

Lecture 01

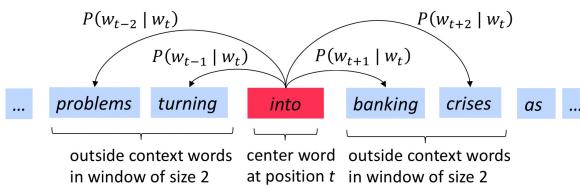
1. 语言是不确定的。
2. 我们应该怎么表示语言的含义，即语义？
3. 方法①：Wordnet：其中包含了词的含义和词的关系
缺点：丢失了词的细微差别；对于一个词的新含义没有收录；需要人工维护和更新；比较主观；不能计算词间的相似度
- 方法②：One-hot 编码：每个词一个向量。词典大小决定向量维度，是由 0、1 组成的稀疏向量
缺点：词典越大，维度越高；每个向量之间都是正交的，丢失了词之间的关系
- 方法③：用上下文表示词
4. 词向量：Word vector 也叫 word embedding
word representation

5. Word2Vec：一种训练词向量的框架.

思想：

- ① 大量的语料
- ② 在固定的词典中，每个词汇都有一个向量表示
- ③ 文本中每一个位置 t ，对应中心词和它的上下文单词
- ④ 使用 C 和 D 的词向量相似性来计算给定 C 时，D 的概率，反向
- ⑤ 调整词向量，使概率最大
使我们能更好地预测

- Example windows and process for computing $P(w_{t+j} | w_t)$



For each position $t = 1, \dots, T$, predict context words within a window of fixed size m , given center word w_t .

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^T \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w_{t+j} | w_t; \theta)$$

θ is all variables to be optimized

对于每一个位置 t ，在大小为 m 的固定窗口内预测上下文单词

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w_{t+j} | w_t; \theta)$$

损失函数

要最小化损失函数，需知道怎么求 $P(w_{t+j} | w_t)$

- Answer: We will use two vectors per word w :

- v_w when w is a center word
- u_w when w is a context word

- Then for a center word c and a context word o :

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

22

u, v 词向量随机初始化，通过多次迭代更新

softmax 函数

center word c and a context word o :

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

$u_o^T v_c$: 两个向量点乘，越相似，结果越大，从而归一化得到的概率值越大。模型的训练正是为了使得具有相似上下文的单词，具有相似的向量

分母：对整个词汇表进行标准化从而给出概率分布

P: 模型的输出概率代表我们词典中每个词有多大可能性和 c 同时出现。

通过梯度下降来调整

求导：

$$\begin{aligned}
 \frac{\partial}{\partial v_c} \log \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)} &= \frac{\partial}{\partial v_c} \log \exp(u_o^T v_c) - \frac{\partial}{\partial v_c} \log \sum_{w \in V} \exp(u_w^T v_c) \\
 &= \frac{\partial}{\partial v_c} u_o^T v_c - \dots \\
 &= u_o - \frac{1}{\sum_{w \in V} \exp(u_w^T v_c)} \times \frac{\partial}{\partial v_c} \sum_{w \in V} \exp(u_w^T v_c) \\
 &= u_o - \frac{1}{\sum_{w \in V} \exp(u_w^T v_c)} \times \sum_{w \in V} \frac{\partial}{\partial v_c} \exp(u_w^T v_c) \\
 &= u_o - \frac{1}{\sum_{w \in V} \exp(u_w^T v_c)} \times \sum_{w \in V} \exp(u_w^T v_c)
 \end{aligned}$$

A

f. 梯度函数

相似性

② Exponentiation makes anything positive

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

① Dot product compares similarity of o and c.
 $u^T v = u \cdot v = \sum_{i=1}^n u_i v_i$
Larger dot product = larger probability

③ Normalize over entire vocabulary to give probability distribution

- This is an example of the softmax function $\mathbb{R}^n \rightarrow (0,1)^n$

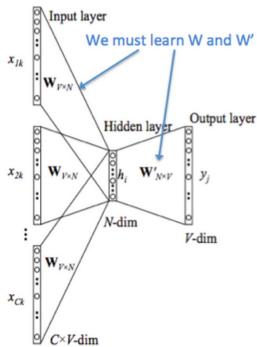
$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^n \exp(x_j)} = p_i$$

Open region

这个梯度函数实际上是一个实例。

7. Word2Vec 分类

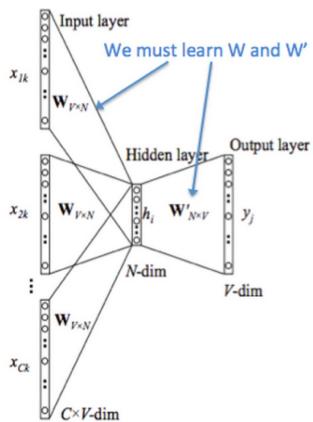
CBOW：给出周围词，预测中心词。



输入C个词向量，求平均
词向量 和词典中每个词
向量做点乘，结果作为
score，经过 softmax 函数得概率
值，概率最高的就是
预测出的中心词。

使用交叉熵作为损失函数。
通过随机梯度下降最小化损失

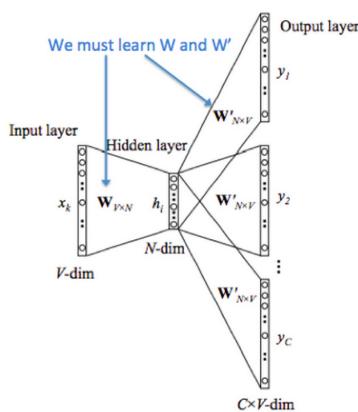
函数，更新词向量
这是 word2vec 的
优化。没有采用神
经网络线性变换加激活函数的方法。
简单对向量求和取平均，使多个词向量变为 1 个
词向量



1. 输入层：上下文单词的onehot. {假设单词向量空间dim为V, 上下文单词个数为C}
2. 所有onehot分别乘以共享的输入权重矩阵W. { V^*N 矩阵, N为自己设定的数, 初始化权重矩阵W}
3. 所得的向量 {因为是onehot所以为向量} 相加求平均作为隐层向量, size为 $1*N$.
4. 乘以输出权重矩阵W' { N^*V }
5. 得到向量 { 1^*V }, 激活函数处理得到V-dim概率分布 {PS:因为是onehot嘛, 其中的每一维都代表着一个单词}, 概率最大的index所指示的单词为预测出的中间词 (target word)
6. 与true label的onehot做比较, 误差越小越好

训练完毕后，输入层的每个单词与矩阵相乘得到的向量就是我们想要的词向量

Skip-gram: 给出中心词，预测周围词



输入中心词的词向量
与词库中每一个词向量
作点积 (score)
同样经过 softmax 层
最后输出 C 个概率最
大值

x : 词向量, 是 one-hot 向量

[https://blog.csdn.net/qq_16633405/
article/details/80227805](https://blog.csdn.net/qq_16633405/article/details/80227805)

Word2Vec 只关心模型训练完成后的副产物——模型参数(神经网络的权重)并
将这些参数，作为输入入的某种向量化表示。

训练完成后，输入一个 x 的 one-hot encoding $[1, 0, \dots, 0]$ ，则在输入层到隐、
藏层的权重里，只有对应 $|x|$ 这个位置的
权重被激活，这些权重个数与隐藏层
节点数一致，因此词向量降维了。

8. 负采样 (negative sampling)

为什么要负采样：

loss 的计算量过大，因为在 softmax 中对所有的 score 进行求和（维度与词汇表维度一致），计算概率。

negative sampling 每次让一个训练样本仅仅更新一小部分的权重参数，从而降低梯度下降过程中的计算量。

如果 vocabulary 大小为 1 万时，当输入样本 ("fox", "quick") 到神经网络时，“fox”经过 one-hot 编码，在输出层我们期望对应 “quick” 单词的那个神经元结点输出 1，其余 9999 个都应该输出 0。在这里，这 9999 个我们期望输出为 0 的神经元结点所对应的单词我们为 negative word。negative sampling 的想法也很直接，将随机选择一小部分的 negative words，比如选 10 个 negative words 来更新对应的权重参数。

Lecture 02

1. 复习 word2vec 的思想

- 对语料库中的每一个单词进行迭代
- 用词向量预测周围的词

2. 优化：梯度下降

3. 随机梯度下降：

① 每次选择一些 Sample

② 只对出现的词进行向量更新

4. 更多关于 word2vec 的细节：

Skip-gram cbow

提升效率的方法：负采样

5. 基于共现次数：

① window 为单位 \Rightarrow 可以捕捉语法、语义信息

- Example corpus:

- I like deep learning.
- I like NLP.
- I enjoy flying.

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

② 以文档为单位 \Rightarrow LSA

6. 基于共现次数的问题

- ① 随词典规模增大而增大
- ② 维度很高，占用大量存储空间
- ③ 后续的分类模型也会因为矩阵的稀疏而存在稀疏性问题，表现不佳。

7. 解决上述问题的方法：降维

问题：如何降维？

- ① 奇异值分解 SVD

8. 基于共现和直接预测的优缺点，比较

	快速	对于频次高 的词重 要性分配 不均	效果更多 可以捕捉多种 词汇相似模式	随语料库个 对数据的利 用不充分
充分利用数据	可以捕捉相似性 但在某些任务上 表现不好			

9. 在神经网络中综合 2 个流派的思想
类现概率的比值可以帮助我们对词汇的编码
(向量化)

	$x = \text{solid}$	$x = \text{gas}$	$x = \text{water}$	$x = \text{random}$
$P(x \text{ice})$	large	small	large	small
$P(x \text{steam})$	small	large	large	small
$\frac{P(x \text{ice})}{P(x \text{steam})}$	large	small	~ 1	~ 1

10. 如何在向量空间中计算类现概率比值?

A: Log-bilinear model: $w_i \cdot w_j = \log P(i|j)$

with vector differences $w_x \cdot (w_a - w_b) = \log \frac{P(x|a)}{P(x|b)}$

11. GloVe 正是这样一种 word embedding 模型
全称: Global Vectors for Word Representation
基于全局词频统计的词嵌入工具
它形成的向量可以捕捉到单词间的相似性
(similarity) - 类比性 (analogy) 等.

GloVe的实现分为以下三步：

- 根据语料库 (corpus) 构建一个共现矩阵 (Co-occurrence Matrix) X (什么是共现矩阵？)，矩阵中的每一个元素 X_{ij} 代表单词 i 和上下文单词 j 在特定大小的上下文窗口 (context window) 内共同出现的次数。一般而言，这个次数的最小单位是1，但是GloVe不这么认为：它根据两个单词在上下文窗口的距离 d ，提出了一个衰减函数 (decreasing weighting)：
$$decay = 1/d$$
 用于计算权重，也就是说距离越远的两个单词所占总计数 (total count) 的权重越小。

In all cases we use a decreasing weighting function, so that word pairs that are d words apart contribute $1/d$ to the total count.

