相同的地方提取出来的，比如当前一层的输出结果是 $x_1, x_2, \ldots, x_t$ 时，我们可以令 $v_i = k_i = q_i = x_i$，确定了三种参数的值以后，注意力机制将进行如下操作：

$$\begin{aligned} e_{ij} &= q_i^T k_j \\ \alpha_{ij} &= \text{softmax}(e_{ij}) \\ \text{output} &= \sum_j \alpha_{ij} v_j \end{aligned} \tag{60}$$

Transformer 模型中用到的多端注意力机制通过以下方式计算得到：



6.2.1 Scaled Dot-Product Attention

Transformer 模型中的特别注意力 (particular attention) 被称为 Scaled Dot-Product Attention(我也不知道这东西要怎么翻译)，其输入包含维度为 d_k 的 query 和 key 以及维度为 d_v 的 values，我们将一系列 query 构成矩阵 Q，一系列的 key 和 value 构成矩阵 K 和 V，这样一来注意力可以用如下方式来计算：

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{61}$$

这个公式其实就是把上面的三个公式稍微调整之后写成矩阵的形式，其中 $\frac{1}{\sqrt{d_k}}$ 称为比例因子 (scaling factor)，会随着规模的增大而对 softmax 中的内容产生一定的约束。使用矩阵形式计算可以大大提高计算的效率，因为编程语言的矩阵计算库底层都对矩阵计算做了非常多的优化。

6.2.2 多端注意力机制

Scaled Dot-Product Attention 的 query，key 以及 value 是同源的，都来自于输入的词嵌入向量 X，而多端注意力机制中，我们可以将输入的词嵌入向量矩阵 X 先进行一次投影，分别投影到 d_k, d_k, d_v 维度的子空间中，形成 query，key，value 对应的矩阵 Q, K, V，这样一来多端注意力就可以表示为：

$$\begin{aligned} \text{MultiHead}(Q, K, V) = &\ \text{Concat}\ (\text{head}_1, \ldots,\ \text{head}_{\text{h}})\, W^O \\ \text{where head}_{\text{i}} = &\ \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right) \end{aligned} \tag{62}$$

这里的矩阵 $W_i^Q, W_i^K, W_i^V, W_i^O$ 都是投影矩阵，并且 $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ 而且 $W_i^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$ 作为 Transformer 模型中待优化的参数，而 Attention 就是上面提到的 Scaled

Dot-Product Attention，这种多端的注意力机制将注意力的计算分解成了 h 个平行的部分，每个部分使用不同的投影矩阵 W 将输入的词嵌入向量 X 投影到不同的空间中分别计算注意力，然后最后再汇总起来投影到一个 d_{model} 维度的向量中去，在论文中，作者使用了 $h = 8$，并且 $d_k = d_v = 64$，这样一来不仅注意力的提取方式更加多样化了，而且模型可以并行地计算不同部分的注意力，大大提高了计算的效率。

6.2.3 注意力机制在 Transformer 中的使用

Transformer 模型中用三种不同的方式来使用多端注意力机制，包括：

- 在" 编码器-解码器注意力" 层中使用，此时的 query 来自前面的解码器层，而 key 和 value 来自于解码器的输出，这使得解码器中的每个位置都可以注意到输入序列中的所有数据
- 编码器层中包含了一系列自注意力层，这时候的 query，key 和 value 都是同源的，都来自于前一层编码器层的输出，并且可以使编码器注意到上一层编码器输出结果的所有的位置
- 解码器层中也包含了一系列自注意力层，需要阻止 leftward information flow

6.3 Transformer 其他组件

6.3.1 Position-Wise 前馈网络

Transformer 模型中的编码器和解码器还包含了一系列全连接前馈网络层，并且对于序列中的每个位置独立起作用，这个前馈全连接层包含两次线性变换和一个 ReLU 激活函数：

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{63}$$

这些参数在同一个层的不同位置上是相同的，而在不同的全连接层上是不同的，并且 FFN 的输入和输出都是 512 维，并且有一个 2048 维的中间层。这种全连接也可以理解为是一种 kernel size 为 1 的卷积。

