深度序列模型

杨 猛

中山大学

机器智能与先进计算
教育部重点实验室

深度序列模型

- 问题背景
- 循环神经网络(RNN)
- 循环神经网络的变体
- Attention和RNN
- RNN中的经典Attention
- Bi-Attention
- Self-Attention

第一部分：问题背景

深度序列模型

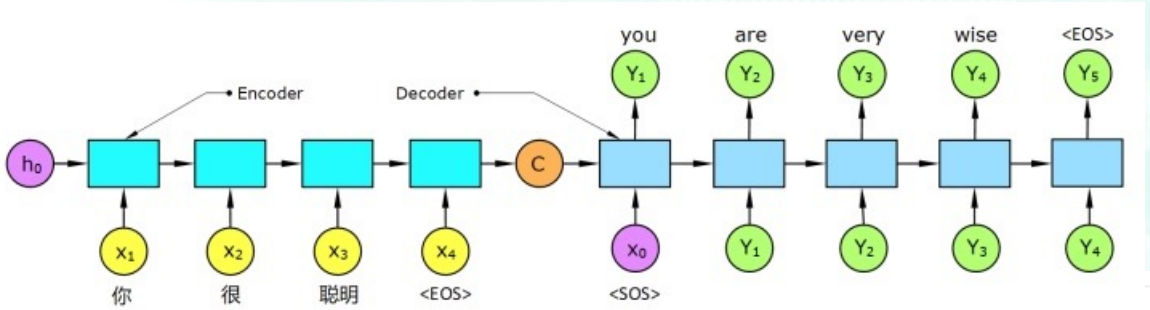
- 背景
  - 许多场景会产生以序列形式存在的数据
  - 不同时刻的数据具有某种相关性
  - 输入输出的序列长度未定

深度序列模型

- 处理序列数据(时序)
  - 普通的深度神经网络
    - 较难处理长度不定的序列数据
  - 卷积神经网络
    - 参数共享：空间上
    - 将时间序列按空间展开可以处理，但通常只可以处理较短距离的信息依赖。
  - 深度序列模型
    - 参数共享：时间(广义)上
    - 模型能够扩展到不同不同长度的样本上并进行泛化。（如果我们在每个时间点都有一个单独的参数，我们不但不能泛化到训练时没有见过序列长度，也不能在时间上共享不同序列长度和不同位置的统计强度。）

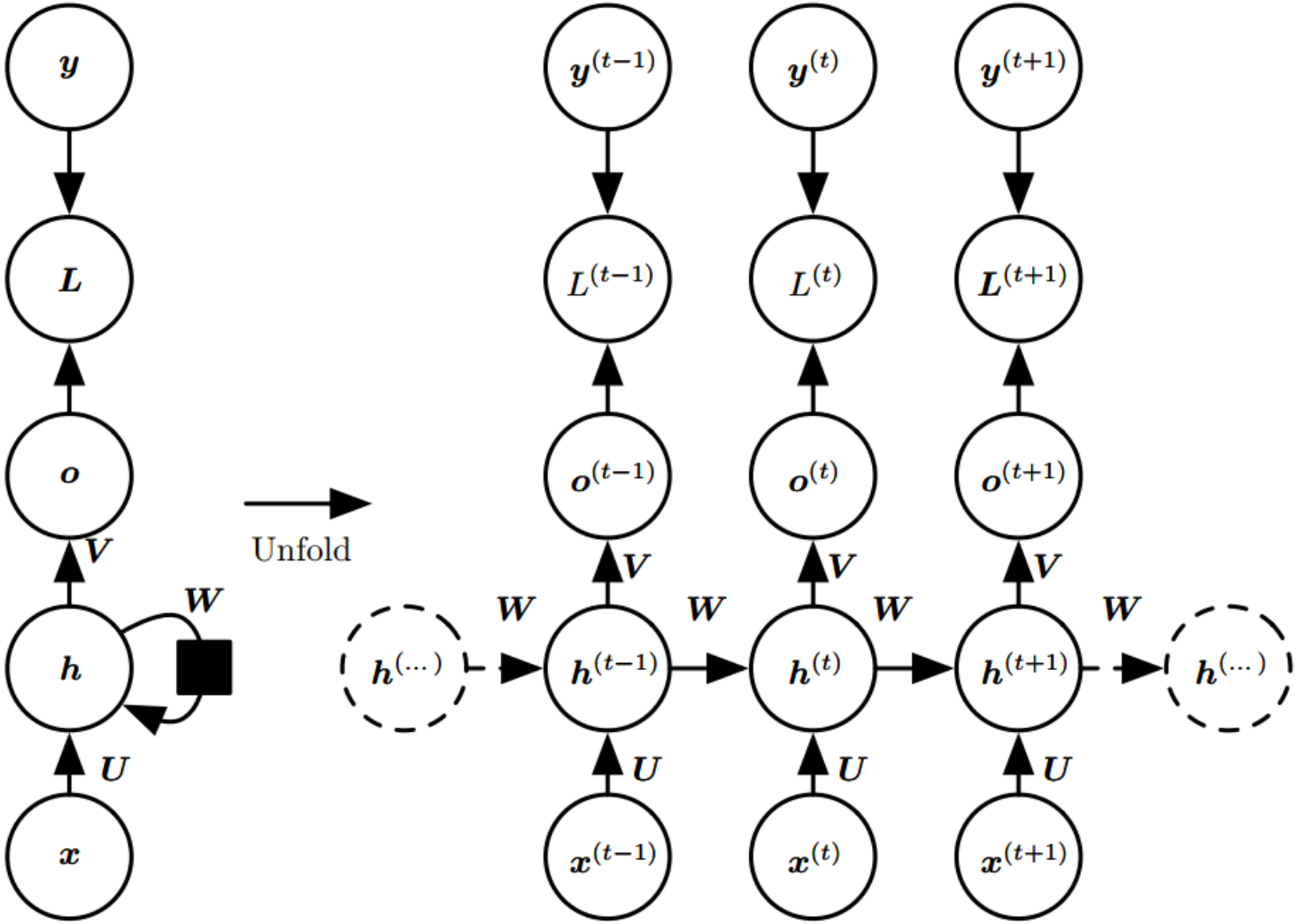
深度序列模型

- 场景
  - 机器翻译
  - 股票预测
  - 空气质量预测

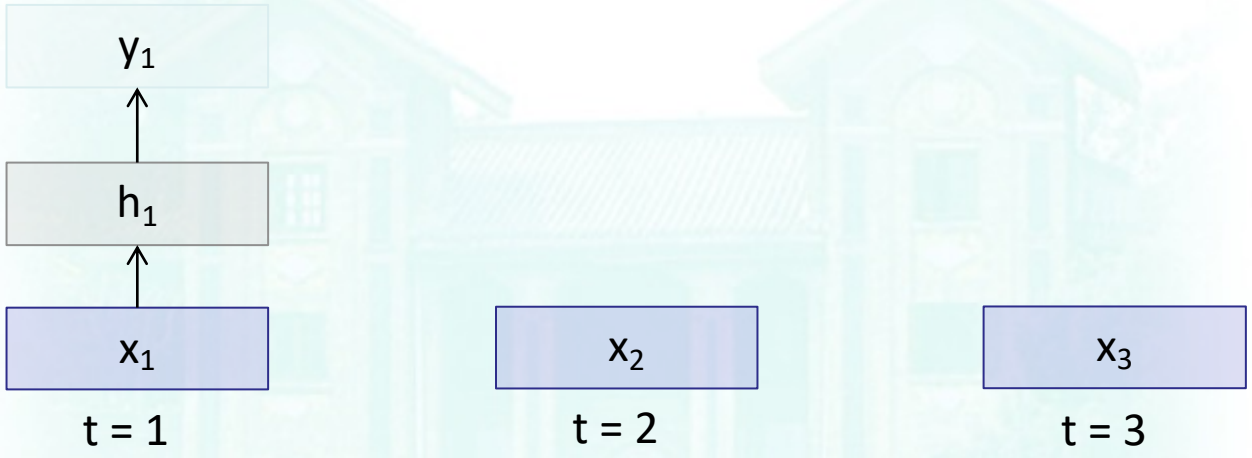


第二部分：循环神经网络

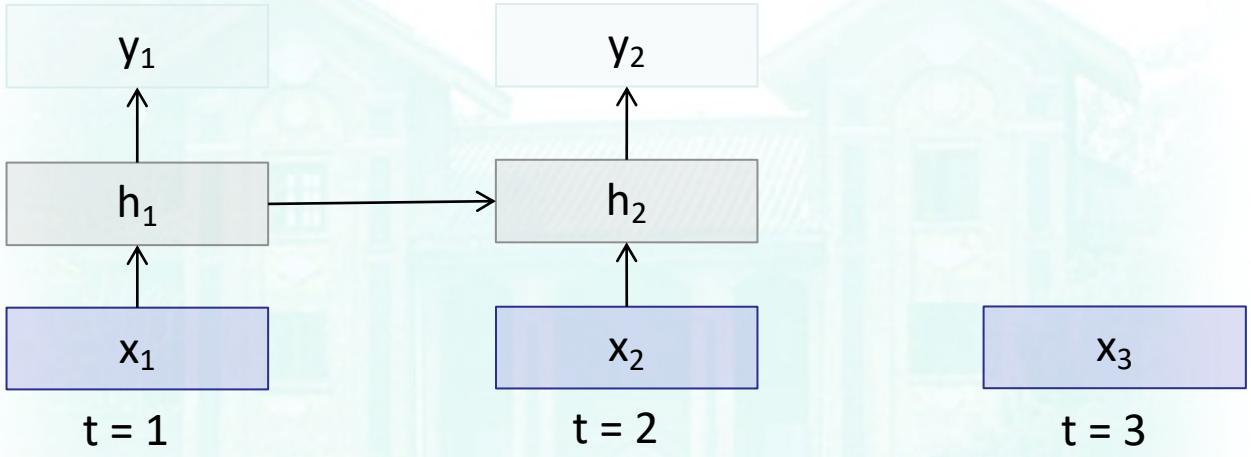
深度序列模型——循环神经网络



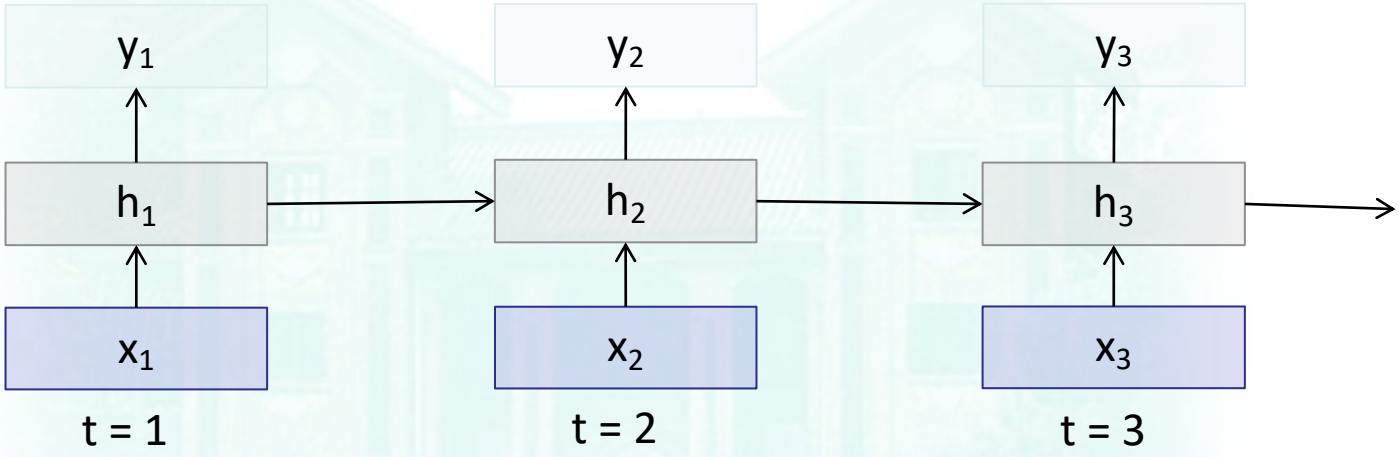
深度序列模型——循环神经网络



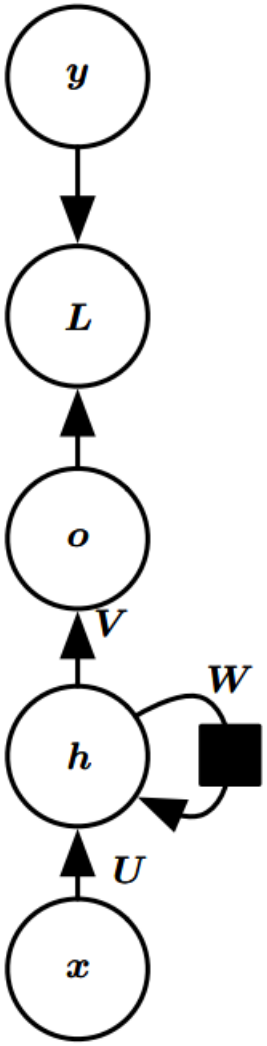
深度序列模型——循环神经网络



深度序列模型——循环神经网络



深度序列模型——循环神经网络

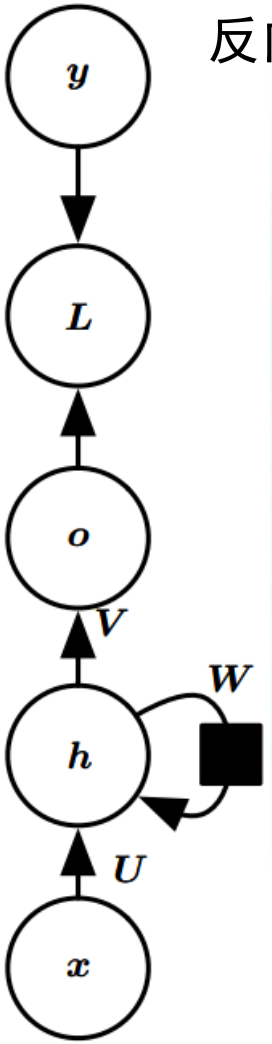


前向传播

$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)},$$
$$\boldsymbol{h}^{(t)} = \tanh(\boldsymbol{a}^{(t)}),$$
$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)},$$
$$\hat{\boldsymbol{y}}^{(t)} = \mathrm{softmax}(\boldsymbol{o}^{(t)}),$$