- 构建词向量 (Word Vector) 和共现矩阵 (Co-occurrence Matrix) 之间的近似关系，论文的作者提出以下的公式可以近似地表达两者之间的关系：

$$w_i^T \bar{w}_j + b_i + \tilde{b}_j = \log(X_{ij}) \quad (1)$$

其中， w_i^T 和 \bar{w}_j 是我们最终要求解的词向量； b_i 和 \tilde{b}_j 分别是两个词向量的bias term。

当然你对这个公式一定有很多的疑问，比如它到底是怎么来的，为什么要使用这个公式，为什么要构造两个词向量 w_i^T 和 \bar{w}_j ？下文我们会详细介绍。

- 有了公式1之后我们就可以构造它的loss function了：

$$J = \sum_{i,j=1}^V f(X_{ij})(w_i^T \bar{w}_j + b_i + \tilde{b}_j - \log(X_{ij}))^2 \quad (2)$$

这个loss function的基本形式就是最简单的mean square loss，只不过在此基础上加了一个权重函数 $f(X_{ij})$ ，那么这个函数起了什么作用，为什么要添加这个函数呢？我们知道在一个语料库中，肯定存在很多单词他们在一起出现的次数是很多的 (frequent co-occurrences)，那么我们希望：

- 1. 这些单词的权重重要大于那些很少在一起出现的单词 (rare co-occurrences)，所以这个函数要是非递减函数 (non-decreasing)；
- 2. 但我们也希望这个权重过大 (overweighted)，当到达一定程度之后应该不再增加；
- 3. 如果两个单词没有在一起出现，也就是 $X_{ij} = 0$ ，那么他们应该不参与到loss function 的计算当中去，也就是 $f(x)$ 要满足 $f(0) = 0$

GloVe 模型 仅对单词共现矩阵中的非零元素训练，从而有效地利用全局统计信息，并生成具有有意义的子结构向量空间。

在相同的语料库、词汇、窗口大小和训练时间上，它的表现都优于 word2vec，它可以更快地实现更好的效果。

12. 如何评估词向量

- ① intrinsic 词向量类比，余弦相似度
- ② extrinsic

13. word sense and words ambiguity

如何处理多义问题

将常用词的上下文进行聚类，得到一些簇，从而将这个常用词分解为多个单词，如 bank_1 bank_2 等。

虽然这种方式比较粗糙，有时划分也不是很明确。

$$v_{\text{pike}} = \alpha_1 v_{\text{pike}_1} + \alpha_2 v_{\text{pike}_2} + \alpha_3 v_{\text{pike}_3}$$

Where $\alpha_1 = \frac{f_1}{f_1 + f_2 + f_3}$, etc., for frequency f

单词在词向量中的不同含义以线性叠加
(加权和) 的形式存在
 f 指出现频率

只是加权平均值就已经能取得很好的效果。

14. extrinsic 评估

所有接下来的任务

如命名实体识别、机器翻译等

Lecture 06

1. Language Modeling 语言建模

含义：根据前面的词预测下一个词是什么
感：将概率分配给一段文本

应用：输入法自动联想；检索补充

2. n-gram 语言模型

使用前n-1个单词预测第n个单词

$$P(\mathbf{x}^{(t+1)} | \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(1)}) = P(\mathbf{x}^{(t+1)} | \underbrace{\mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)}}_{n-1 \text{ words}})$$

prob of a n-gram $\rightarrow P(\mathbf{x}^{(t+1)}, \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)})$
prob of a (n-1)-gram $\rightarrow P(\mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)})$

如何求P：通过统计数据近似

$$\approx \frac{\text{count}(\mathbf{x}^{(t+1)}, \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)})}{\text{count}(\mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)})}$$

在很大的语料库中统计词频

“students opened their” occurred 1000 times

“students opened their books” occurred 400 times

- $\rightarrow P(\text{books} | \text{students opened their}) = 0.4$

“students opened their exams” occurred 100 times

- $\rightarrow P(\text{exams} | \text{students opened their}) = 0.1$

} Should we have
discarded the
“proctor” context?

n-gram 模型的问题

① 稀疏性问题：分子为0或分母为0
smoothing backoff

② 存储问题：n越大，语料库越大，所占存储空间越大

3. 使用n-gram模型生成文本 (sample)

发现：语义很不连贯，因为考虑的语境信息不足
但n的增大会造成更严重的稀疏性问题。
且存储空间更大

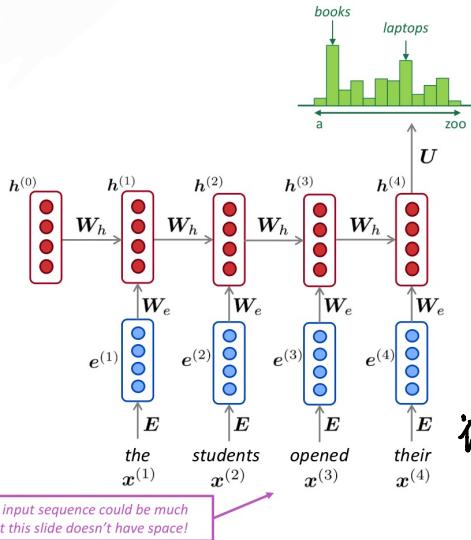
因此，我们需要能考虑任意长度语境文本，又不会出现上述问题的模型。

普通的神经网络，可以处理任意长度文本而不
会产生稀疏性问题，但随着 window-size 增大，权重
矩阵 W 也会增大，仍然很占存储空间。

最终，RNN 通过重复使用相同的 W 解决了
这个问题。

4. Recurrent neural network 循环神经网络

$$\hat{y}^{(4)} = P(x^{(5)} | \text{the students opened their.})$$



Note: this input sequence could be much longer, but this slide doesn't have space!

词向量矩阵, 如 Word2Vec

优点:

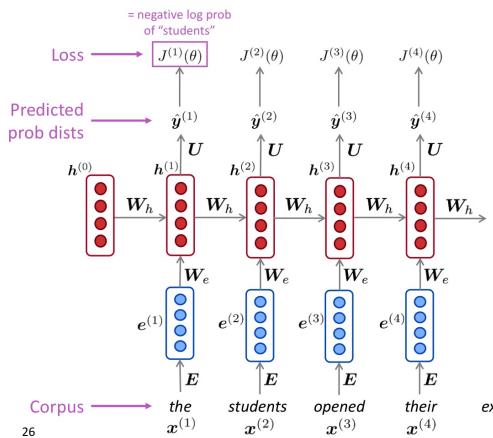
- Can process **any length** input
- Computation for step t (in theory) use information from **many steps back**
- **Model size doesn't increase** for longer input
- Same weights applied on every timestep, so there is **symmetry** in how inputs are processed.

缺点:

不能并行计算, 速度慢

实际上, 很难获得多步以前的信息

5. 如何训练一个 RNN 模型



26

反向公式定义

- Loss function on step t is cross-entropy between predicted probability distribution $\hat{y}^{(t)}$, and the true next word $y^{(t)}$ (one-hot for $x^{(t+1)}$):

$$J^{(t)}(\theta) = CE(y^{(t)}, \hat{y}^{(t)}) = - \sum_{w \in V} y_w^{(t)} \log \hat{y}_w^{(t)} = -\log \hat{y}_{x_{t+1}}^{(t)}$$

- Average this to get overall loss for entire training set:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^T J^{(t)}(\theta) = \frac{1}{T} \sum_{t=1}^T -\log \hat{y}_{x_{t+1}}^{(t)}$$

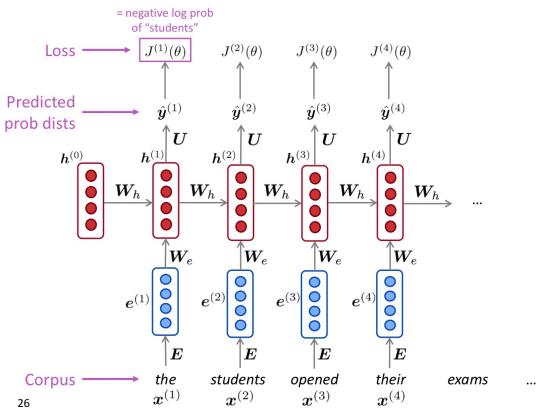
使用随机梯度下降

6. 如何进行反向传播

RNN 的反向传播也叫 BPTT (back-propagation through time)

由于 U 、 W_h 、 W_e 在各个位置是共享的，所以反向传播我们更新的是相同参数

以 $J^{(t)}(\theta)$ 为例，假设这是最后一层，我们要从这开始将误差向前传播



26

$$\frac{\partial J^{(4)}(\theta)}{\partial w}$$

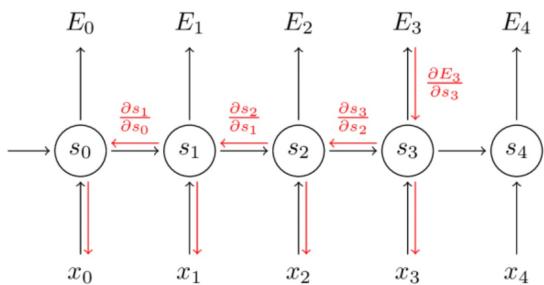
仅取决于 $h^{(4)}$, $\hat{y}^{(4)}$, $y^{(4)}$

$$\frac{\partial J^{(4)}(\theta)}{\partial w} = \frac{\partial J}{\partial \hat{y}^{(4)}} \frac{\partial \hat{y}^{(4)}}{\partial h^{(4)}} \frac{\partial h^{(4)}}{\partial w}$$

其中 $h^{(4)}$ 受 $h^{(3)}$ 影响
 \therefore 不能简单地将其视为常量
 需再次应用链式法则

$$\frac{\partial J^{(4)}}{\partial w} = \sum_{t=0}^4 \frac{\partial J^{(4)}}{\partial \hat{y}_t} \frac{\partial \hat{y}^t}{\partial h^t} \frac{\partial h^t}{\partial w}$$

相当于对每个 time-step 的参数 w 的梯度进行求和.



7. 使用RNN 生成文本

可以生成与训练语料同风格的文本

8. 评估指标：困惑度

是交叉熵损失函数的指数

9. 为什么要研究语言建模

① 是NLP 中的基准任务，帮助我们衡量在语言理解上的进步

② 可以用于许多现实任务中

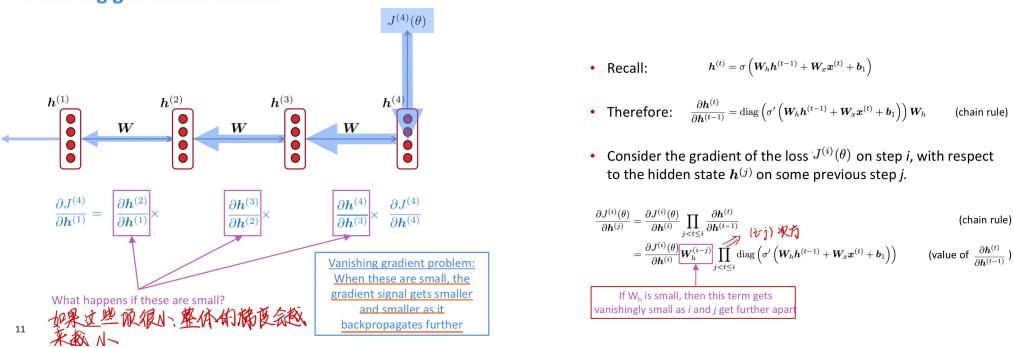
10. RNN 在词性标注、情感分类以及编码模块的应用

11. RNN 可能会出现梯度消失或梯度爆炸问题

Lecture 07

1. 梯度消失问题

Vanishing gradient intuition



- Consider matrix L2 norms: L2 正则化

$$\left\| \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(j)}} \right\| \leq \left\| \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(i)}} \right\| \left\| \mathbf{W}_h \right\|^{(i-j)} \prod_{j < t \leq i} \left\| \text{diag}(\sigma'(\mathbf{W}_h \mathbf{h}^{(t-1)} + \mathbf{W}_x \mathbf{x}^{(t)} + \mathbf{b}_t)) \right\|$$

如果权重矩阵的最大特征值小于1，则会出现梯度消失；大于1 → 梯度爆炸

2. 为什么梯度消失会产生问题？

因为越往后梯度越小，模型的权重仅根据相邻 state 进行更新

② 梯度可以看成是对过去影响未来的效果的一种度量。如果梯度变得很小，我们就不知道应该是下面的哪一种情况：

- t 和 $t+1$ 状态没有依赖关系
- 我们捕捉二者间关系的参数错了。

3. 为什么梯度爆炸会产生问题?