6.3.2 嵌入层和 softmax 层

这一部分和其他的序列模型一样，嵌入层负责将输入和输出用 d 维度的向量来表示，softmax 用于生成一个概率分布，在 Transformer 模型中输入时的嵌入层和输出时的嵌入层采用相同的权重矩阵，并且要将权重乘以 $\sqrt{d}$

6.3.3 位置编码 Positional Encoding

Transformer 模型中没有使用循环或者卷积，因此为了进行序列建模，需要增加一定的序列特征，Transformer 模型采用的办法就是在嵌入向量中注入位置编码，分别在编码器和解码器的底部 (这里指的应该是嵌入层之后的下一层)，而 Transformer 模型中使用的位置编码方式是：

$$\begin{aligned} \mathrm{PE}(pos, 2i) &= \sin(\frac{pos}{10000^{\frac{2i}{d_{model}}}}) \\ \mathrm{PE}(pos, 2i+1) &= \cos(\frac{pos}{10000^{\frac{2i}{d_{model}}}}) \end{aligned} \tag{64}$$

公式里的 pos 表示序列中的位置而 i 表示维度，虽然我很难理解为什么这种位置编码方式是有效的。

6.4 预训练 Pretraining

很多语言往往隐藏了复杂的形态或者词语结构，有限的词汇推断在很多语言中可能意味着更少的信息，子单词 (subword) 建模扩大了词汇表的容量和范围，可以更好地推断单词的结构。现代的主流模式是学习由部分单词 (子单词) 组成的词汇表

在训练和测试的时候，每个单词都会被分成一系列已知的子单词，字节对编码 (byte-pair encoding) 是一种常见的定义子单词词汇表的简单而有效的策略，它有这样几个步骤：

- 从一个只有表示“end-of-word”符号的词汇表开始
- 使用一系列语料，找到最相近的共有的一些邻近字母，组成子单词加入词汇表
- 用新的 subword 替换一系列字母对的实例，直到词汇表达到目标的大小

模型的预训练方式：主要有三种，分别是解码器的预训练，编码器的预训练以及编码器和解码器的同时预训练。我看了半天感觉预训练就是使用一大堆各种各样的语料库先给模型随便训练一通，初始化一系列参数，然后再根据具体的问题输入真正的训练集来进行训练，模型本身的结构并没有在预训练的过程中发生变化。

6.4.1 基于词嵌入的预训练

自然语言处理的核心思想是，我们理解一个单词必须根据上下文，这也就是分布式语义，并且导致了 word2vec 的产生，在实际的训练中，我们可以使用没有上下文的单词来进行词嵌入的预训练，也可以对整个模型进行预训练来初始化一些模型中的参数。预训练方法会隐藏部分的输入内容，并且训练模型来重新构建这些部分。预训练的技术可以帮助构建跟精确的语言表示，进行参数初始化并且采样概率分布。

6.4.2 解码器的预训练

我们可以将解码器当作一个语言模型来进行预训练，可以忽略训练 $p(w_t|w_{1:t-1})$ 的原始目的并进行微调来训练一个预测最后一个单词隐藏状态的分类器，这样的预训练方式在输出结果是一个序列的时候非常有用，可以预先训练出一个非常大的词汇表。一个非常成功的案例是 2018 年提出的 Generative Pretrained Transformer(GPT)，不过目前看不懂，就先不看了。

6.4.3 编码器的预训练

编码器必须要获取双向的上下文并进行编码，因此在预训练的时候不能使用语言模型来训练，可以用掩码来代替输入语料中的部分内容来，通过来预测这些单词完成编码器的预训练。BERT(Bidirectional Encoder Representations from Transfer) 是一个成功的案例，通过预训练的模型进行一定的微调就可以使用，其实这部分也没怎么看懂。

6.4.4 编码器和解码器的预训练

对于同时拥有编码器和解码器的神经网络我们可以综合使用以上两种预训练方式来同时训练编码器和解码器，用一些特殊的符号来代替输入文本中的部分内容，然后训练编码器-解码器来解码这一部分的内容，通过这样的方式来完成模型的预训练。

6.4.5 预训练意味着什么

通过预训练可以使模型预先从大范围的样本中学到大量的知识，然后根据实际任务的特殊需要再进行微调就可以获得最终想要的模型，预训练可以使模型预先学习到当前语言的一些语法，共指关系，文本语义，上下文关系，基本的语句结构等内容。