$$tanh(a) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}}$$

深度序列模型——循环神经网络



反向传播

x: 输入，*h*: 隐藏状态，*o*：输出结果，*y*：目标输出。*L*为输出结果与目标输出计算出损失。在这里简单假定只使用最后时刻的*o*

需要更新的参数为 *U, V, W, b, c*

中间计算媒介

$$(\nabla_{o^{(t)}} L)_i = \frac{\partial L}{\partial o_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}}$$

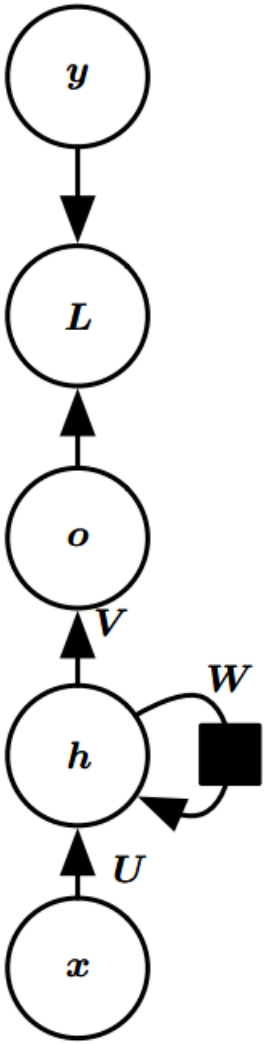
$$\frac{\partial \mathbf{h}^t}{\partial \mathbf{a}^t} = diag(1 - \mathbf{h}^{t\,2})$$

$$\frac{\partial h^{(t+1)}}{\partial h^{(t)}} = \frac{\partial h^{(t+1)}}{\partial a^{(t)}} \frac{\partial a^{(t)}}{\partial h^{(t)}} = \mathbf{W}^T diag(1 - (h^{(t+1)})^2)$$

$$\nabla_{h^{(t)}} L = \left(\frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{h}^{(t)}}\right)^\top (\nabla_{h^{(t+1)}} L) + \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h}^{(t)}}\right)^\top (\nabla_{o^{(t)}} L)$$

$$= \boldsymbol{W}^\top (\nabla_{h^{(t+1)}} L) \mathrm{diag}\left(1 - (\boldsymbol{h}^{(t+1)})^2\right) + \boldsymbol{V}^\top (\nabla_{o^{(t)}} L),$$

深度序列模型——循环神经网络



$$\nabla_c L = \sum_t \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{c}}\right)^\top \nabla_{\boldsymbol{o}^{(t)}} L = \sum_t \nabla_{\boldsymbol{o}^{(t)}} L,$$

$$\nabla_b L = \sum_t \left(\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{b}^{(t)}}\right)^\top \nabla_{\boldsymbol{h}^{(t)}} L = \sum_t \mathrm{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \nabla_{\boldsymbol{h}^{(t)}} L,$$

$$\nabla_V L = \sum_t \sum_i \left(\frac{\partial L}{\partial o_i^{(t)}}\right) \nabla_V o_i^{(t)} = \sum_t (\nabla_{\boldsymbol{o}^{(t)}} L) \boldsymbol{h}^{(t)\top},$$

点积展开

$$\nabla_W L = \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{W^{(t)}} h_i^{(t)}$$

$$= \sum_t \mathrm{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) (\nabla_{\boldsymbol{h}^{(t)}} L) \boldsymbol{h}^{(t-1)\top},$$

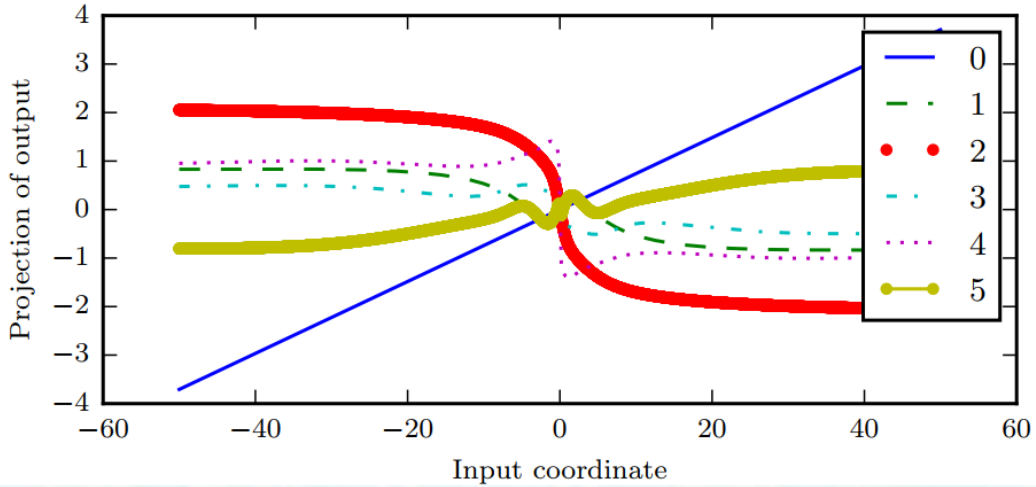
$$\nabla_U L = \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{U^{(t)}} h_i^{(t)}$$

$$= \sum_t \mathrm{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) (\nabla_{\boldsymbol{h}^{(t)}} L) \boldsymbol{x}^{(t)\top},$$

深度序列模型——循环神经网络

长期依赖，梯度消失和爆炸

循环网络涉及相同函数的多次组合，每个时间步一次。
这些组合可以导致极端非线性行为。



重复组合函数。当组合许多非线性函数(如这里所示的线性 tanh 层)时，结果是高度非线性的，通常大多数值与微小的导数相关联，也有一些具有大导数的值，以及在增加和减小之间的多次交替。此处，我们绘制从 100维隐藏状态降到单个维度的线性投影，绘制于y轴上。x轴是 100 维空间中沿着随机方向的初始状态的坐标。因此，我们可以将该图视为高维函数的线性截面。曲线显示每个时间步之后的函数，或者等价地，转换函数被组合一定次数之后。

深度序列模型——循环神经网络

长期依赖，梯度消失和爆炸

这里简单定性分析，忽略激活函数等非线性操作，隐藏状态的变化可以被变形为：

$$\boldsymbol{h}^{(t)} = (\boldsymbol{W}^t)^\top \boldsymbol{h}^{(0)},$$

再假定 W 满足下述形式的特征分解

$$\boldsymbol{W} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top,$$

其中Q正交

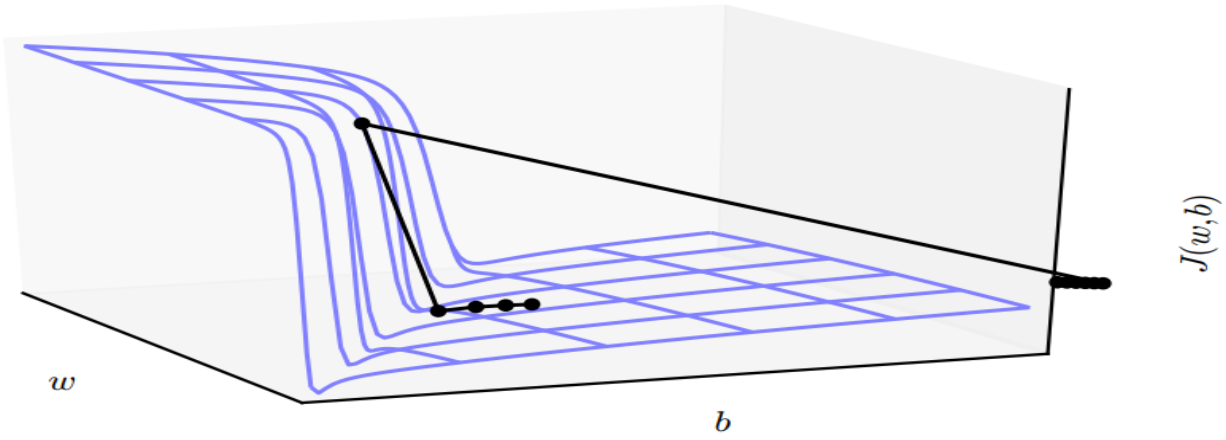
深度序列模型——循环神经网络

长期依赖，梯度消失和爆炸

循环性可以进一步变化为

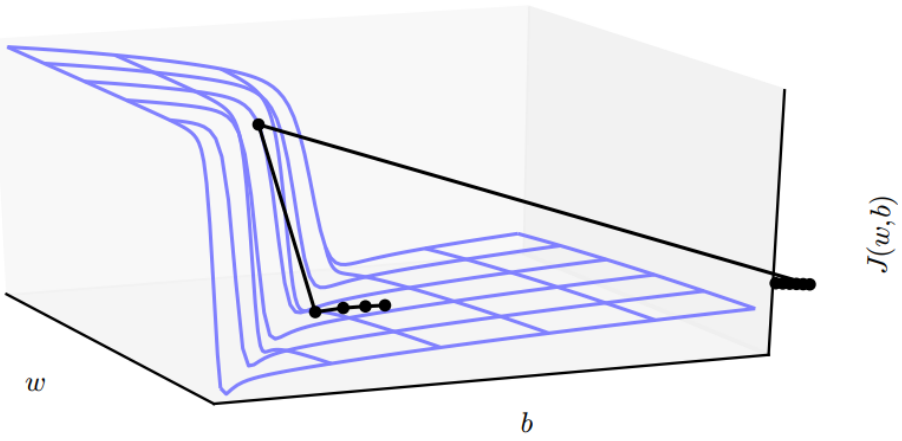
$$\boldsymbol{h}^{(t)} = \boldsymbol{Q}^{\top} \boldsymbol{\Lambda}^{t} \boldsymbol{Q} \boldsymbol{h}^{(0)}.$$