- If the gradient becomes too big, then the SGD update step becomes too big:

$$\theta^{new} = \theta^{old} - \overbrace{\hat{\alpha} \nabla_{\theta} J(\theta)}^{\text{learning rate}} \underbrace{\nabla_{\theta} J(\theta)}_{\text{gradient}}$$

步子太大，得到一些表现糟糕的参数

- This can cause bad updates: we take too large a step and reach a bad parameter configuration (with large loss)
- In the worst case, this will result in Inf or NaN in your network (then you have to restart training from an earlier checkpoint)

4. 解决梯度爆炸的方法: gradient clipping

如果梯度大于某个阈值，就将它变小后再进行梯度更新

```
ĝ ← ∂E / ∂θ
if ||ĝ|| ≥ threshold then
    ĝ ← threshold * ĝ / ||ĝ||
end if
```

核心思想：向同一方向走一小步

5. 如何解决梯度消失问题

梯度消失的主要问题在：RNN很难保存多个 time step 前的信息。

$$h^{(t)} = \sigma(W_h h^{(t-1)} + W_x x^{(t)} + b)$$

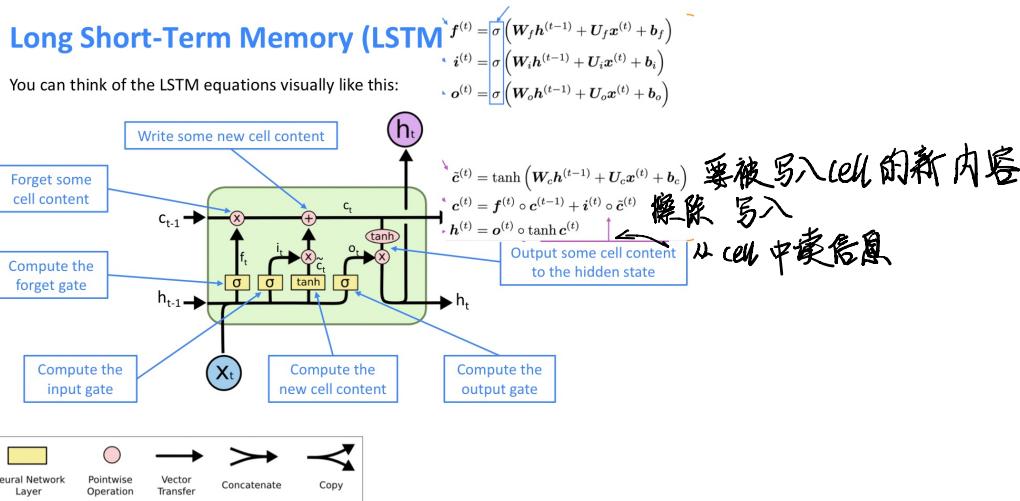
隐藏状态总被重写，因此 我们希望有单独的存储空间来保存我们希望在以后用到的信息。

6. LSTM

根据上述分析，LSTM 应运而生。
它是解决梯度消失问题的一种方案。

Long Short-Term Memory (LSTM)

You can think of the LSTM equations visually like this:



Source: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

25

每个 timestep，有 hidden state $h^{(t)}$ 和 cell state $c^{(t)}$
cell 存储长期信息，LSTM 可以从 cell 中读取、写入、擦除信息。

这部分操作分别由 Output 门、Input 门和 forget 门控制
门的取值在 0~1 之间，是动态变化的

7. 为什么 LSTM 解决了梯度下降问题

① LSTM 的结构让 RNN 更容易地保存了多个 timestep 前的信息

LSTM 并不保证完全解决梯度消失/爆炸的问题

8. Gated Recurrent Units (GRU)

- Proposed by Cho et al. in 2014 as a simpler alternative to the LSTM.
- On each timestep t we have input $x^{(t)}$ and hidden state $h^{(t)}$ (**no cell state**).

Update gate: controls what parts of hidden state are updated vs preserved

$$u^{(t)} = \sigma(W_u h^{(t-1)} + U_u x^{(t)} + b_u)$$

Reset gate: controls what parts of previous hidden state are used to compute new content

$$r^{(t)} = \sigma(W_r h^{(t-1)} + U_r x^{(t)} + b_r)$$

New hidden state content: reset gate selects useful parts of prev hidden state. Use this and current input to compute new hidden content.

$$\tilde{h}^{(t)} = \tanh(W_h(r^{(t)}) \circ h^{(t-1)} + U_h x^{(t)} + b_h)$$

$$h^{(t)} = (1 - u^{(t)}) \circ h^{(t-1)} + u^{(t)} \circ \tilde{h}^{(t)}$$

Hidden state: update gate simultaneously controls what is kept from previous hidden state, and what is updated to new hidden state content

How does this solve vanishing gradient?

Like LSTM, GRU makes it easier to retain info long-term (e.g. by setting update gate to 0)

update gate: 控制隐藏状态中的哪一部分要被更新或保留

(forget gate input gate in LSTM)

reset gate: 控制先前的隐藏状态中的哪一部分用于计算新内容. 可以看作用来判断以前隐藏状态的哪部分是有用的, 那部分使用

9. LSTM vs GRU

GRU: 计算更快 参数更少

没有证据表明哪一个模型可以完全胜过另一模型

LSTM 是很好的默认选择 (尤其在 long dependencies 和数据量很大时).

可以先使用 LSTM, 如果想提高效率, 再尝试 GRU

10.

梯度消失/下降是否只发生在RNN中.

对所有神经网络结构都有可能发生, 尤其是深度网络.

由于链式法则/非线性函数的选择.

反向传播时梯度可能变得很小.

低层的网络层学习得很慢(梯度小).

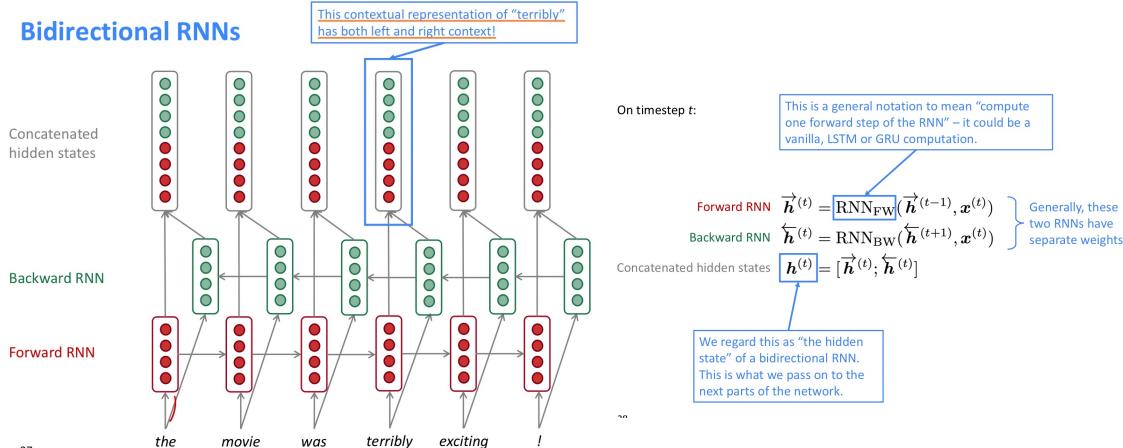
难以训练.

解决方案: 在网络中添加更多直连连接(使梯度能够传下)

11. Bidirectional RNNs 双向RNN

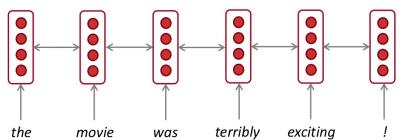
Motivation: 需要左右两侧的语境

Bidirectional RNNs



简化的结构图

$h^{(t)}$ 是前向和后向的相连.



The two-way arrows indicate bidirectionality and the depicted hidden states are assumed to be the concatenated forwards+backwards states.

注意：

- ① 双向RNN 只能应用于有完整的序列的情况。因此不能用它来做LM任务，因为LM需要根据前面的词预测后面的。
- ② 如果有完整的序列，双向 RNN 十分强大，它应该是默认的选择

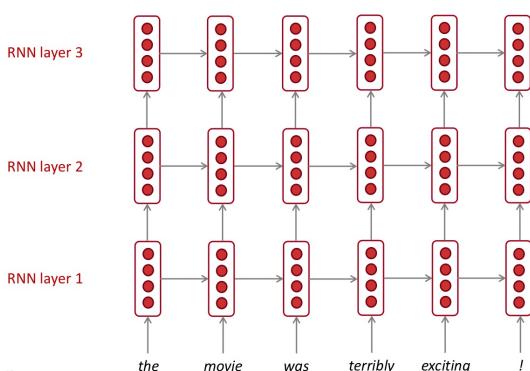
12. Multi-layer RNNs 多层 RNN

低层 RNN 计算低维特征，高层 RNN 计算高维特征。

Multi-layer RNNs

The hidden states from RNN layer i are the inputs to RNN layer $i+1$

应用于不同场景，所需层数也不同



Lecture 08

1. 在神经网络之前的机器翻译

① 1950年代早期：基于规则的，使用双语词典一一对应翻译

② 1990年代~2010年代：统计机器翻译
 $\arg\max_y P(y|x) = \arg\max_y P(x|y)P(y)$

$P(x|y)$ 翻译模型，知道小块的单词、短语如何翻译
 $P(y)$ 语言模型

如何学习 $P(x|y)$ ？

需要大量的平行语料库，并将 $P(x|y)$ 分为 $P(x, a|y)$
其中 a 是对齐方式 (alignment)，表示 2 个句子中特定的单词应该如何对应

对齐方式可能有一对多、多对一 或 多对多

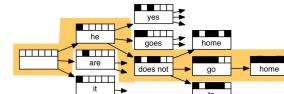
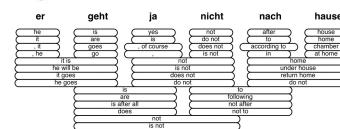
如何计算 $\arg\max$ ？

如果要计算每个可能的 y 的概率率，代价太大。

因此采用启发式搜索算法 (heuristic search algorithm)
丢弃概率率较低的假设

这个过程叫 decoding

Decoding for SMT

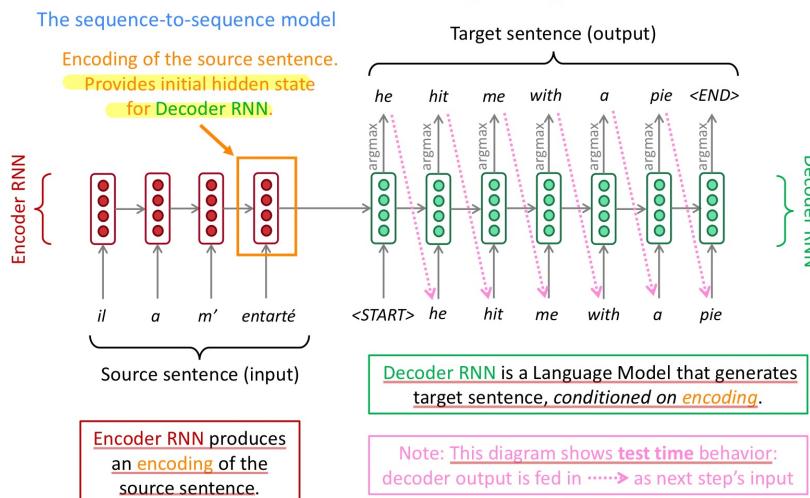


总结：统计机器翻译

- ① SMT是一个很大的研究领域
- ② 最优的系统十分复杂。
 - 有数百个重要的细节
 - 有许多特别设计的子组件
 - 需要很多特征工程
 - 需要编制、维护一些额外的资源
 - 需要大量人工的维护

2. 神经网络机器翻译

用于机器翻译的神经网络叫 sequence-to-sequence (seq2seq)
由2个RNN网络构成。



seq2seq不仅可以用于机器翻译，它还被应用在自动摘要、对话、解析、代码生成上。

Encoder

将一段任意长度的文本压缩为固定维度的向量很困难，尤其对复杂的机器翻译任务来说。因此 encoder 往往使用多层 LSTM。

往往按句子的倒序进行加工。

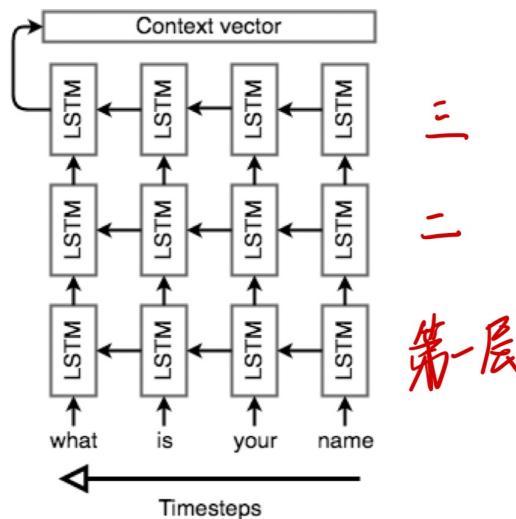


Figure 1: Example of a Seq2Seq encoder network. This model may be used to translate the English sentence "what is your name?" Note that the input tokens are read in reverse. Note that the network is unrolled; each column is a timestep and each row is a single layer, so that horizontal arrows correspond to hidden states and vertical arrows are LSTM inputs/outputs.

Decoder

通常也使用多层 LSTM

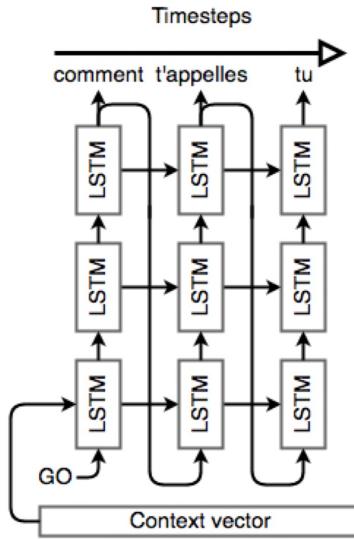


Figure 2: Example of a Seq2Seq decoder network. This decoder is decoding the context vector for "what is your name" (see Fig. 1 into its French translation, "comment t'appeles tu?") Note the special "GO" token used at the start of generation, and that generation is in the forward direction as opposed to the input which is read in reverse. Note also that the input and output do not need to be the same length.

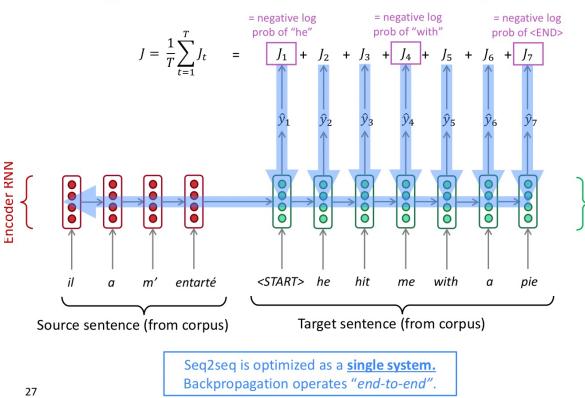
Neural Machine Translation (NMT) 的特点

① 是条件语言模型，因为对下一个词的预测是在源句子 x 的条件下进行的。

② NMT 直接预测 $P(y|x)$

如何训练 NMT？

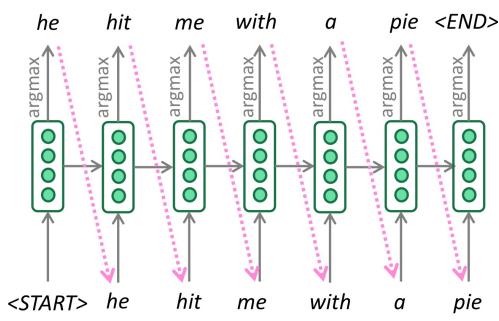
Training a Neural Machine Translation system



与用于预测时相比，训练时
不将 y 输入到下个 timestep。
 \hat{y} 仅用于计算 loss

Encoder 和 Decoder 是一起被
训练的

Greedy decoding



每一步选出概率最大的词进
下一个 time step
问题：不能回溯，一旦有一个词
错了，后面的预测可能都有
问题

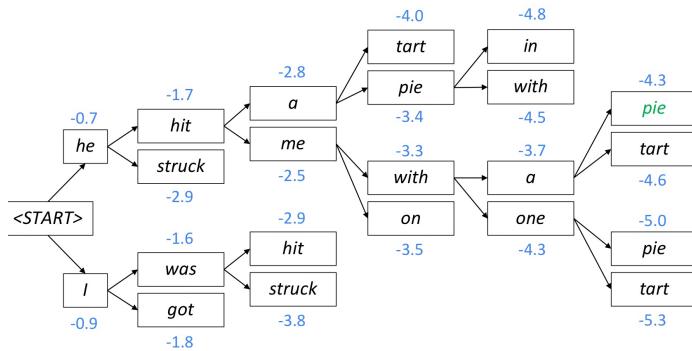
Exhaustive search encoding

尝试找所有可能的 y ,代价太大

Beam search encoding

综合上述两种 encoding 方法, 跟踪 k 个最有可能的翻译

Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$



停止搜索的标准:

在 greedy 方法中, 我们根据 $\langle \text{End} \rangle$ 标签来确定

但在 beam 方法中, 不同的路径出现 $\langle \text{End} \rangle$ 的时间不同
因此可以设置

T : T 个词以内的句子

K : 需要几个完整的翻译.

选择最高分句子的方法

- How to select top one with highest score?
 - Each hypothesis y_1, \dots, y_t on our list has a score
- $\text{score}(y_1, \dots, y_t) = \log P_{\text{LM}}(y_1, \dots, y_t | x) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$
- Problem with this: longer hypotheses have lower scores
 - Fix: Normalize by length. Use this to select top one instead:

$$\frac{1}{t} \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$$

如果不修正, 总是会选择最短的句子

NMT 的优点

Compared to SMT, NMT has many **advantages**:

- Better **performance** **更好的表现**
 - More **fluent** **更流畅**
 - Better use of **context** **更好地利用语境**
 - Better use of **phrase similarities** **更好地利用短语相似度**
- A **single neural network** to be optimized end-to-end
 - No subcomponents to be individually optimized
- Requires much **less human engineering effort** **需要的人力更少**
 - No feature engineering **无需特征工程**
 - Same method for all language pairs
对所有语言都具有一种方法

47

NMT 的缺点

Compared to SMT:

- NMT is **less interpretable** **可解释性差**
 - Hard to debug **难以调试**
- NMT is **difficult to control** **难以控制**
 - For example, can't easily specify rules or guidelines for translation **难以设置规则**
 - Safety concerns!
说脏话

如何评估机器翻译

BLEU

BLEU (Bilingual Evaluation Understudy)

1.12

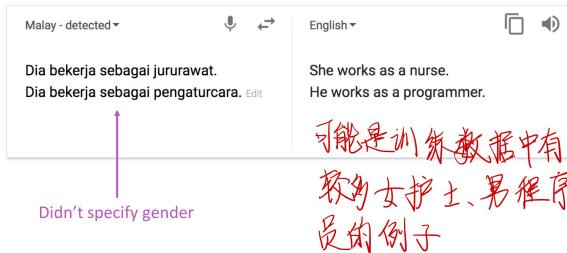
You'll see BLEU in detail
in Assignment 4!

- BLEU compares the **machine-written translation** to one or several **human-written translation(s)**, and computes a **similarity score** based on:
 - **n-gram precision** (usually for 1, 2, 3 and 4-grams)
 - Plus a penalty for **too-short system translations** ●
- BLEU is **useful** but **imperfect**
 - There are many valid ways to translate a sentence
 - So a **good** translation can get a **poor** BLEU score because it has low **n-gram overlap** with the human translation ⊕

3. 机器翻译仍存在的问题

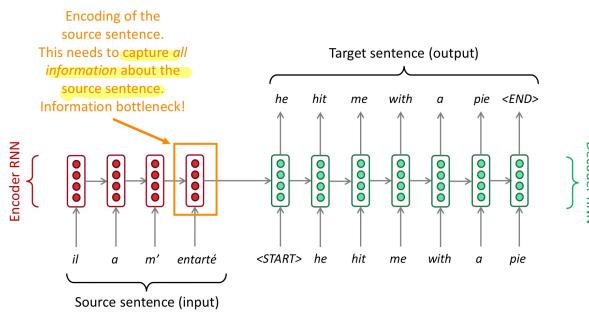
- ① 不包含在词典中的词
- ② 训练集、测试集所属的领域不同
- ③ 保留长文本的语境
- ④ 缺乏平行语料库
- ⑤ 难以运用常识进行翻译
- ⑥ 会有从训练集中学到的偏差

- NMT picks up **biases** in training data



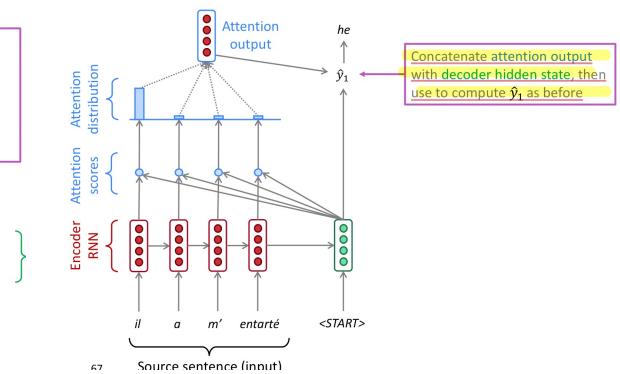
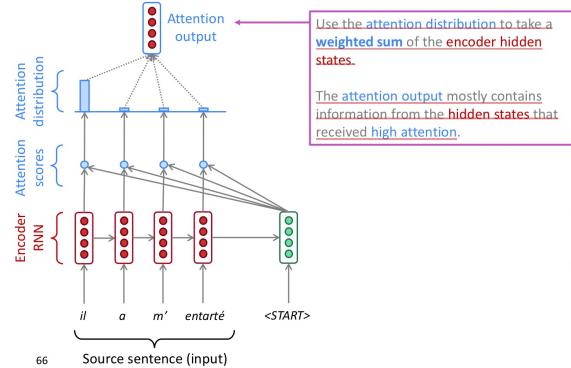
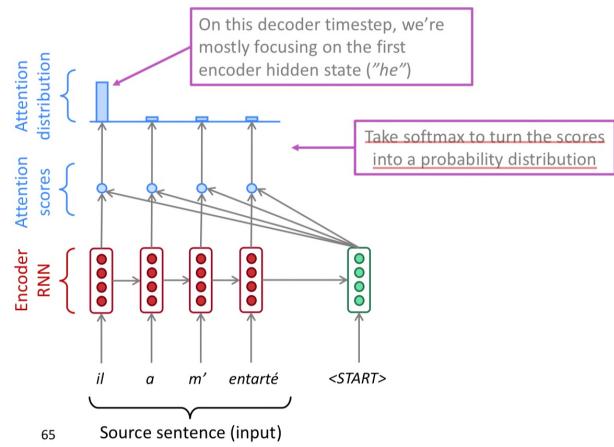
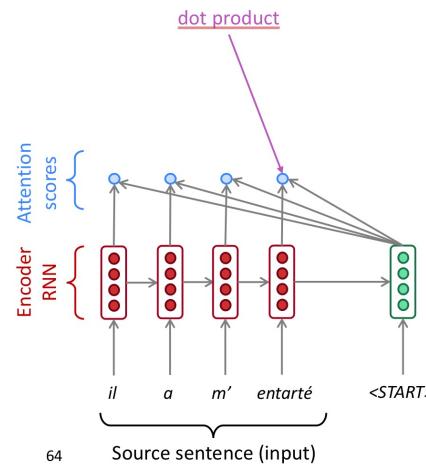
4. Attention 注意力机制

motivation: 信息瓶颈问题



如果源句的某些信息不在 encoder 输出的向量中，decoder 就无法正确地翻译原句

attention: 解决信息瓶颈的方法
 核心思想: decoder 的每一个 timestep, 都与 encoder 直接连接, 用来关注源句的某一特定部分



attention的优点

- ① 提升了NMT的表现，让decoder能关注某一特定部分
- ② 解决了信息瓶颈问题
- ③ 对解决梯度消失问题有帮助
- ④ 提供了一些可解释性，让NMT有了alignment的能力，而且是soft alignment
- ⑤ 可以高效地翻译长句子

attention是一种构造的机器学习方法

• More general definition of attention:

- Given a set of vector **values**, and a vector **query**, **attention** is a technique to compute a weighted sum of the values, dependent on the query.
- 加权的方式取决于query

- We sometimes say that the **query attends to the values**.
- For example, in the seq2seq + attention model, each decoder hidden state (query) attends to all the encoder hidden states (values).