t 次计算后，导致不到 1 的特征值衰减到零，而大于 1 的就会激增（梯度爆炸）。任何不与最大特征向量对齐的 h(0) 的部分将最终被丢弃（梯度消失）。



深度序列模型——循环神经网络

长期依赖，梯度消失和爆炸



在计算机中，由于物理限制，梯度爆炸可能会导致超出数据表示范围而数值变为 NAN。

第三部分：循环神经网络变体及改进

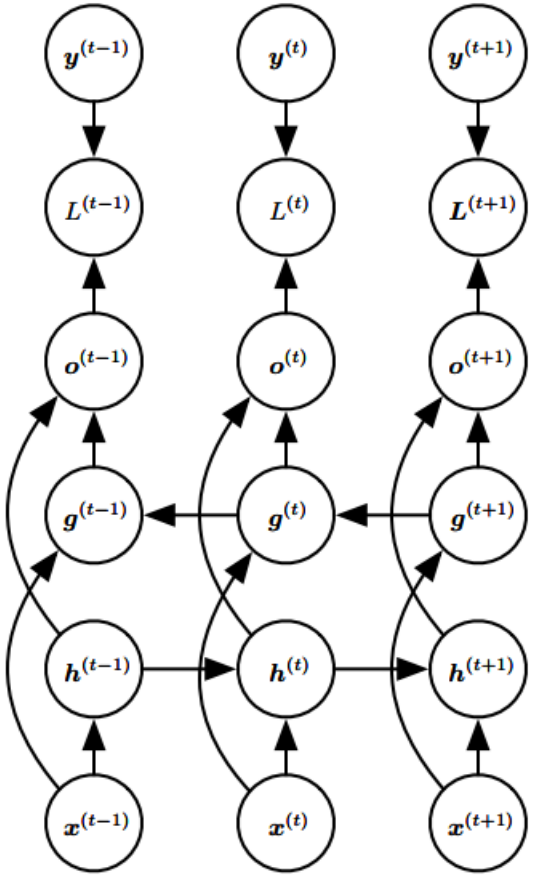
深度序列模型——循环神经网络改进及变体

改进：双向循环神经网络

前述简单RNN只有一种信息流通方式
从前向后。

然而很多场景中，输出可能依赖于
全局。

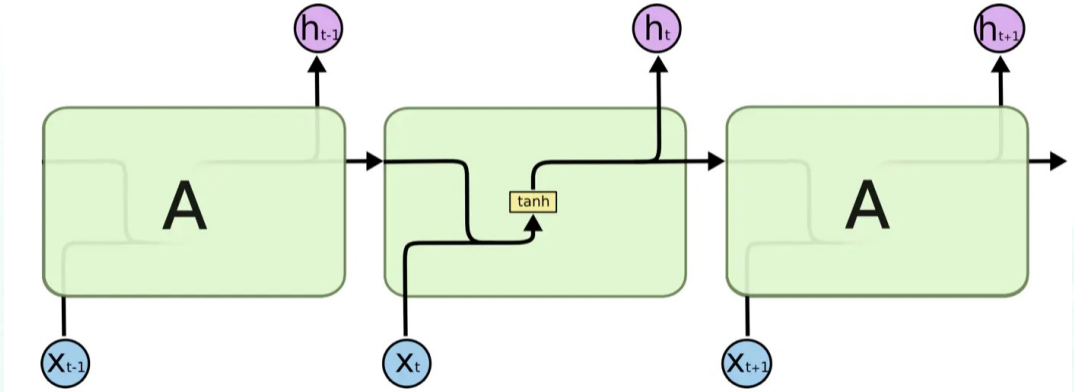
一个简单方案是双向循环神经网络。



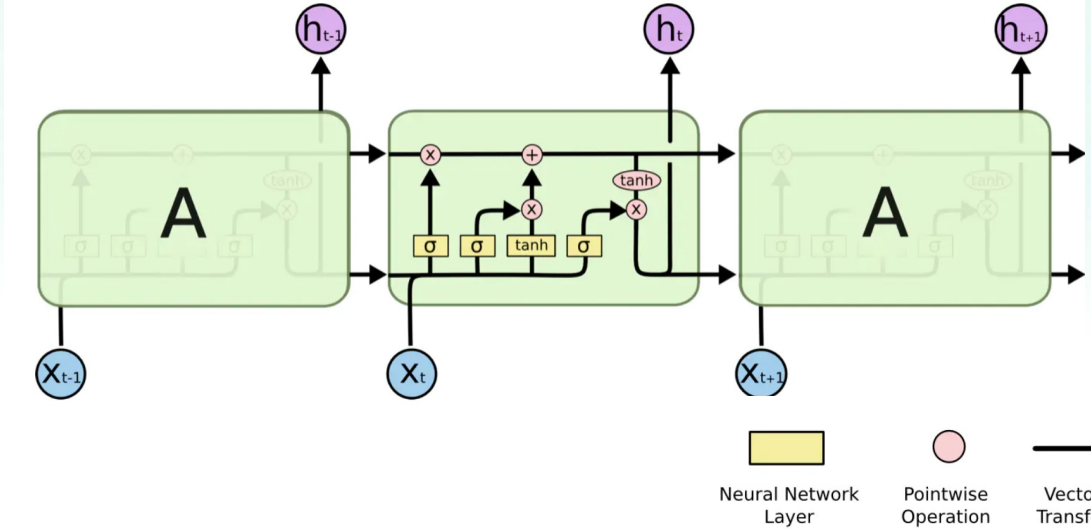
深度序列模型——循环神经网络改进及变体

改进：长短时记忆神经网络(LSTM)——较长的短时记忆神经网络

RNN



LSTM



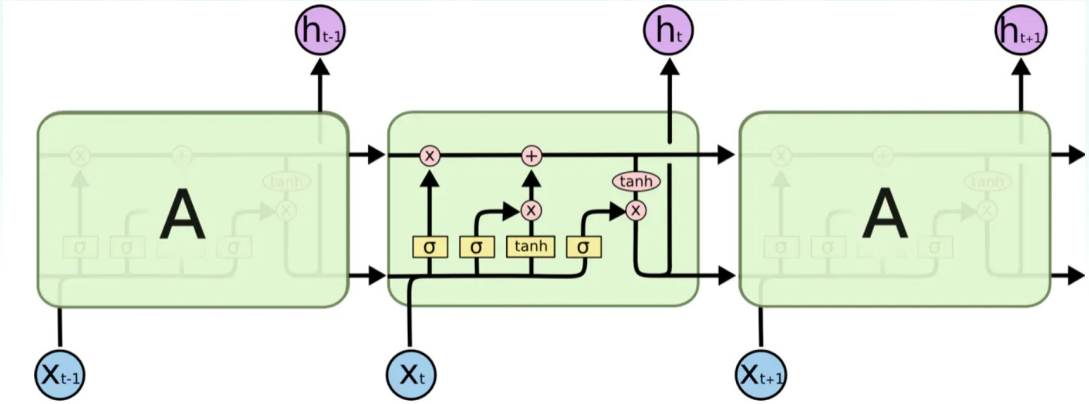
深度序列模型——循环神经网络改进及变体

改进：长短时记忆神经网络(LSTM)——较长的短时记忆神经网络

普通的RNN那样只有一种记忆叠加方式。

LSTM通过门控状态来根据输入内容动态地控制传输状态，
记住需要长时间记忆的重要信息，忘记不重要的信息；

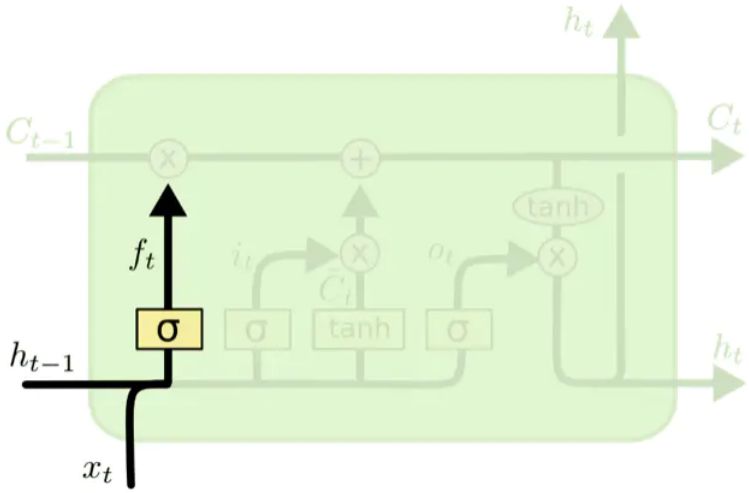
适合需要“长期记忆”的任务。



深度序列模型——循环神经网络改进及变体

改进：长短时记忆神经网络(LSTM)——较长的短时记忆神经网络

遗忘门

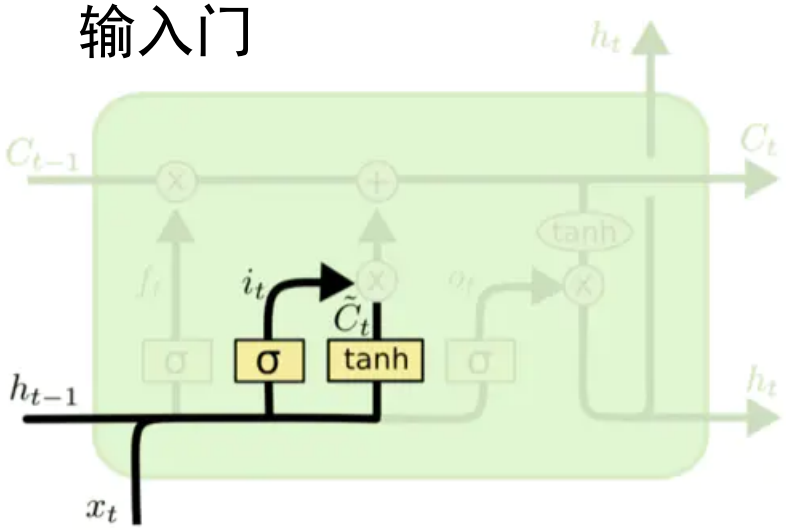


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

深度序列模型——循环神经网络改进及变体

改进：长短时记忆神经网络(LSTM)——较长的短时记忆神经网络

输入门



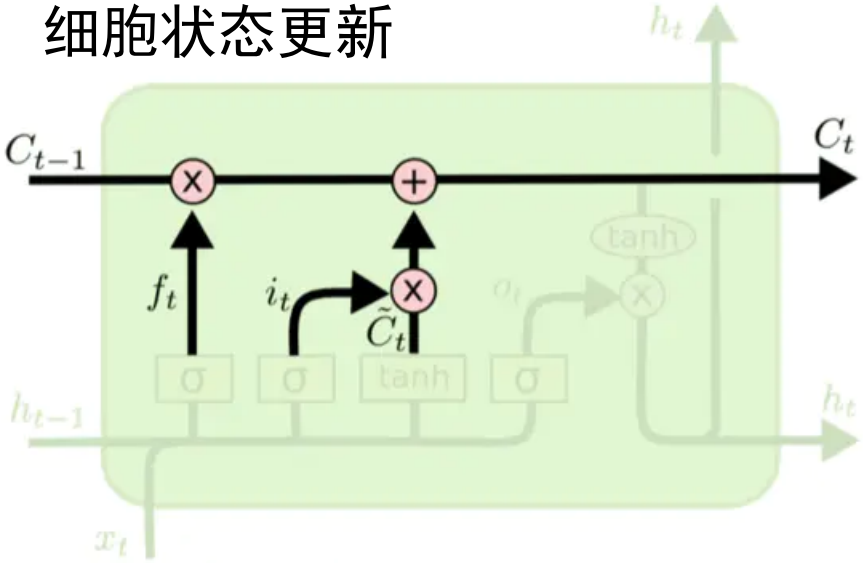
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

深度序列模型——循环神经网络改进及变体

改进：长短时记忆神经网络(LSTM)——较长的短时记忆神经网络

细胞状态更新

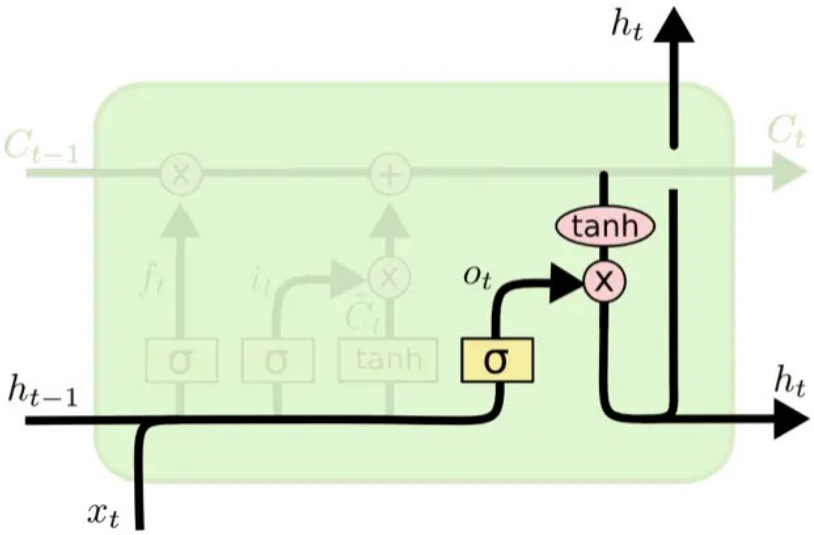


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

深度序列模型——循环神经网络改进及变体

改进：长短时记忆神经网络(LSTM)——较长的短时记忆神经网络

输出门



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

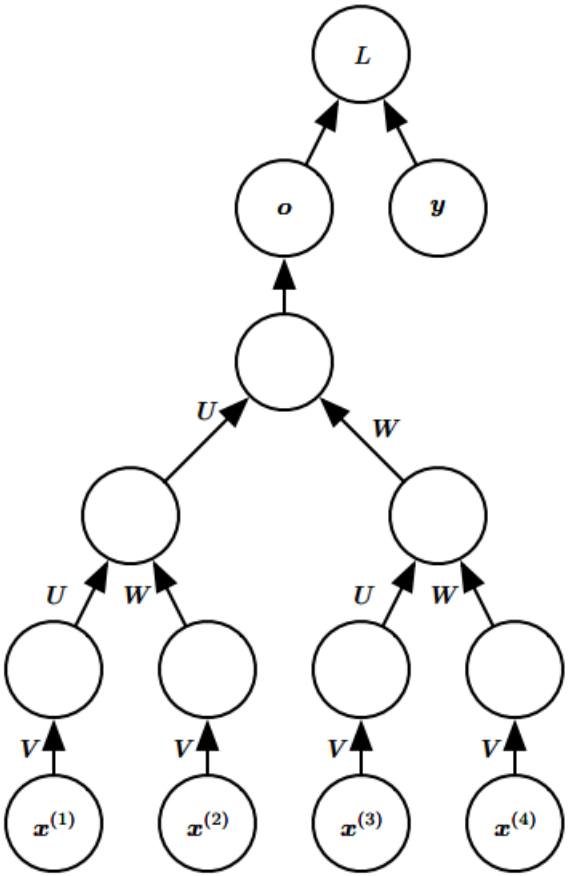
深度序列模型——循环神经网络改进及变体

改进：长短时记忆神经网络(LSTM)——较长的短时记忆神经网络

LSTM的前向传播和后向传播可前往
http://arunmallya.github.io/writeups/nn/lstm/index.html#/1

深度序列模型——循环神经网络改进及变体

扩展——递归神经网络(recursive neural network)



网络的展开形式被构造为深的树状结构而不是 RNN 的链状结构

优势：具有相同长度 τ 的序列，深度（通过非线性操作的组合数量来衡量）可以急剧地从 τ 减小为 O(log τ)。

问题：树的构造方法

某些应用领域，外部方法可以为选择适当的树结构提供借鉴。

例如，处理自然语言的句子时，用于递归网络的树结构可以被固定为句子语法分析树的结构（可以由自然语言语法分析程序提供）

深度序列模型——循环神经网络改进及变体

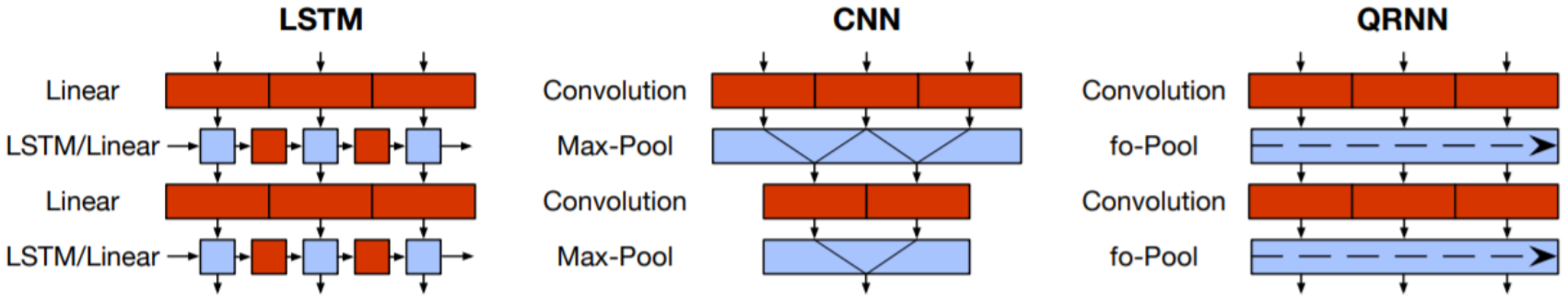
改进：加速

- 循环神经网络作为序列模型，其t时刻的结果依赖于t-1时刻的输出，无法并行地将0-t时刻的输入并行处理，与卷积神经网络的处理速度相比相形见绌，难以满足大规模数据下的训练速度需求。
- 又因为某些场景中不需要过长的信息依赖，如自然语言处理n-gram假设下的语言模型，有些时候卷积神经网络比循环神经网络更适合或权衡之下更适合序列数据。
- 而且，循环神经网络及其部分变种，长距离依赖的处理能力并不强过卷积神经网络许多，更进一步限制了循环神经网络应用场景。
- 速度是循环神经网络的一个痛点。

深度序列模型——循环神经网络改进及变体

改进：加速

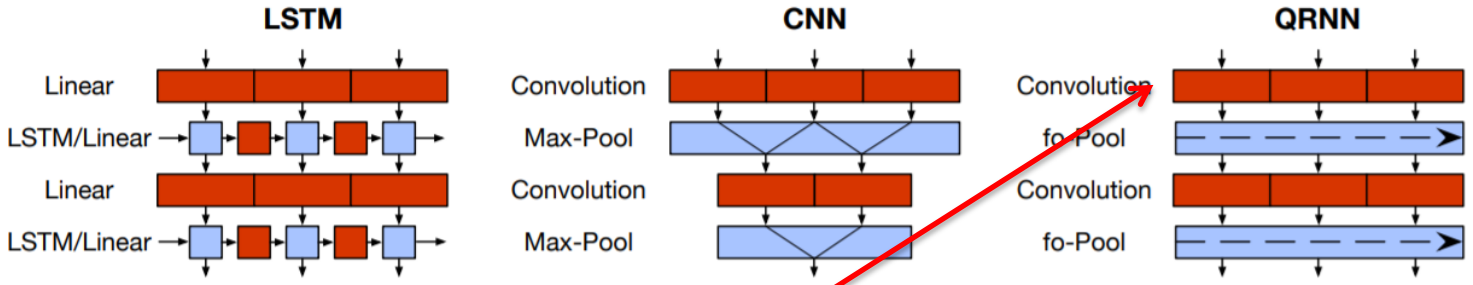
Quasi RNN



深度序列模型——循环神经网络改进及变体

改进：加速

Quasi RNN



$$\mathbf{z}_t = \tanh(\mathbf{W}_z^1 \mathbf{x}_{t-1} + \mathbf{W}_z^2 \mathbf{x}_t)$$
$$\mathbf{f}_t = \sigma(\mathbf{W}_f^1 \mathbf{x}_{t-1} + \mathbf{W}_f^2 \mathbf{x}_t)$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_o^1 \mathbf{x}_{t-1} + \mathbf{W}_o^2 \mathbf{x}_t).$$

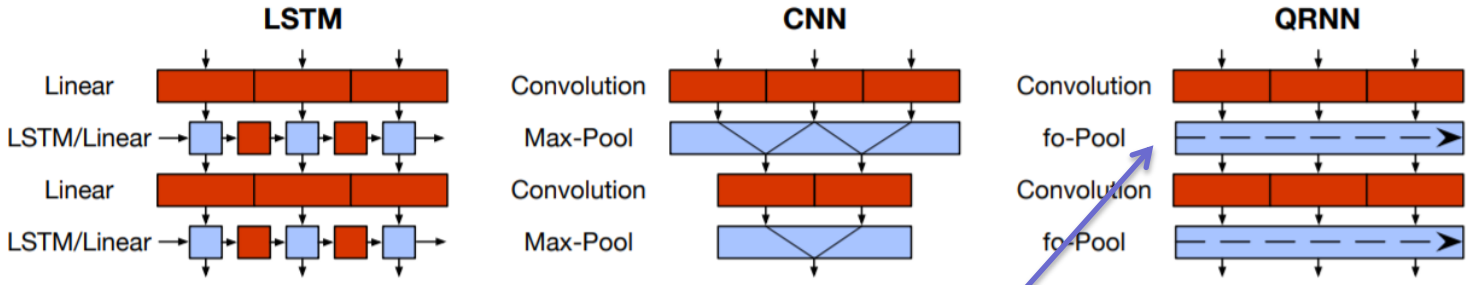
卷积聚合相邻两处输入的信息，可并行化

$$\mathbf{h}_t = \mathbf{f}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{f}_t) \odot \mathbf{z}_t,$$

深度序列模型——循环神经网络改进及变体

改进：加速

Quasi RNN



$$\mathbf{z}_t = \tanh(\mathbf{W}_z^1 \mathbf{x}_{t-1} + \mathbf{W}_z^2 \mathbf{x}_t)$$
$$\mathbf{f}_t = \sigma(\mathbf{W}_f^1 \mathbf{x}_{t-1} + \mathbf{W}_f^2 \mathbf{x}_t)$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_o^1 \mathbf{x}_{t-1} + \mathbf{W}_o^2 \mathbf{x}_t).$$

$$\mathbf{h}_t = \mathbf{f}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{f}_t) \odot \mathbf{z}_t,$$

序列信息依赖，无法并行，
用元素点乘而不是矩阵乘法，降低计算量

第四部分：ATTENTION和RNN

什么是Attention？

- Attention模仿了生物观察行为的内部过程：
- 例如，人在观察图片/文本时，会通过快速扫描全局图像/文本，获得需要重点关注的目标区域。

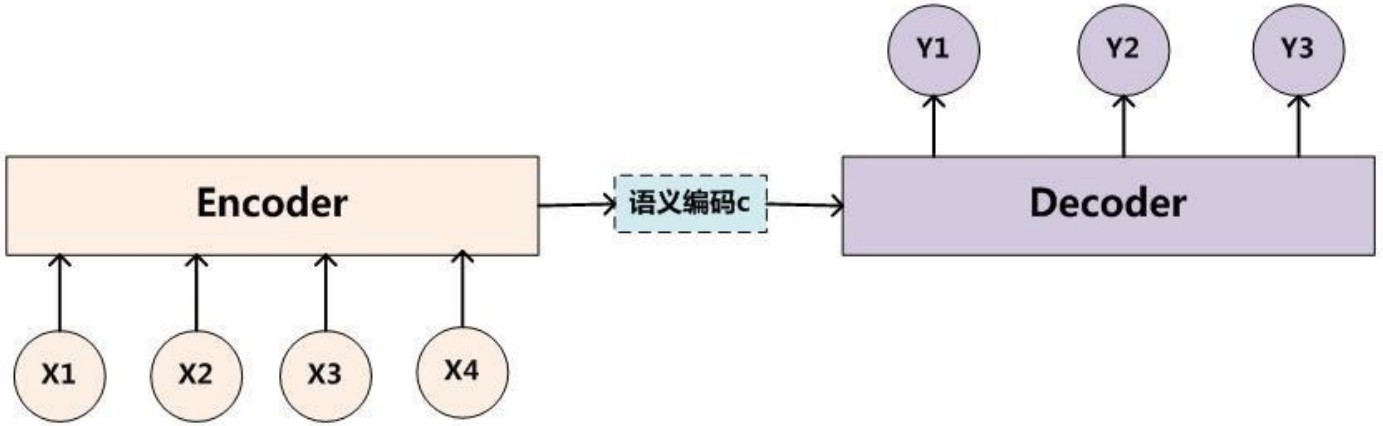
什么是Attention？

- 深度学习中的Attention受生物观察行为的启发，先观察全局计算每个区域的“重要程度”，然后再根据这个“重要程度”对各区域加权求和。

- 在序列模型中，Attention对输入序列的各个元素（词）分别计算一个概率，然后以这个概率为权重，对各元素（词）加权求和，得到Attention的结果。

为什么RNN需要Attention？

- RNN的Attention最初指Encoder-Decoder框架下的Attention：



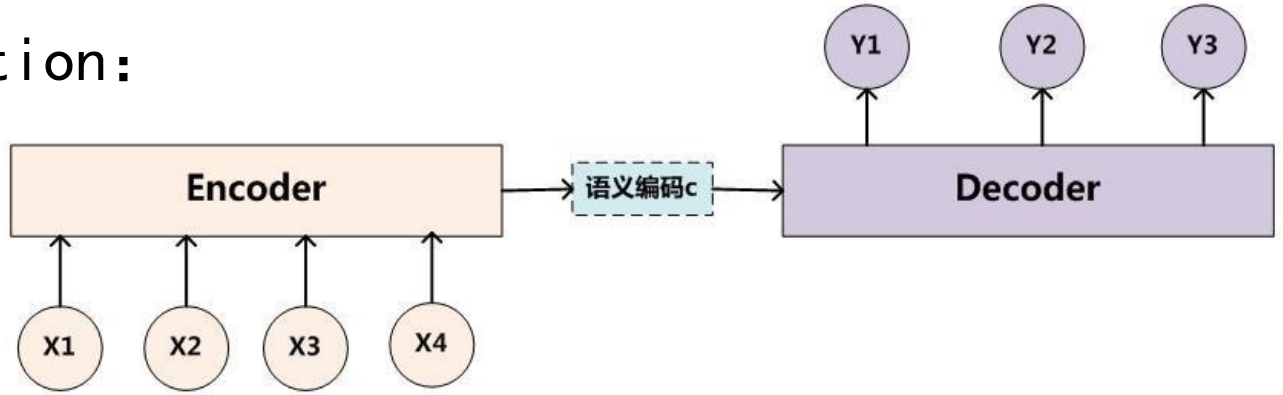
- 第一个角度：Encoder将边长序列压缩成固定维的embedding，造成信息损失；
- 第二个角度：未引入Attention时，对输入序列的每个元素（词）同等对待，不符合人类机制。