75

attention的多种形式

通用框架

- We have some **values** $h_1, \dots, h_N \in \mathbb{R}^{d_1}$ and a **query** $s \in \mathbb{R}^{d_2}$

- Attention always involves:

1. Computing the **attention scores** $e \in \mathbb{R}^N$
2. Taking softmax to get **attention distribution** α :

$$\alpha = \text{softmax}(e) \in \mathbb{R}^N$$

3. Using attention distribution to take weighted sum of values:

$$a = \sum_{i=1}^N \alpha_i h_i \in \mathbb{R}^{d_1}$$

thus obtaining the **attention output** a (sometimes called the **context vector**)

There are multiple ways to do this

There are several ways you can compute $e \in \mathbb{R}^N$ from $h_1, \dots, h_N \in \mathbb{R}^{d_1}$ and $s \in \mathbb{R}^{d_2}$:

- **Basic dot-product attention:** $e_i = s^T h_i \in \mathbb{R}$
 - Note: this assumes $d_1 = d_2$
 - This is the version we saw earlier
- **Multiplicative attention:** $e_i = s^T W h_i \in \mathbb{R}$
 - Where $W \in \mathbb{R}^{d_2 \times d_1}$ is a weight matrix
- **Additive attention:** $e_i = v^T \tanh(W_1 h_i + W_2 s) \in \mathbb{R}$
 - Where $W_1 \in \mathbb{R}^{d_3 \times d_1}$, $W_2 \in \mathbb{R}^{d_3 \times d_2}$ are weight matrices and $v \in \mathbb{R}^{d_3}$ is a weight vector.
 - d_3 (the attention dimensionality) is a hyperparameter

Lecture 10

1. Motivation: Question Answering

我们拥有大量的全文文档集合，例如 web，但简单地返回相关文档的作用是有限的，我们更想直接得到问题的答案。

因此，可以将 Question answering 分解为 2 部分：

- ① 查找可能包含答案的文档 — 由信息检索实现
- ② 在一段或一份文档中找到答案 — 通常被称为“阅读理解”

2. MCTest：机器阅读理解领域的一个语料库

Passage (P) + Question (Q) → Answer (A)

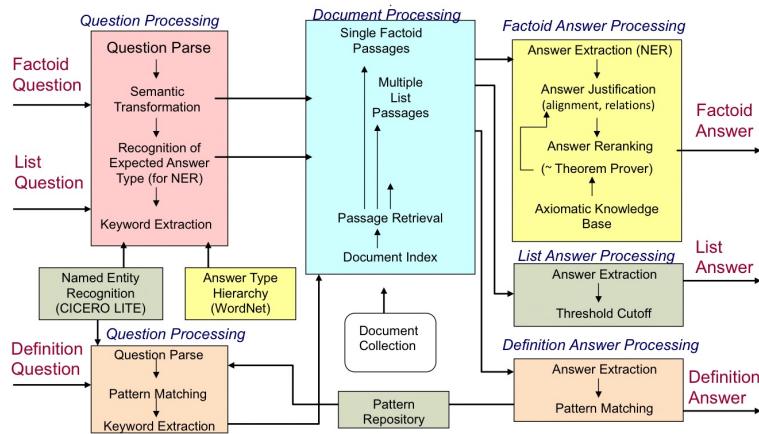
P Alyssa got to the beach after a long trip. She's from Charlotte. She traveled from Atlanta. She's now in Miami. She went to Miami to visit some friends. But she wanted some time to herself at the beach, so she went there first. After going swimming and laying out, she went to her friend Ellen's house. Ellen greeted Alyssa and they both had some lemonade to drink. Alyssa called her friends Kristin and Rachel to meet at Ellen's house.....

Q Why did Alyssa go to Miami? A To visit some friends

3. 一个 NLP QA 的例子

Turn-of-the Millennium Full NLP QA:

[architecture of LCC (Harabagiu/Moldovan) QA system, circa 2003]
Complex systems but they did work fairly well on “factoid” questions



一个非常复杂的多模块多组件的系统

- ① 首先对问题进行解析，识别出期望得到的答案类型
- ② 信息检索系统找到可能包含答案的段落，排序后进行选择

这样的QA系统在特定领域很有效，例如针对具体的问题。

4. Stanford Question Answering Dataset (SQuAD)

Question: Which team won Super Bowl 50?

Passage

Super Bowl 50 was an American football game to determine the champion of the National Football League (NFL) for the 2015 season. The American Football Conference (AFC) champion Denver Broncos defeated the National Football Conference (NFC) champion Carolina Panthers 24–10 to earn their third Super Bowl title. The game was played on February 7, 2016, at Levi's Stadium in the San Francisco Bay Area at Santa Clara, California.

Passage 是来自维基百科的一段文本，系统需要在文章中找出答案

100k examples

100k examples

答案总是来自段落中的某句话。

Answer must be a span in the passage

A.k.a. extractive question answering

提取式问答

5. SQuAD evaluation, v1.1

Private schools, also known as independent schools, non-governmental, or nonstate schools, are not administered by local, state or national governments; thus, they retain the right to select their students and are funded in whole or in part by charging their students tuition, rather than relying on mandatory taxation through public (government) funding; at some private schools students may be able to get a scholarship, which makes the cost cheaper, depending on a talent the student may have (e.g. sport scholarship, art scholarship, academic scholarship), financial need, or tax credit scholarships that might be available.

对每个问题，收集 3 个人的标准答案

系统在 2 个指标下计算得分。

① 精确匹配：0/1，是否匹配 3 个答案中的一个

② F1，被视为更可靠的指标，作为主要指标使用

- F1: Take system and each gold answer as bag of words, evaluate

$$\text{Precision} = \frac{TP}{TP+FP}, \text{Recall} = \frac{TP}{TP+FN}, \text{harmonic mean F1} = \frac{2PR}{P+R}$$

Score is (macro-)average of per-question F1 scores

这两个指标都忽视标点符号和冠词

6. SQuAD 2.0

SQuAD 1.0 的一个缺陷是，所有问题在文章中都有答案。系统对可能的答案进行排序并选择最好的一个。这就变成了排序任务，而系统实际并不知道这个答案有没有回答问题。

在 SQuAD 2.0 中，训练集的数据有言没有答案，测试集/验证集有言没有答案。

对于 No Answer 例子，no answer 得分为 1，任何其他结果得分都为 0。

如何构建系统

方法①：对于一个答案是否回答了问题设一个分数或值

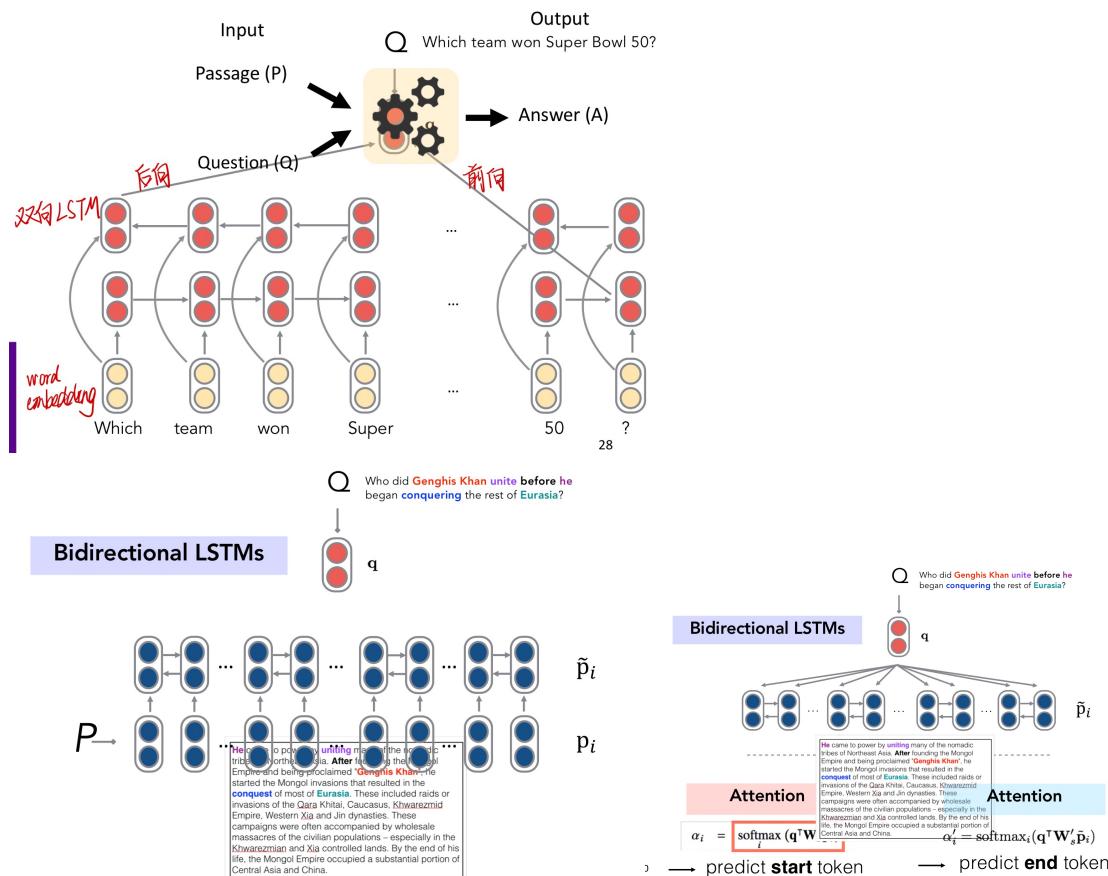
方法②：增加确认回答的组件，类似自然语言推理或答案验证

7. SQuAD 仍存在的局限性

- ① 只有基于原文内容的问题，没有 yes/no, 计数, 隐含问题
- ② 问题完全根据原文构造，这通常不是我们真正的需求，并且回答与原文高度重合，无论是单词还是句法结构。

8. Stanford Attentive Reader

是一个最简洁的、非常成功的阅读理解和 QA 框架



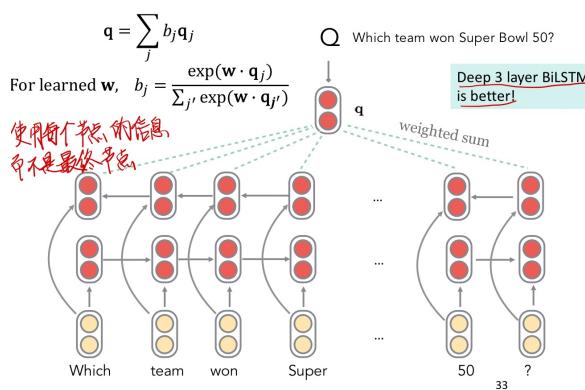
首先将问题用向量表示

- 对问题中的每个单词，查找其词向量
- 输入到双向 LSTM 中，并将最终的 hidden-state 连接

再处理文章

- 查找每个单词的词向量并输入到双向 LSTM 中
- 使用双线性 attention，将每个 LSTM 的表示与问题表示做运算，获得答案的起始、结束位置

Stanford Attentive Reader++

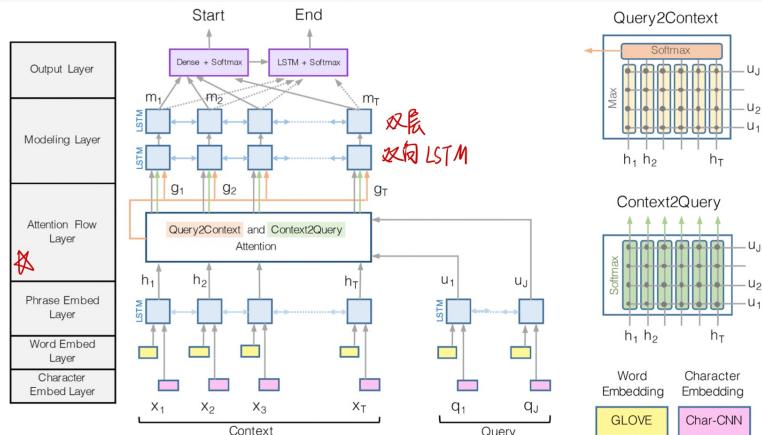


- \mathbf{p}_i : Vector representation of each token in passage
Made from concatenation of
 - Word embedding (GloVe 300d)
 - Linguistic features: POS & NER tags, one-hot encoded
 - Term frequency (unigram probability)
 - Exact match: whether the word appears in the question
 - 3 binary features: exact, uncased, lemma
 - Aligned question embedding ("car" vs "vehicle")
$$f_{align}(p_i) = \sum_j a_{i,j} \mathbf{E}(q_j) \quad q_{i,j} = \frac{\exp(\alpha(\mathbf{E}(p_i)) \cdot \alpha(\mathbf{E}(q_j)))}{\sum_{j'} \exp(\alpha(\mathbf{E}(p_i)) \cdot \alpha(\mathbf{E}(q_{j'})))}$$