为什么RNN需要Attention？

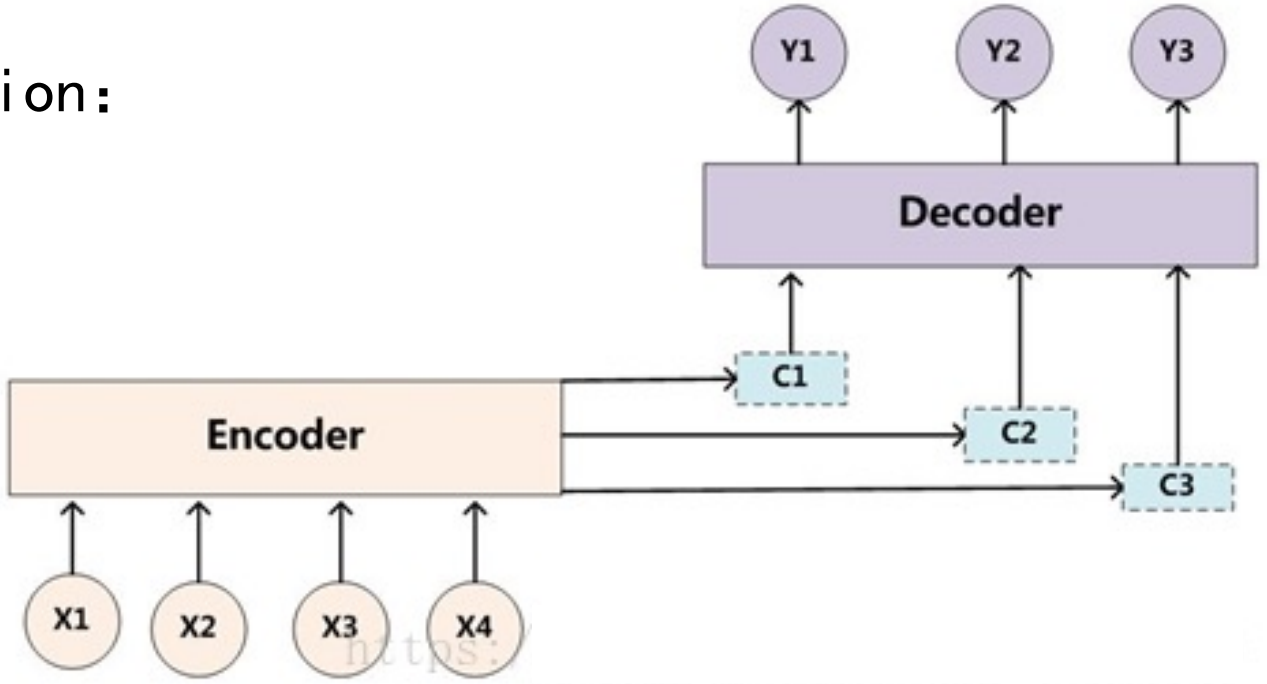
- RNN中Attention的作用：
- 1. 减少长输入序列的计算负担，不必将所有信息都存在一个固定维的embedding中；
- 2. 去伪存真，让模型专注于找到与当前输出相关的有用信息，提高输出的质量；
- 3. 多个元素之间的相互关系往往很难表达，所以很难有针对性地设计系统；而Attention不需要监督信号，对于这种缺少认知先验的问题十分有效。

为什么RNN需要Attention？

无Attention：

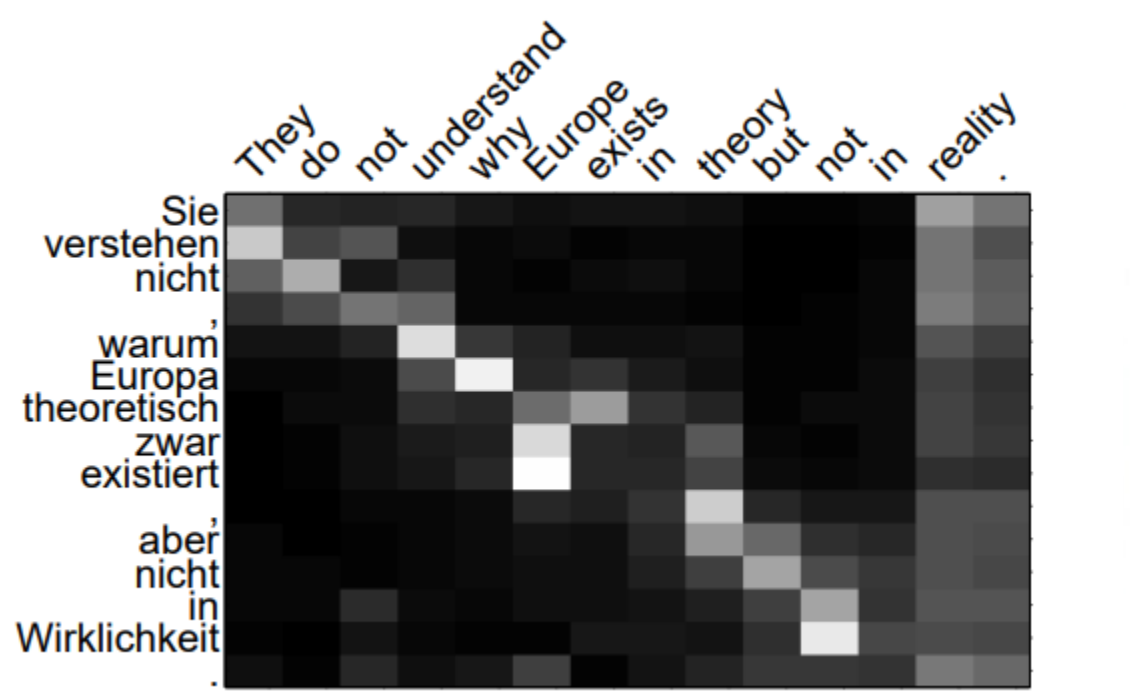


有Attention：



为什么RNN需要Attention？

机器翻译任务，Attention的可视化：



第五部分：RNN中的经典ATTENTION

Bahdanau Attention

- 该Attention由Bahdanau在机器翻译的论文《NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE》中提出。
- Encoder-RNN每时刻的输入是单词的词向量；
- Decoder-RNN每时刻的输出是每个词库里各词的生成概率。

Bahdanau Attention

- Step 1：Decoder-RNN进行到第i位置时，计算第i位置输出和第j位置输入的“相关程度”：

$$e_{ij} = a(s_{i-1}, h_j)$$

其中，a（·）可以是MLP层，和整个模型一起训练。

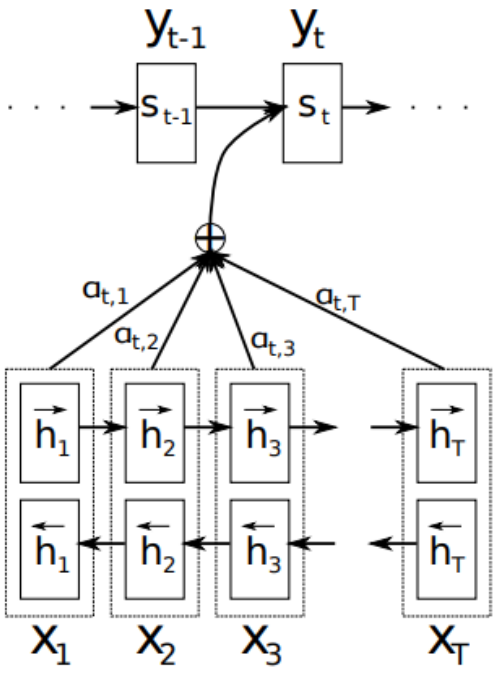


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

Bahdanau Attention

- Step 2：将这个“相关程度”，利用softmax转化成概率：

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})},$$

- Step 3：根据这个概率，对输入序列加权求和：

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j.$$

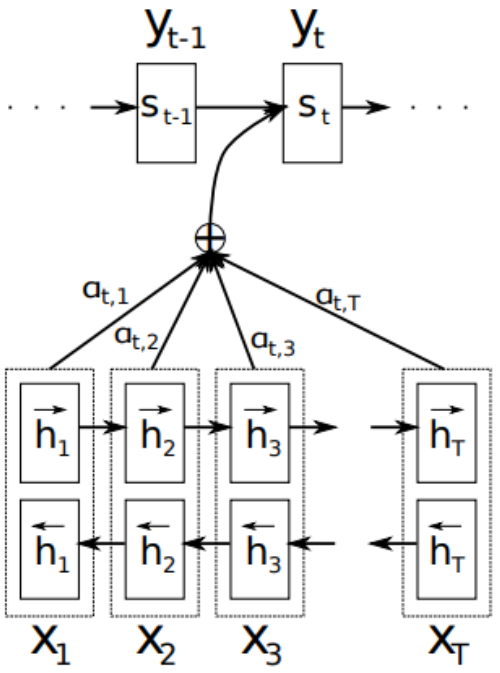


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

Bahdanau Attention

- Step 4：加权求和得到的context vector，输入到Decoder-RNN的隐含状态中：

$$s_i = f(s_{i-1}, y_{i-1}, c_i).$$

- 利用了Attention的结果来影响Decoder-RNN的状态更新。

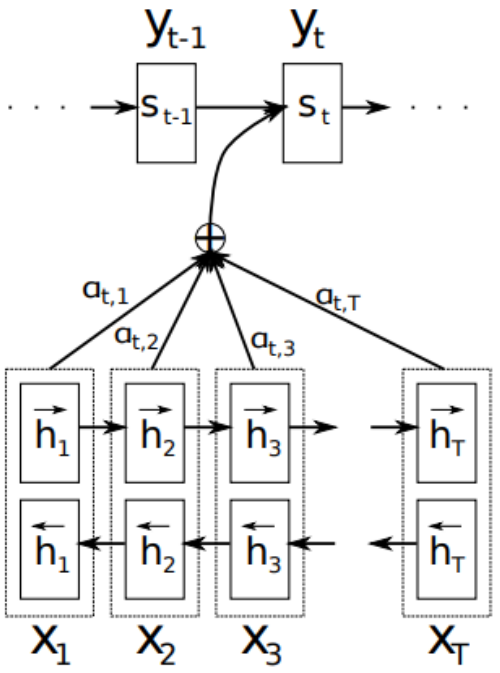


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

Luong Attention

- 该Attention由Luong在论文《Effective Approaches to Attention-based Neural Machine Translation》中提出。

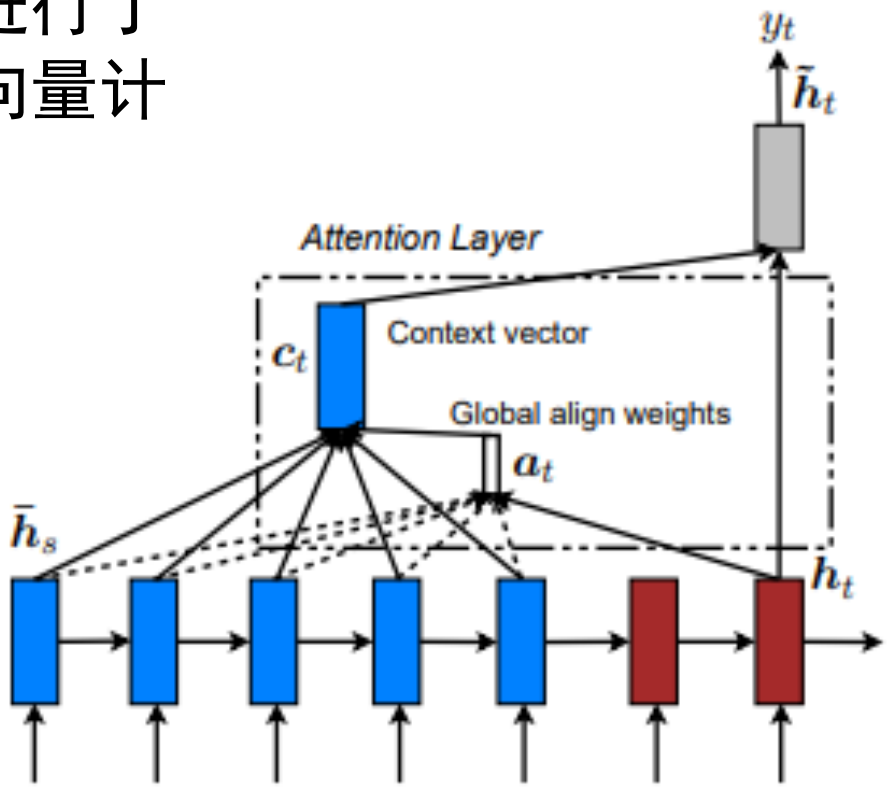
- 该论文的任务与Bahdanau Attention的任务一样，都是机器翻译，但也可用于其他序列任务。