34

Where α is a simple one layer FFNN

9. BiDAF: Bi-Directional Attention Flow for Machine Comprehension



核心思想是 Attention Flow Layer

认为：attention 应该双向流动，从上下文到问题，从问题到上下文

Context - to - Question: 哪些查询词与上文词最相关

Question - to - Context: 上文中最重要的单词相对于查询的加权和。

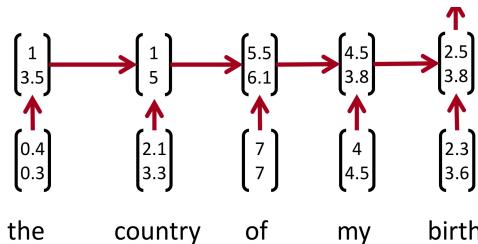
Lecture 11

1. 从 RNN 到 CNN

RNN 的问题：

不能捕捉没有上下文的短语

在最终向量中，过于偏重最后的单词



卷积网络的想法：

为每一个子序列计算一定长度的向量（不管子序列是否符合语法），然后对它们分组。

- Example: “tentative deal reached to keep government open” computes vectors for:
 - tentative deal reached, deal reached to, reached to keep, to keep government, keep government open

用于文本的一维卷积

tentative	0.2	0.1	-0.3	0.4
deal	0.5	0.2	-0.3	-0.1
reached	-0.1	-0.3	-0.2	0.4
to	0.3	-0.3	0.1	0.1
keep	0.2	-0.3	0.4	0.2
government	0.1	0.2	-0.1	-0.1
open	-0.4	-0.4	0.2	0.3

t,d,r	-1.0
d,r,t	-0.5
r,t,k	-3.6
t,k,g	-0.2
k,g,o	0.3

Apply a filter (or kernel) of size 3

$$\begin{aligned}
 & 0.2 \times 3 + 0.1 \times 1 - 0.3 \times 2 - 0.4 \times 1 \\
 & - 0.5 + 0.2 \times 2 - 0.3 + 0.1 \times 3 \\
 & - 0.1 - 0.3 + 0.2 + 0.4 \\
 & = -1.0
 \end{aligned}$$

size = 3.

带 padding 的卷积



Padding 填充

1D convolution for text with padding

\emptyset	0.0	0.0	0.0	0.0
tentative	0.2	0.1	-0.3	0.4
deal	0.5	0.2	-0.3	-0.1
reached	-0.1	-0.3	-0.2	0.4
to	0.3	-0.3	0.1	0.1
keep	0.2	-0.3	0.4	0.2
government	0.1	0.2	-0.1	-0.1
open	-0.4	-0.4	0.2	0.3
\emptyset	0.0	0.0	0.0	0.0

\emptyset, t, d	-0.6
t, d, r	-1.0
d, r, t	-0.5
r, t, k	-3.6
t, k, g	-0.2
k, g, o	0.3
g, o, \emptyset	-0.5

Apply a filter (or kernel) of size 3

3	1	2	-3
-1	2	1	-3
1	1	-1	1

11

得到与输入长度相同的输出
maintain size

多通道的卷积

3 channel 1D convolution with padding = 1

\emptyset	0.0	0.0	0.0	0.0
tentative	0.2	0.1	-0.3	0.4
deal	0.5	0.2	-0.3	-0.1
reached	-0.1	-0.3	-0.2	0.4
to	0.3	-0.3	0.1	0.1
keep	0.2	-0.3	0.4	0.2
government	0.1	0.2	-0.1	-0.1
open	-0.4	-0.4	0.2	0.3
\emptyset	0.0	0.0	0.0	0.0

filter 1	filter 2	...	
\emptyset, t, d	-0.6	0.2	1.4
t, d, r	-1.0	1.6	-1.0
d, r, t	-0.5	-0.1	0.8
r, t, k	-3.6	0.3	0.3
t, k, g	-0.2	0.1	1.2
k, g, o	0.3	0.6	0.9
g, o, \emptyset	-0.5	-0.9	0.1

Apply 3 filters of size 3

3	1	2	-3
-1	2	1	-3
1	1	-1	1

Could also use (zero)
padding = 2
Also called "wide convolution"

不同的filter以某种方式专注
于不同的特征，如是否 polite.
是否含有 ...

池化

conv1d, padded with max pooling over time

\emptyset	0.0	0.0	0.0	0.0
tentative	0.2	0.1	-0.3	0.4
deal	0.5	0.2	-0.3	-0.1
reached	-0.1	-0.3	-0.2	0.4
to	0.3	-0.3	0.1	0.1
keep	0.2	-0.3	0.4	0.2
government	0.1	0.2	-0.1	-0.1
open	-0.4	-0.4	0.2	0.3
\emptyset	0.0	0.0	0.0	0.0

\emptyset, t, d	-0.6	0.2	1.4
t, d, r	-1.0	1.6	-1.0
d, r, t	-0.5	-0.1	0.8
r, t, k	-3.6	0.3	0.3
t, k, g	-0.2	0.1	1.2
k, g, o	0.3	0.6	0.9
g, o, \emptyset	-0.5	-0.9	0.1

max p	0.3	1.6	1.4
-------	-----	-----	-----

Apply 3 filters of size 3

3	1	2	-3
-1	2	1	-3
1	1	-1	1

13

池化：用于总结一个卷积层的输出

最大池化：每个 filter 的最大值

平均池化



conv1d, padded with ave pooling over time

\emptyset	0.0	0.0	0.0	0.0
tentative	0.2	0.1	-0.3	0.4
deal	0.5	0.2	-0.3	-0.1
reached	-0.1	-0.3	-0.2	0.4
to	0.3	-0.3	0.1	0.1
keep	0.2	-0.3	0.4	0.2
government	0.1	0.2	-0.1	-0.1
open	-0.4	-0.4	0.2	0.3
\emptyset	0.0	0.0	0.0	0.0

\emptyset, t, d	-0.6	0.2	1.4
t, d, r	-1.0	1.6	-1.0
d, r, t	-0.5	-0.1	0.8
r, t, k	-3.6	0.3	0.3
t, k, g	-0.2	0.1	1.2
k, g, o	0.3	0.6	0.9
g, o, \emptyset	-0.5	-0.9	0.1

每个 filter 的平均值

常发现最大池化效果更好，
因为可以发现某个特征是否被激活

Apply 3 filters of size 3

3	1	2	-3	1	0	0	1	1	-1	2	-1
-1	2	1	-3	1	0	-1	-1	1	0	-1	3
14	1	1	-1	1	0	1	0	0	2	2	1

步长



Other less useful notions: stride = 2

步长

\emptyset	0.0	0.0	0.0	0.0
tentative	0.2	0.1	-0.3	0.4
deal	0.5	0.2	-0.3	-0.1
reached	-0.1	-0.3	-0.2	0.4
to	0.3	-0.3	0.1	0.1
keep	0.2	-0.3	0.4	0.2
government	0.1	0.2	-0.1	-0.1
open	-0.4	-0.4	0.2	0.3
\emptyset	0.0	0.0	0.0	0.0

Apply 3 filters of size 3

3	1	2	-3	1	0	0	1	1	-1	2	-1
-1	2	1	-3	1	0	-1	-1	1	0	-1	3
16	1	1	-1	1	0	1	0	0	2	2	1

局部最大池化



局部最大池化

Less useful: local max pool, stride = 2

\emptyset	0.0	0.0	0.0	0.0
tentative	0.2	0.1	-0.3	0.4
deal	0.5	0.2	-0.3	-0.1
reached	-0.1	-0.3	-0.2	0.4
to	0.3	-0.3	0.1	0.1
keep	0.2	-0.3	0.4	0.2
government	0.1	0.2	-0.1	-0.1
open	-0.4	-0.4	0.2	0.3
\emptyset	0.0	0.0	0.0	0.0

每2个做一次池化

Apply 3 filters of size 3

3	1	2	-3	1	0	0	1	1	-1	2	-1
-1	2	1	-3	1	0	-1	-1	1	0	-1	3
1	1	-1	1	0	1	0	1	0	2	2	1

前 k 个最大池化



前 k 个最大池化

conv1d, k-max pooling over time, k = 2

\emptyset	0.0	0.0	0.0	0.0
tentative	0.2	0.1	-0.3	0.4
deal	0.5	0.2	-0.3	-0.1
reached	-0.1	-0.3	-0.2	0.4
to	0.3	-0.3	0.1	0.1
keep	0.2	-0.3	0.4	0.2
government	0.1	0.2	-0.1	-0.1
open	-0.4	-0.4	0.2	0.3
\emptyset	0.0	0.0	0.0	0.0

2-max p 可以反映出一个特征被多次激活的情况

3	1	2	-3	1	0	0	1	1	-1	2	-1
-1	2	1	-3	1	0	-1	-1	1	0	-1	3
18	1	1	-1	1	0	1	0	1	0	2	2

Lecture 12

1. Morphology: Parts of words

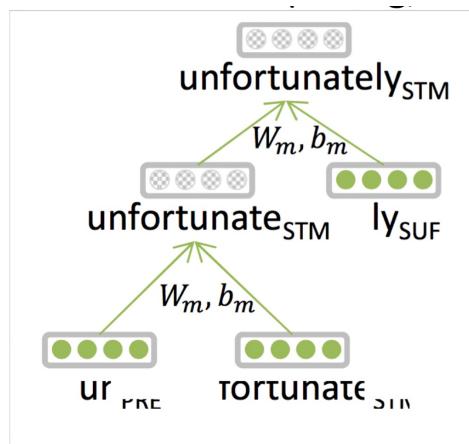
morphology 调法

semantic unit

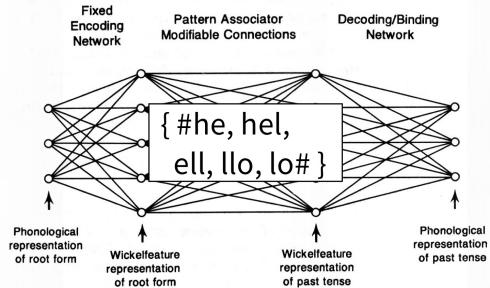
传统上，认为 morphemes (词素) 是最小的语义单位

- [[un [[fortun(e)]_{ROOT} ate]_{STEM}]_{STEM} ly]_{WORD}

在深度学习中，词法的研究较少，一种尝试是 DNN，被作为处理更多词汇量的一种方式（因为大多数未知的单词是新的形态或数字）



一种简单的替代方法是使用字符 n-gram



也有使用者想层的想法

2. Words in writing system 书写系统

不同的书写系统在表达单词的方式上各不相同
没有分词(在单词间放空格)：中文

附着词： $\text{هـا} + \text{نـا} + \text{قـالـ} + \text{فـ}$ = so+said+we+it

复合名词：Lebensversicherungsgesellschaftsangestellter

大多数深度学习 NLP 的工作都是从语言的书面形式开始的，这是一种容易处理的，现成的数据。但各种语言的书写系

统

- Phonemic (maybe digraphs) jiyawu ngabulu Wambaya
- Fossilized phonemic thorough failure English
- Syllabic/moraic જીયાવુ નગબુલુ Inuktitut
- Ideographic (syllabic) 去年太空船二号坠毁 Chinese
- Combination of the above インド洋の島 Japanese

3. Models below the word level 单词级别的模型遇到的困难

需要处理数量很大的开放词汇：巨大的、无限的单词空间

一丰富的词形 nejneobhospodařovávatelnějšímu

("to the worst farmable one")

一音译(特别是名字，在翻译中基本是音译)