Luong Attention

- Luong Attention与Bahdanau Attention整体类似。

- Luong Attention对原结构进行了一些调整，其中Attention向量计算方法：

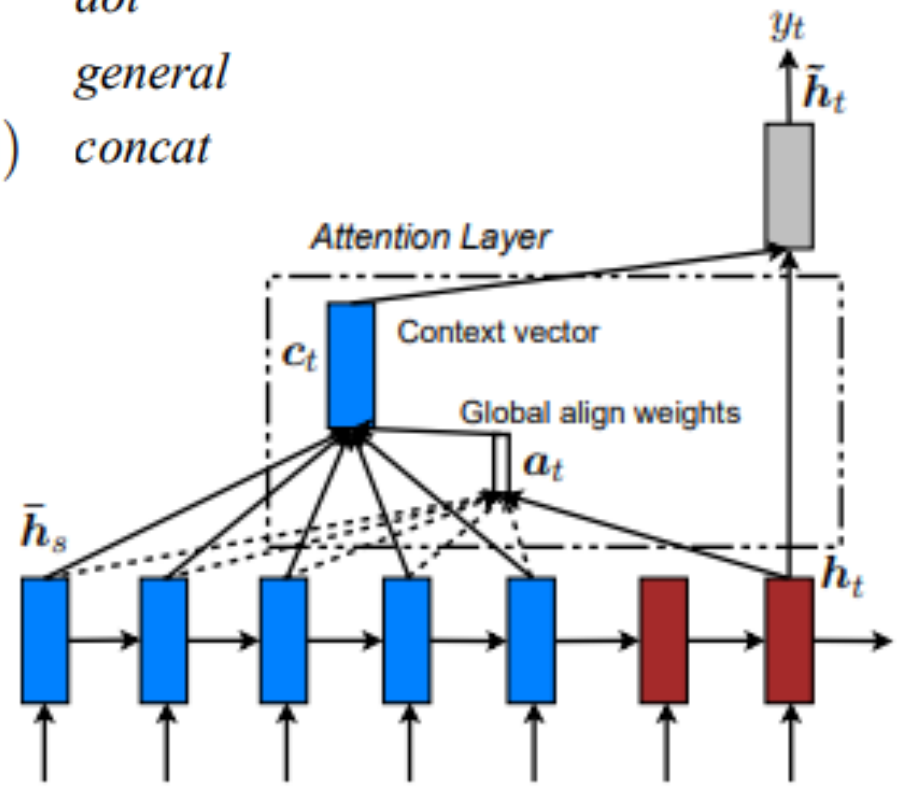
$$\boldsymbol{a}_t(s) = \text{align}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s)$$
$$= \frac{\exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s)\right)}{\sum_{s'} \exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_{s'})\right)}$$



Luong Attention

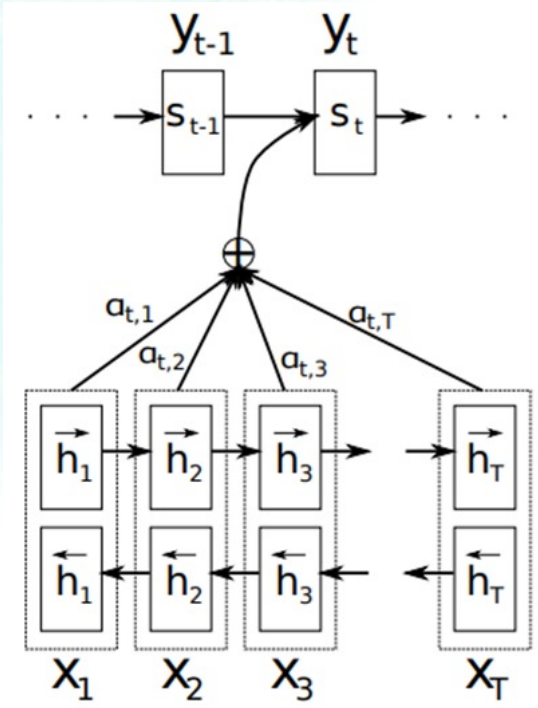
- Luong Attention还尝试了不同的Attention得分计算方式：

$$\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \bar{\boldsymbol{h}}_s & \textit{dot} \\ \boldsymbol{h}_t^\top \boldsymbol{W}_a \bar{\boldsymbol{h}}_s & \textit{general} \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W}_a[\boldsymbol{h}_t; \bar{\boldsymbol{h}}_s]\right) & \textit{concat} \end{cases}$$



Bahdanau Attention 和 Luong Attention

- Bahdanau Attention使用 s_{t-1} 和 h_j 计算score
  Attention的结果用于更新Decoder-RNN;
- Luong Attention使用s_t和h_j计算score;
  Attention的结果不用于更新Decoder-RNN,
  而是直接输出。
- 二者性能上的区别不明显。



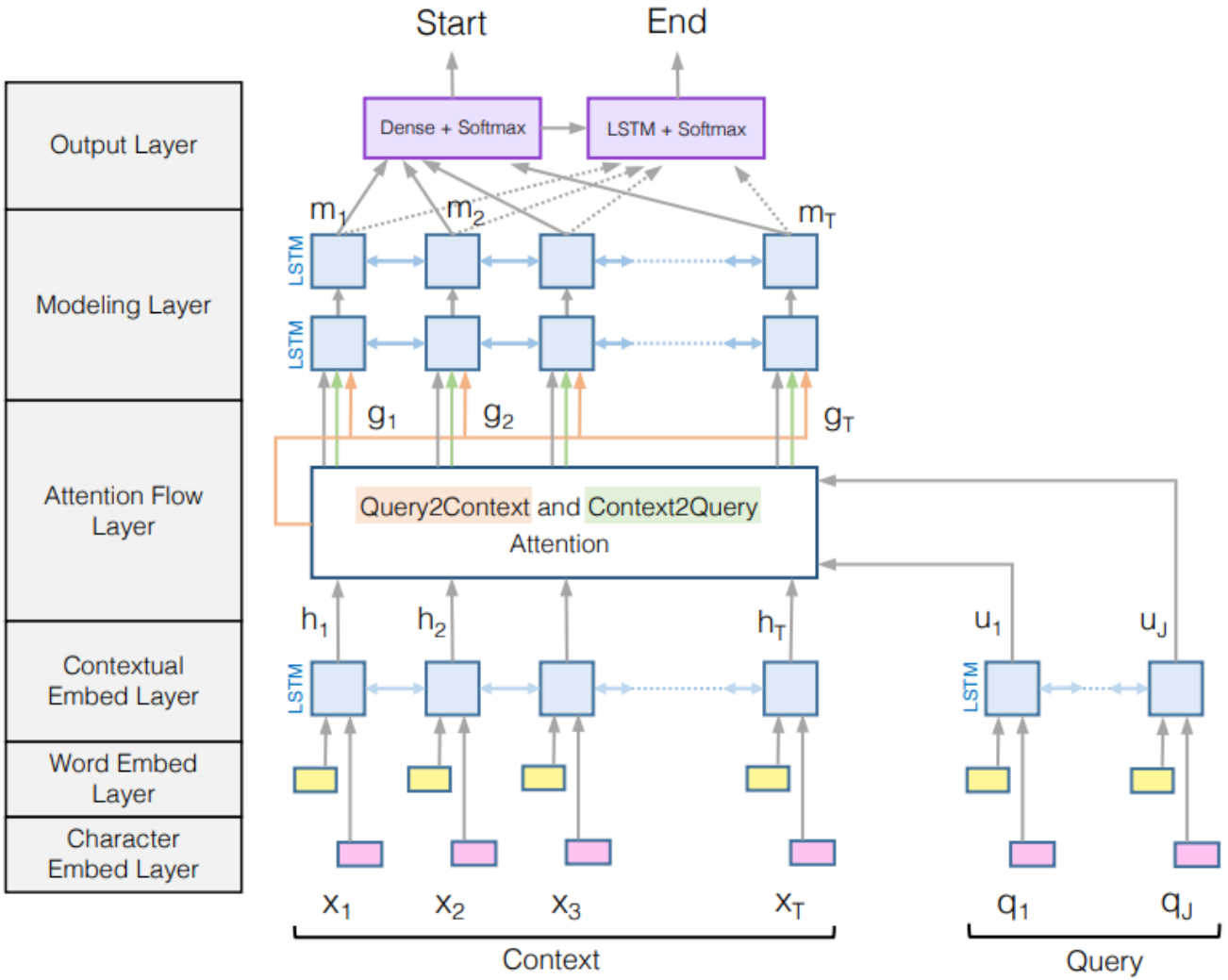
第六部分：BI-ATTENTION

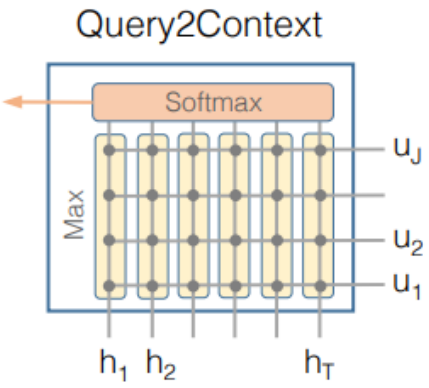
Bi-Attention简介

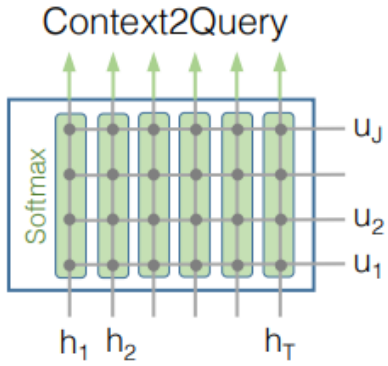
- 前面介绍的Attention主要用于Encoder-Decoder框架的生成任务中；
- 对于两个输入序列的任务（例如，阅读理解任务，输入[问题]和[段落]），有无更合适的Attention？
- Bi-Attention是对两个输入序列的Attention，并且这两个序列往往是存在差异的。

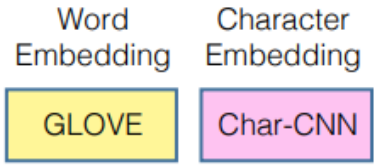
《Bi-Directional Attention Flow For Machine Comprehension》

Bi-Attention简介





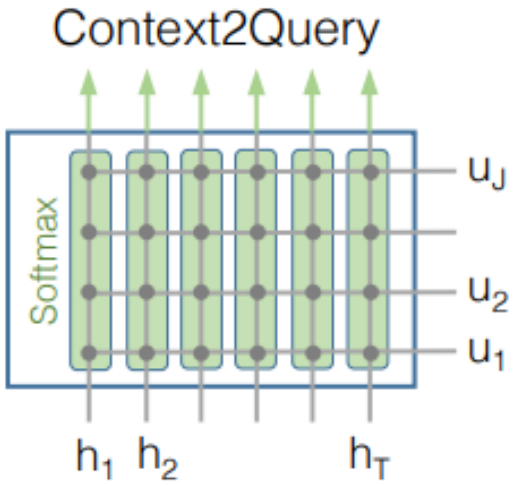




Bi-Attention简介

- Context-to-query Attention:
  对于段落中的每一个词，问题中哪个词与它最接近。

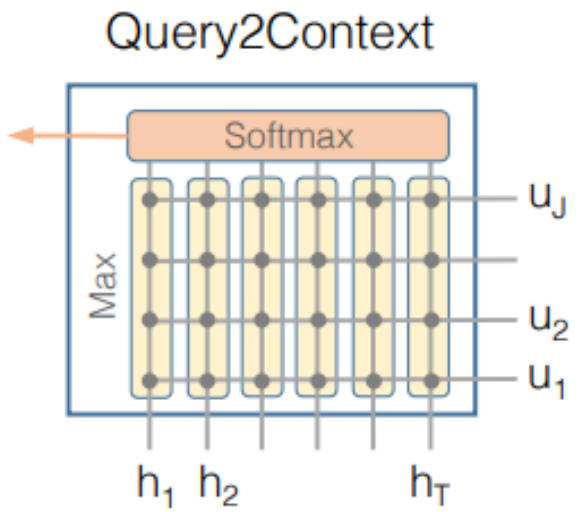
- 将问题中每个词与段落中的一个词计算相似度，经softmax归一化后，计算问题的向量加权和。结果作为对应于段落这个词的问题表示向量。



Bi-Attention简介

- Query-to-context Attention:
  对于问题中的每一个词，段落中的哪一个词和它最相似（最重要）

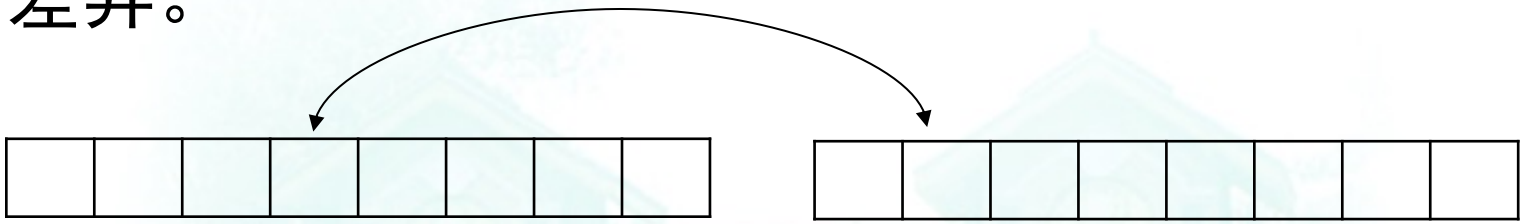
- 点击后，取每列最大值，然后将这些最大值经softmax归一化后，计算段落的向量加权和，将这个向量平铺T次（T为段落长度）。



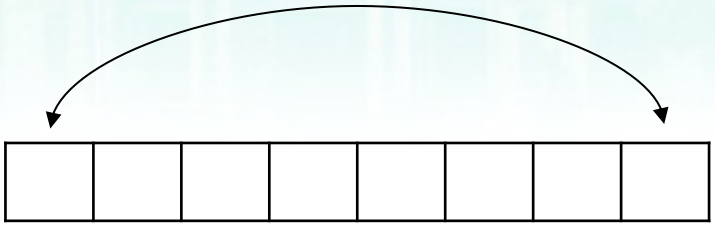
第七部分：SELF-ATTENTION

Self-Attention简介

- 前面介绍的Attention都是在两个序列之间进行的Attention，能够反映序列之间的元素的重要程度差异。

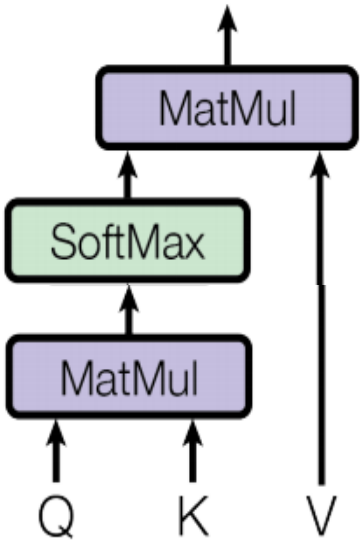


- Self-Attention是对一个序列进行的Attention，能够反映该序列里各元素两两之间的重要程度差异。



Self-Attention简介

- 在Encoder-Decoder框架中，Q(Query)是Decoder-RNN的t时刻隐含状态，K(Key)和V(Value)均是Encoder的各时刻隐含状态。
- 在Self-Attention中，Q、K、V都是输入序列。
- K是需要被翻译的语言经过Encoders之后的词向量表示
- Q是目标语言的所有单词的特征向量对应的矩阵



$$Q = X \cdot W^{Q}$$

$$K = X \cdot W^{K}$$

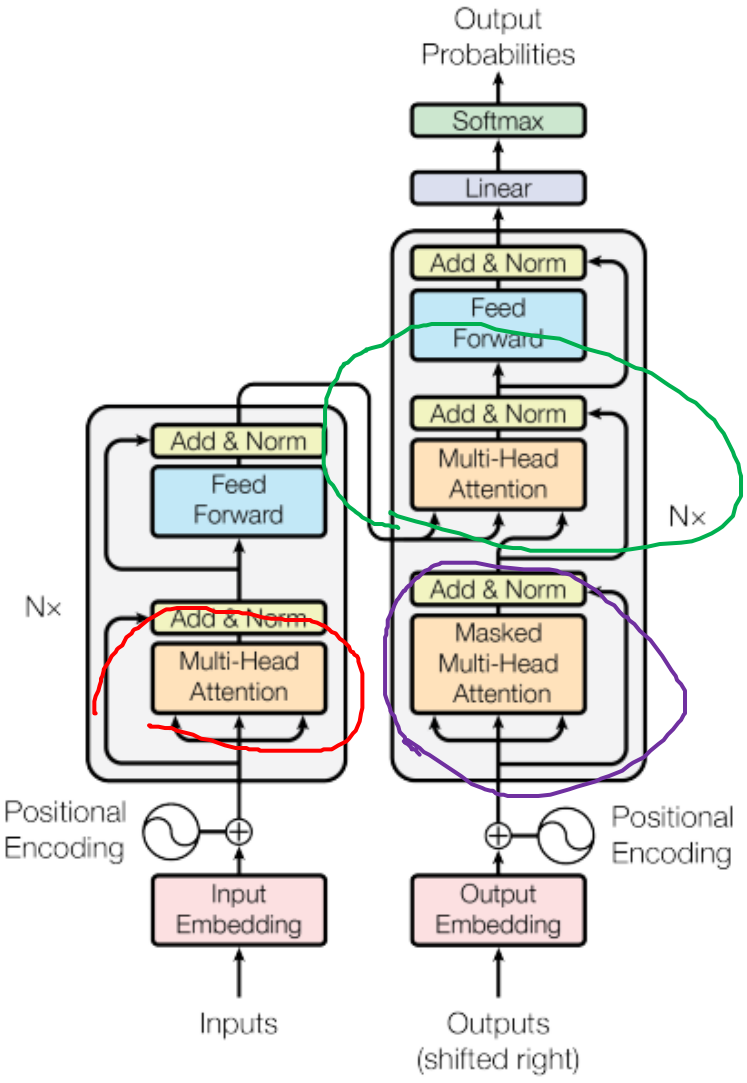
$$V = X \cdot W^{V}$$

Transformer简介

- 基于RNN的Encoder-Decoder模型受限于RNN，无法并行，性能也不佳；
- Google提出Transformer模型：用Self-Attention的结构代替RNN。
- Transformer模型也对Attention本身做了一些改进，提出了Scaled Dot-Product Attention和Multi-Head Attention。

Transformer整体结构

- Transformer由Encoder和Decoder组成。
- Encoder由N个Encoder-Block堆叠而成；Decoder由N个Decoder-Block堆叠而成。
- （堆叠的作用是获取更深层的语义信息）

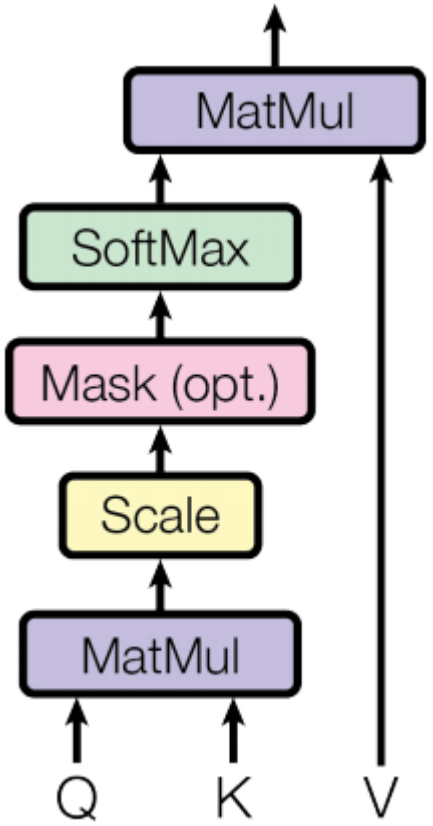


Transformer：Scaled Dot-Product Attention

- 在维度较高的时候，Q和K点积会得到一个很大的数。
- 为了防止其结果过大，在Q和K点积时除以一个scaling factor $\sqrt{d_k}$：

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

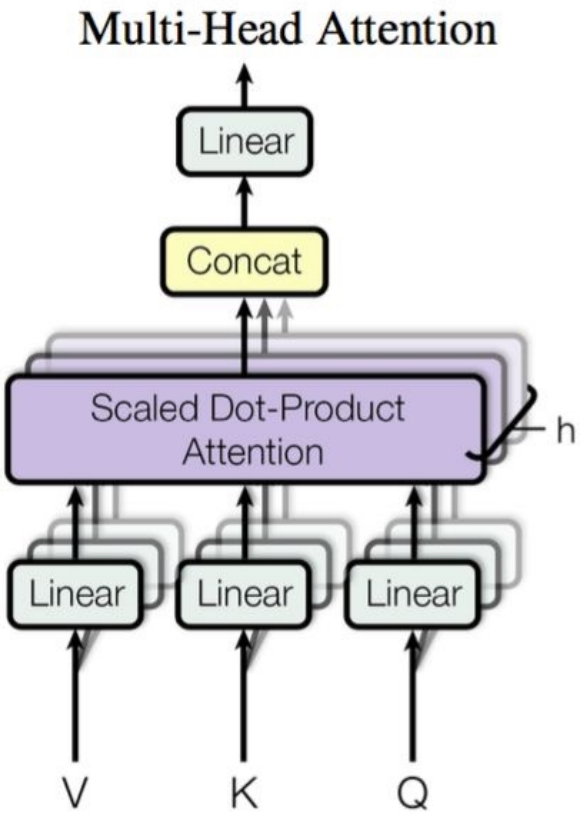
其中，d_k是K的维度。



Transformer：Multi-Head Attention

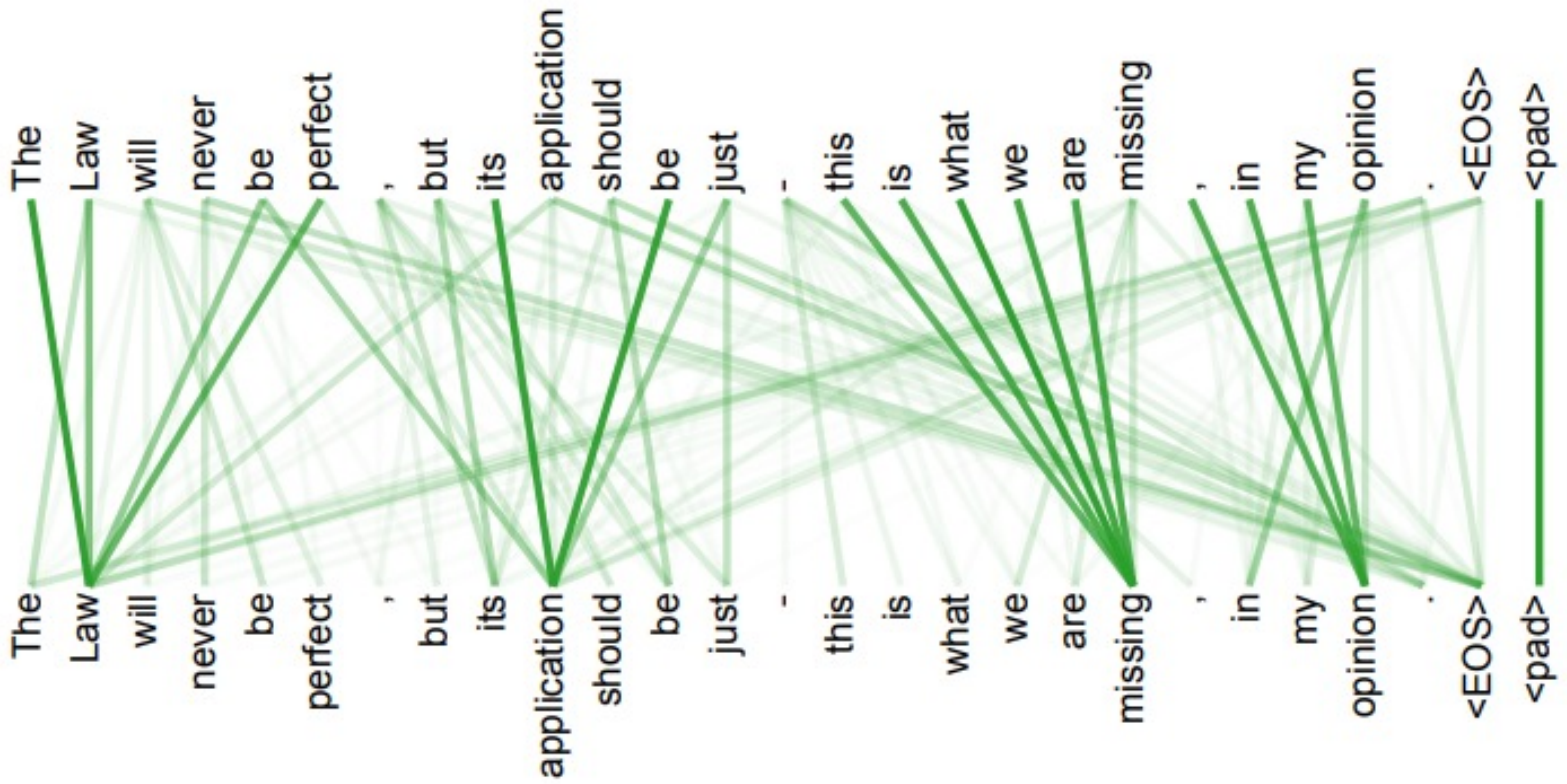
- Multi-Head Attention将Q、K、V映射到h个不同的空间，再在这h个不同的空间里各自做Self-Attention，最后将它们的结果拼接起来：

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1,...,\text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$