4. Character - Level Models 字符级别的模型的2种思路

① 词向量可以由字符向量生成

- 为未知单词生成词向量

- 相似的拼写有相似的向量

- 解决了 OOV 问题

OOV: Out-of-Vocabulary, 数据集中的有些词不在词典中

② 连续语言可以作为字符处理: 即所有的语言处理均建立在字符序列上, 不考虑 word-level

这两种方法都被证明是非常成功的

深度学习模型可以存储和构建来自多个字母组的含义表示
从而模拟语素和更大单位的意义, 从而汇总形成语义

下面先讲纯字符模型

5. Purely character-level models 纯字符串级模型

- Initially, unsatisfactory performance
 - (Vilar et al., 2007; Neubig et al., 2013)
- Decoder only
 - (Junyoung Chung, Kyunghyun Cho, Yoshua Bengio. arXiv 2016).
- Then promising results
 - (Wang Ling, Isabel Trancoso, Chris Dyer, Alan Black, arXiv 2015)
 - (Thang Luong, Christopher Manning, ACL 2016)
 - (Marta R. Costa-Jussà, José A. R. Fonollosa, ACL 2016)

刚开始，表现不佳 (2013)

只能用于解码 (2016)

有前景的结果 (2016)

12

① English-Czech WMT 2015 Results

a pure character-level seq2seq NMT system
和 word-level 相比 同样好
但运行非常慢

System	BLEU
Word-level model (single; large vocab; UNK replace)	15.7
Character-level model (single; 600-step backprop)	15.9

13

source | Her **11-year-old** daughter, **Shani Bart**, said it felt a little bit **weird**

human | Její **jedenáctiletá** dcera **Shani Bartová** prozradila, že je to trochu **zvláštní**

char | Její **jedenáctiletá** dcera, **Shani Bartová**, říkala, že cítí trochu **divně**

word | Její <unk> dcera <unk> <unk> řekla, že je to trochu **divné**

word | Její **11-year-old** dcera **Shani**, řekla, že je to trochu **divné**

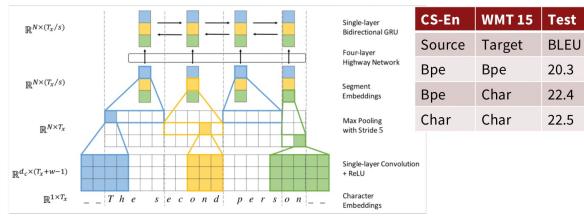
word-level 不能正确翻译不在 vocabulary 中的词

②

Fully Character-Level Neural Machine Translation without Explicit Segmentation

Jason Lee, Kyunghyun Cho, Thomas Hoffmann. 2017.

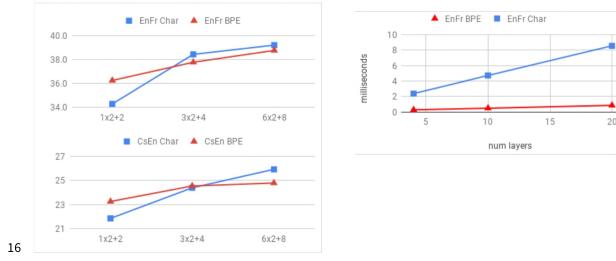
Encoder as below; decoder is a char-level GRU



③

Stronger character results with depth in LSTM seq2seq model

Revisiting Character-Based Neural Machine Translation with Capacity and Compression. 2018.
Cherry, Foster, Bapna, Firat, Macherey, Google AI



模型较小 . word-level
好

模型较大 . character-
level 好

6. Sub-word models 2个方向

① 与 word 级模型相同的架构

一但使用更小的单元：“word pieces”

- [Sennrich, Haddow, Birch, ACL'16a],
[Chung, Cho, Bengio, ACL'16].

② 混合架构

一主模型使用单词，其他使用字符级

Byte Pair Encoding

- [Sennrich, Haddow, Birch, ACL'16a],
[Chung, Cho, Bengio, ACL'16].

BPE 并不是深度学习的算法，但已经成为标准地、成功表示 pieces of words 的方法，可以获得一个有限的词典和无限且有效的词汇表。

它最初是一个压缩算法

- 最频繁的字节 → 一个新字节
- 用字节 n-gram 替换字节

BPE的分词算法：

- 比较简单，像自下而上的短序列聚类
- 将数据中的所有 Unicode 字符组成一个 Unigram 的词典
- 最常见的 n-gram pairs 视为一个新的 ngram

Dictionary

5 low
2 lower
6 newest
3 widest

Vocabulary

l, o, w, e, r, n, w, s, t, i, d

Start with all characters
in vocab

20

(Example from Sennrich)

Dictionary

5 low
2 lower
6 new es t
3 wide st

Vocabulary

l, o, w, e, r, n, w, s, t, i, d, es

unigram

Add a pair (e, s) with freq 9

..

Dictionary

5 low
2 lower
6 new est
3 wide st

Vocabulary

l, o, w, e, r, n, w, s, t, i, d, es, est

ngram pair

Add a pair (es, t) with freq 9

关于BPE的其他信息：

- ①有一个目标词汇量，当达到它时就停止制造 word pieces.
- ②2016年 WMT 排名第一，仍广泛应用于2018年 WMT

Wordpiece / Sentencepiece model BPE的变体

Google NMT 开发的变体：

V1: wordpiece model

V2: Sentencepiece model

不使用字符的 n-gram 频次，而是使用贪心近似来最大化语言模型的对数似然函数值。选择对应的 pieces.

- Wordpiece model tokenizes inside words
- Sentencepiece model works from raw text
 - Whitespace is retained as special token (_) and grouped normally
 - You can reverse things at end by joining pieces and recoding them to spaces

BERT 使用了 word piece 模型的一个变体：常用词在 vocabulary 中，其他单词由 wordpieces 组成

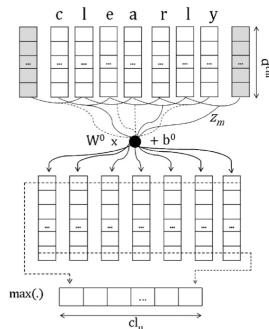
7. Character-level to build word-level

由字符级模型构建词模型

Learning Character-level Representations for Part-of-Speech Tagging (Dos Santos and Zadrozny 2014)

- Convolution over characters to generate word embeddings
- Fixed window of word embeddings used for PoS tagging

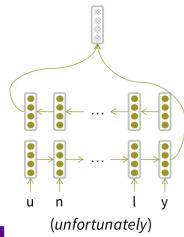
28



对字符进行卷积以生成词向量

为用于词性标注的词向量固定窗口.

Character-based LSTM to build word rep'n's



Bi-LSTM builds word representations
(unfortunately)

基于LSTM

双向LSTM形成word representations

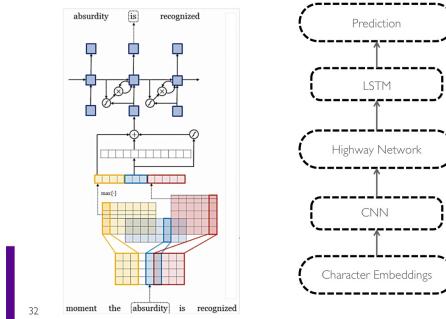
Ling, Luís, Marujo, Astudillo, Amir, Dyer, Black, Trancoso. *Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation*. EMNLP'15.

29

A more complex/sophisticated approach

Motivation

- Derive a powerful, robust language model effective across a variety of languages.
- Encode subword relatedness: *eventful*, *eventfully*, *uneventful*...
- Address rare-word problem of prior models.
- Obtain comparable expressivity with fewer parameters.



一个更复杂 / 精密的方法

动机：

- ① 形成一个强大的、健壮的语言模型，该模型在多种语言中有效
- ② 编码单词间的关联性：*eventful*, *eventfully*, ...
- ③ 解决现有模型的罕见字问题
- ④ 用更少的参数获得更好的表达

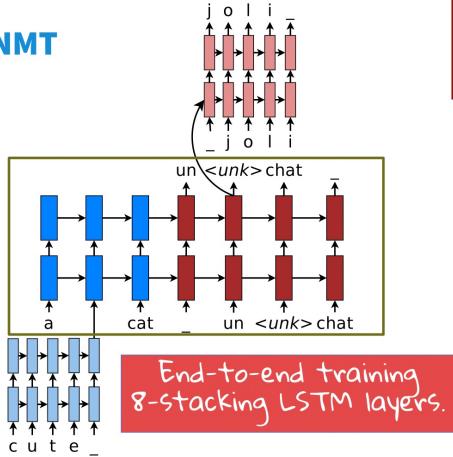
8. Hybrid NMT

一种两全其美的结构：

- ① 翻译大部分是单词级别的
- ② 只有在需要时进入字符级别

Hybrid NMT

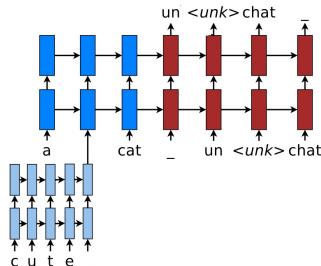
Word-level
(4 layers)



41

2-stage Decoding

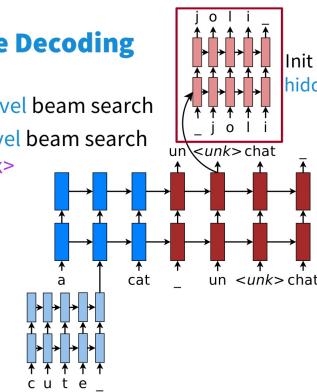
- Word-level beam search



42

2-stage Decoding

- Word-level beam search
- Char-level beam search for <unk>



43

而2部分完成 decode

結果：

English-Czech Results

- Train on WMT'15 data (12M sentence pairs)
 - newstest2015

Systems	BLEU	
Winning WMT'15 (Bojar & Tamchyna, 2015)	18.8	30x data 3 systems Large vocab + copy mechanism
Word-level NMT (Jean et al., 2015)	18.3	
Hybrid NMT (Luong & Manning, 2016)*	20.7	Then SOTA!

30x data
3 systems
Large vocab
+ copy mechanism

Then SOTA!

source	The author Stephen Jay Gould died 20 years after diagnosis .
human	Autor Stephen Jay Gould zemřel 20 let po diagnóze .
char	Autor Stepher Stepher zemřel 20 let po diagnóze .
word	Autor Stephen Jay <unk> zemřel 20 let po <unk>.
hybrid	Autor Stephen Jay Gould zemřel 20 let po po .
	Autor Stephen Jay <unk> zemřel 20 let po <unk>.
	Autor Stephen Jay Gould zemřel 20 let po diagnóze .

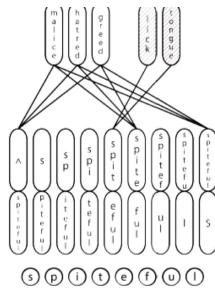


9. Chars for word embeddings

①

A Joint Model for Word Embedding and Word Morphology
(Cao and Rei 2016)

- Same objective as w2v, but using characters
- Bi-directional LSTM to compute embedding
- Model attempts to capture morphology
- Model can infer roots of words



一种用于词向量和单词形态学的联合模型

- 与 word2vec 目标相同，但使用字符
- 双向 LSTM 计算单词表示
- 模型试图捕获词法
- 模型可以推断单词词根

② FastText embeddings

用子单词信息丰富单词向量

Enriching Word Vectors with Subword Information

Bojanowski, Grave, Joulin and Mikolov. FAIR. 2016.