Transformer

- 权重可视化：



Transformer

- 权重可视化（仅画出”making”的结果）：

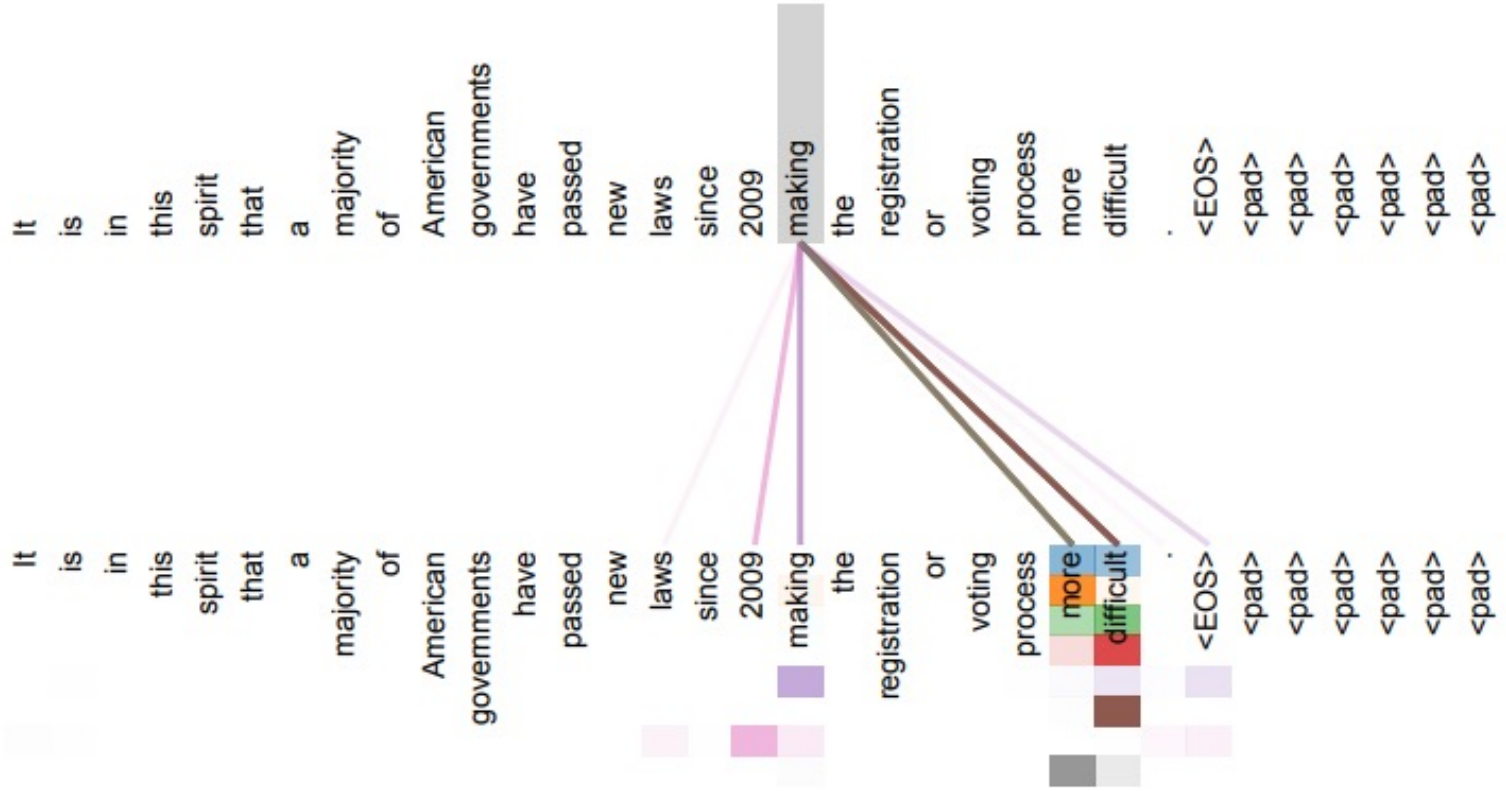
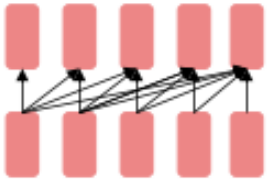


Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb ‘making’, completing the phrase ‘making...more difficult’. Attentions here shown only for the word ‘making’. Different colors represent different heads. Best viewed in color.

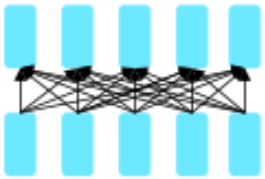
Pretraining based on Transformer

The neural architecture influences the type of pretraining, and natural use cases.



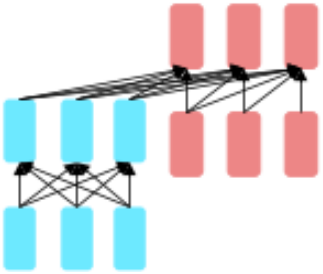
Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words
- **Examples:** GPT-2, GPT-3, LaMDA



Encoders

- Gets bidirectional context – can condition on future!
- Wait, how do we pretrain them?
- **Examples:** BERT and its many variants, e.g. RoBERTa



Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?
- **Examples:** Transformer, T5, Meena

Pretraining Decoders

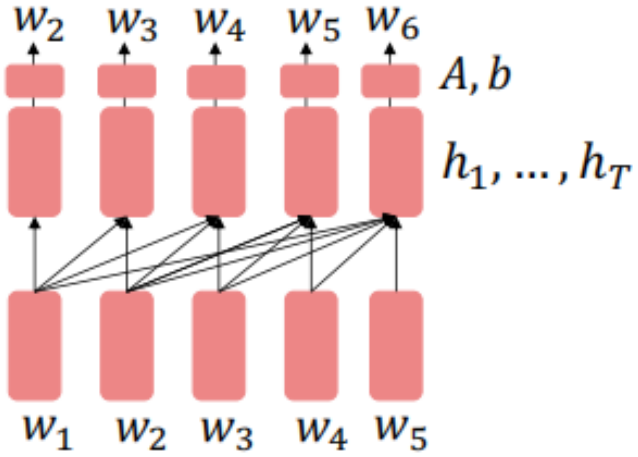
It's natural to pretrain decoders as language models and then use them as generators, finetuning their $p_\theta(w_t|w_{1:t-1})$!

This is helpful in tasks **where the output is a sequence** with a vocabulary like that at pretraining time!

- Dialogue (context=dialogue history)
- Summarization (context=document)

$$h_1, \ldots, h_T = \text{Decoder}(w_1, \ldots, w_T)$$
$$w_t \sim Ah_{t-1} + b$$

Where A, b were pretrained in the language model!



[Note how the linear layer has been pretrained.]

Generative Pretrained Transformer (GPT)

- 2018’ s GPT was a big success in pretraining a decoder!
  - Transformer decoder with 12 layers.
  - 768-dimensional hidden states, 3072-dimensional feed-forward hidden layers.
  - Byte-pair encoding with 40,000 merges
  - Trained on BooksCorpus: over 7000 unique books.
  - Contains long spans of contiguous text, for learning long-distance dependencies.

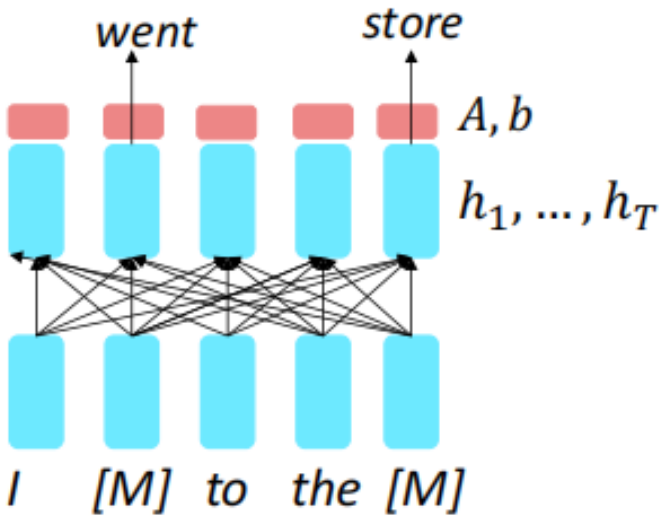
Pretraining encoders

What pretraining objective to use?

So far, we've looked at language model pretraining. But **encoders get bidirectional context,** so we can't do language modeling!

Idea: replace some fraction of words in the input with a special [MASK] token; predict these words.

Only add loss terms from words that are "masked out." If $\tilde{x}$ is the masked version of x, we're learning $p_\theta(x|\tilde{x})$. Called **Masked LM**.



[Devlin et al., 2018]

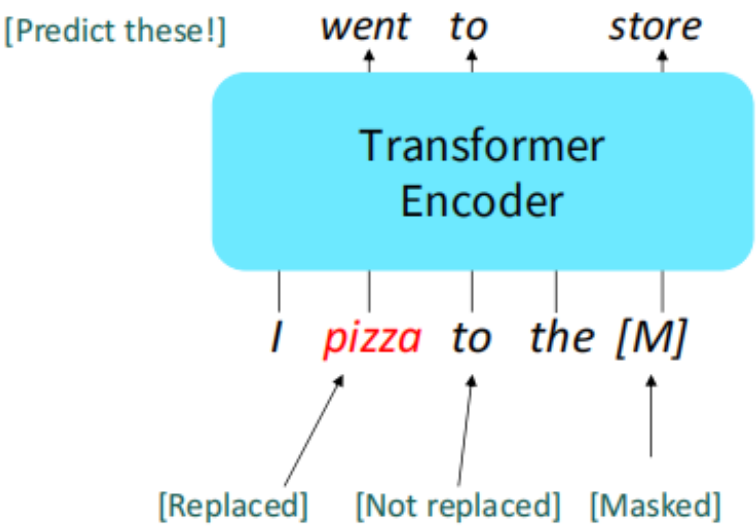
BERT

Bidirectional Encoder Representations from Transformers

Devlin et al., 2018 proposed the "Masked LM" objective, open-sourced their model as the tensor2tensor library, and **released the weights of their pretrained Transformer (BERT).**

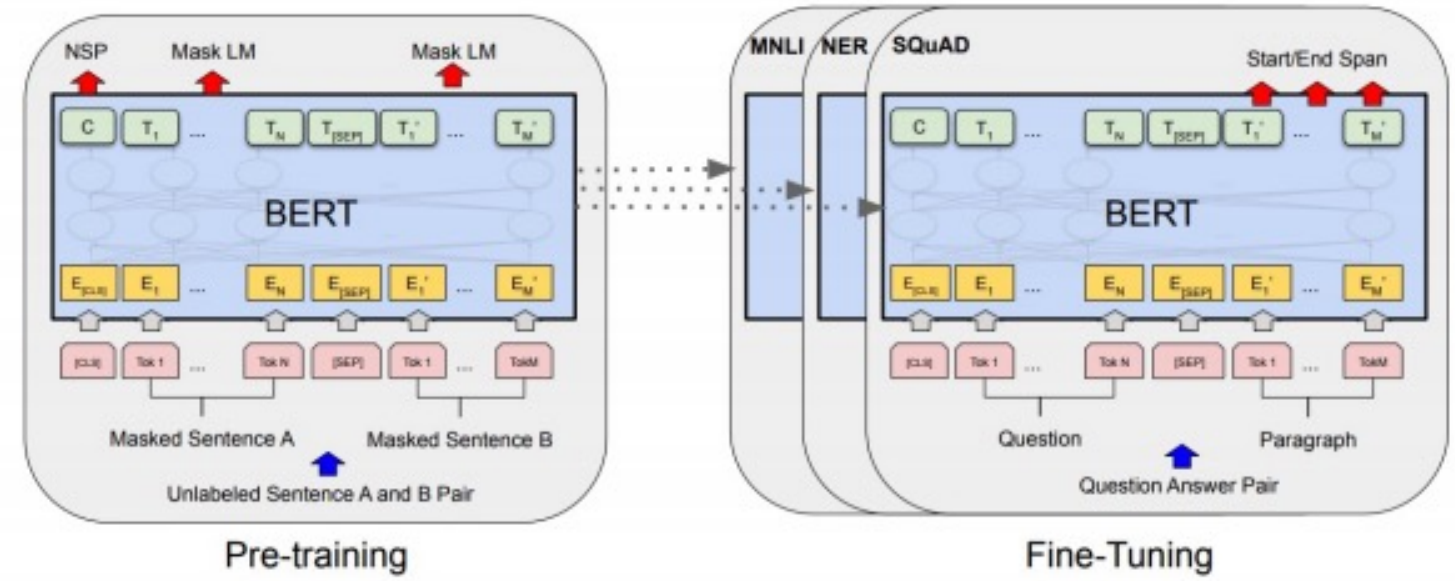
Some more details about Masked LM for BERT:

- Predict a random 15% of (sub)word tokens.
  - Replace input word with [MASK] 80% of the time
  - Replace input word with a random token 10% of the time
  - Leave input word unchanged 10% of the time (but still predict it!)
- Why? Doesn't let the model get complacent and not build strong representations of non-masked words. (No masks are seen at fine-tuning time!)



BERT

- **Unified Architecture:** As shown below, there are minimal differences between the pre-training architecture and the fine-tuned version for each downstream task.



参考书

《Deep learning》（花书）
《Understanding-LSTMs》
《QUASI-RECURRENT NEURAL NETWORKS》
……