<https://arxiv.org/pdf/1607.04606.pdf> • <https://fasttext.cc>

- 类似于 word2vec 的词表示，但在稀有词和带有很多的语素上做得更好
- Aim: a next generation efficient word2vec-like word representation library, but better for rare words and languages with lots of morphology
 - An extension of the w2v skip-gram model with character n-grams

形态学

Lecture 13

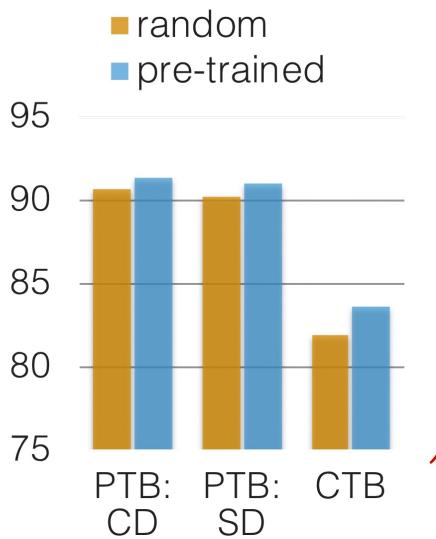
1. Representations for word 词表示

我们已经学习了一种词的表示方法：词向量 (Word2Vec, Glove, FastText 等)

Pre-trained word vectors 预训练词向量

我们可以先随机初始化词向量，然后根据特定任务的要求训练。

但在大多数情况下，使用预训练的词向量 效果更好，
因为可以在更多的数据上训练出更多词汇。



对未知词如何形成词向量

简单且常见的方法：设置一个词频阈值 f ，低于 f 的即为不常见词，收到 $\langle \text{UNK} \rangle$ 中。对 $\langle \text{UNK} \rangle$ 训练出一个词向量。

问题：没有办法区分不同的未知词

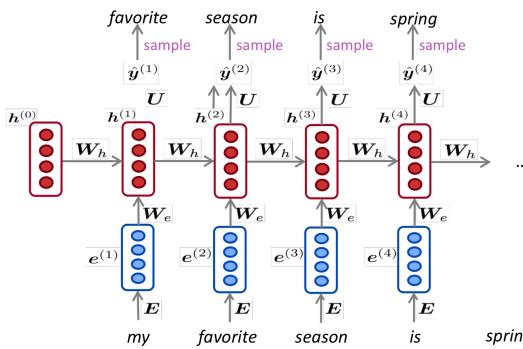
解决方案：

- ① 用字符级模型构建未知词向量
- ② 如果出现未知词，就随机初始化一个向量，加入到 vocabulary 中
- ③ 对未知词进行分类，每一类有一个向量

以上的词表示方法存在问题

- ① 一个词有多种含义，如 star 有天文学中的含义和娱乐圈中的含义。我们需要进行细粒度的词义消歧
- ② 我们对一个词只有一种表示，但单词有不同的维度，如词义、词性等。如 arrive 、 arrival 含义相同，但使用的位置不同。

是否有办法解决上述问题
学过的神经网络语言模型



这些语言模型在每一个位置生成一个基于特定上下文的表示。

2. "Pre-ELMo" - Tag LM

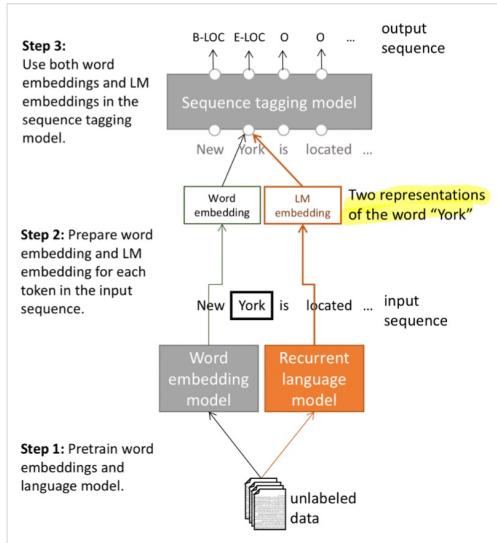
核心思想：想获取词在上下文中的意思，但标准的、
基于任务的RNN只在小的、有监督的数据上训练。
为什么不采用半监督的方式在大型的无标签数据集上训练NLM，而不仅是词向量？

什么是命名实体识别 (Name Entity Recognition, NER)

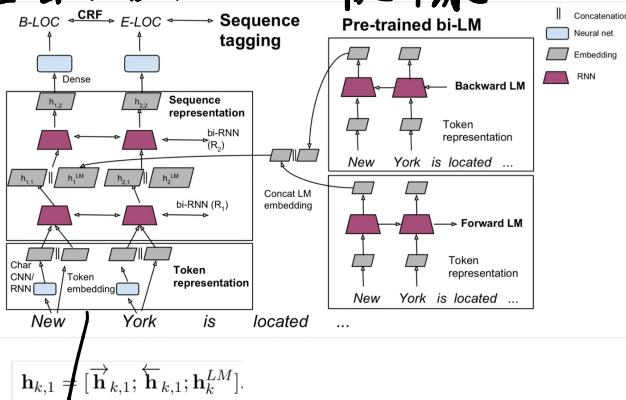
查找和分类文本中的实体

- The decision by the independent MP Andrew Wilkie to withdraw his support for the minority Labor government sounded dramatic but it should not further threaten its stability. When, after the 2010 election, Wilkie, Rob Oakeshott, Tony Windsor and the Greens agreed to support Labor, they gave just two guarantees: confidence and supply.

Person
Date
Location
Organization



在 NER 任务中：



将预训练 LM 生成的向量和 word embedding 的向量连接起来

$$\mathbf{h}_{k,1} = [\overrightarrow{\mathbf{h}}_{k,1}; \overleftarrow{\mathbf{h}}_{k,1}; \mathbf{h}_k^{LM}]$$

如 word2vec

Tag LM 在 Billion word Benchmark 数据集中训练了 8 亿个词汇，观察结果如下：

- ① 在有监督的数据集下训练的 LM 并不会受益。
- ② 更大的模型更好（训练的数据更多）

基于特定任务的 Bi-LSTM 的观察结果：

- ① 仅使用 LM 向量来预测效果不好。

ELMo: Embeddings from Language Model

不同于以往的一个词对应一个向量，是固定的。在 ELMo 的世界里，预训练好的模型不再只是向量对应关系。使用时，将一段话输入模型，模型会根据上下文来推断每个词对应的词向量，对多义词，可结合语境进行理解。

- 用大语境（不是语境窗口）来学习词向量
- 学习一个深层的 Bi-LSTM，在预测时用它每一层的信息

Peters et al. (2018): ELMo: Embeddings from Language Models

- Train a bidirectional LM
- Aim at performant but not overly large LM:
 - Use 2 biLSTM layers
 - Use character CNN to build initial word representation (only)
 - 2048 char n-gram filters and 2 highway layers, 512 dim projection
 - User 4096 dim hidden/cell LSTM states with 512 dim projections to next input
 - Use a residual connection
 - Tie parameters of token input and output (softmax) and tie these between forward and backward LMs

- This is an innovation that improves on just using top layer of LSTM stack

$$\begin{aligned} R_k &= \{\mathbf{x}_k^{LM}, \vec{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} \mid j = 1, \dots, L\} \\ &= \{\mathbf{h}_{k,j}^{LM} \mid j = 0, \dots, L\}, \end{aligned}$$

仅使用最顶层的 LSTM
输出

更好的方法：使用权重

$$\text{ELMo}_k^{\text{task}} = E(R_k; \Theta^{\text{task}}) = \gamma^{\text{task}} \sum_{j=0}^L s_j^{\text{task}} \mathbf{h}_{k,j}^{LM}$$

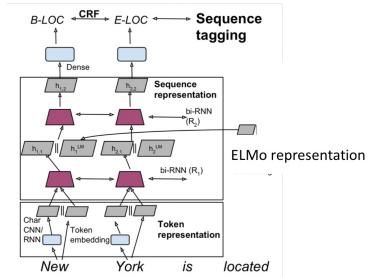
- γ^{task} scales overall usefulness of ELMo to task;
- s^{task} are softmax-normalized mixture model weights

针对具体的NLP任务，训练权重

Peters et al. (2018): ELMo: Use with a task

- First run biLM to get representations for each word
- Then let (whatever) end-task model use them
 - Freeze weights of ELMo for purposes of supervised model
 - Concatenate ELMo weights into task-specific model
 - Details depend on task
 - Concatenating into intermediate layer as for TagLM is typical
 - Can provide ELMo representations again when producing outputs, as in a question answering system

ELMo used in a sequence tagger



$$\mathbf{h}_{k,1} = [\vec{\mathbf{h}}_{k,1}; \overleftarrow{\mathbf{h}}_{k,1}; \mathbf{h}_k^{LM}]$$

ELMo的效果：

ELMo results: Great for all tasks

TASK	PREVIOUS SOTA	OUR BASELINE	ELMO + BASELINE	INCREASE (ABSOLUTE/RELATIVE)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17
SRL	He et al. (2017)	81.7	81.4	84.6
Coref	Lee et al. (2017)	67.2	67.2	70.4
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5

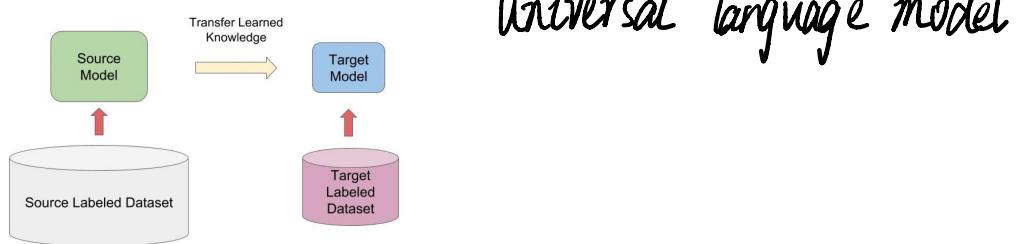
ELMO: Weighting of layers

2层 Bi-LSTM 层有不同的用处

低层擅长低级语法一词性标注 句法依赖 命名实体识别
高层适合更高级的语义一情感 语义角色标注 回答

ULMfit (与ELMo 同一时期的另一工作)

同样的 idea: 迁移 NLM 的知识
应用于文本分类

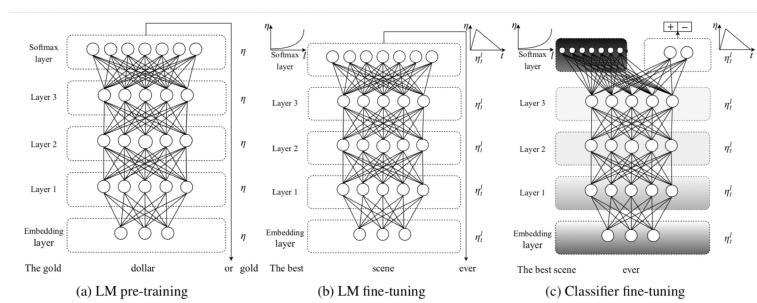


ULMFiT方法分为以下几个阶段：

LM pre-training: 在一个很大的通用领域语料库上训练 LM, 捕获不同层次文本的一般特征

LM fine-tuning: 针对 target-domain 微调 LM, 学习不同层次文本在 target domain 上的特征

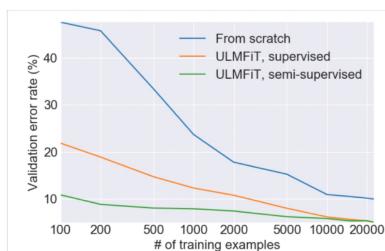
Classifier fine-tuning: 训练模型的顶层结构



ULMFiT的效果

- Text classifier error rates

	Model	Test	Model	Test
IMDb	CoVe (McCann et al., 2017)	8.2	CoVe (McCann et al., 2017)	4.2
	oh-LSTM (Johnson and Zhang, 2016)	5.9	TBCNN (Mou et al., 2015)	4.0
	Virtual (Miyato et al., 2016)	5.9	LSTM-CNN (Zhou et al., 2016)	3.9
	ULMFiT (ours)	4.6	ULMFiT (ours)	3.6





ULMfit	GPT	BERT	GPT-2
Jan 2018	June 2018	Oct 2018	Feb 2019
Training:	Training	Training	Training
1 GPU day	240 GPU days	256 TPU days ~320–560 GPU days	~2048 TPU v3 days according to a reddit thread



33

All of these models are Transformer architecture models ... so maybe we had better learn about Transformers?

ULMfit	GPT	BERT	GPT-2
Jan 2018	June 2018	Oct 2018	Feb 2019
Training:	Training	Training	Training
1 GPU day	240 GPU days	256 TPU days ~320–560 GPU days	~2048 TPU v3 days according to a reddit thread



36

4. The Motivation for Transformers

- ① 我们想要并行处理但 RNN 只能顺序执行
- ② 尽管有 GPU 加速, 有强大的 LSTM, 但 RNN 类网络仍需要 attention mechanism 来处理长范围依赖
- ③ 如果 attention 可以让我们访问任一状态, 是否可以只使用 attention 不用 RNN?