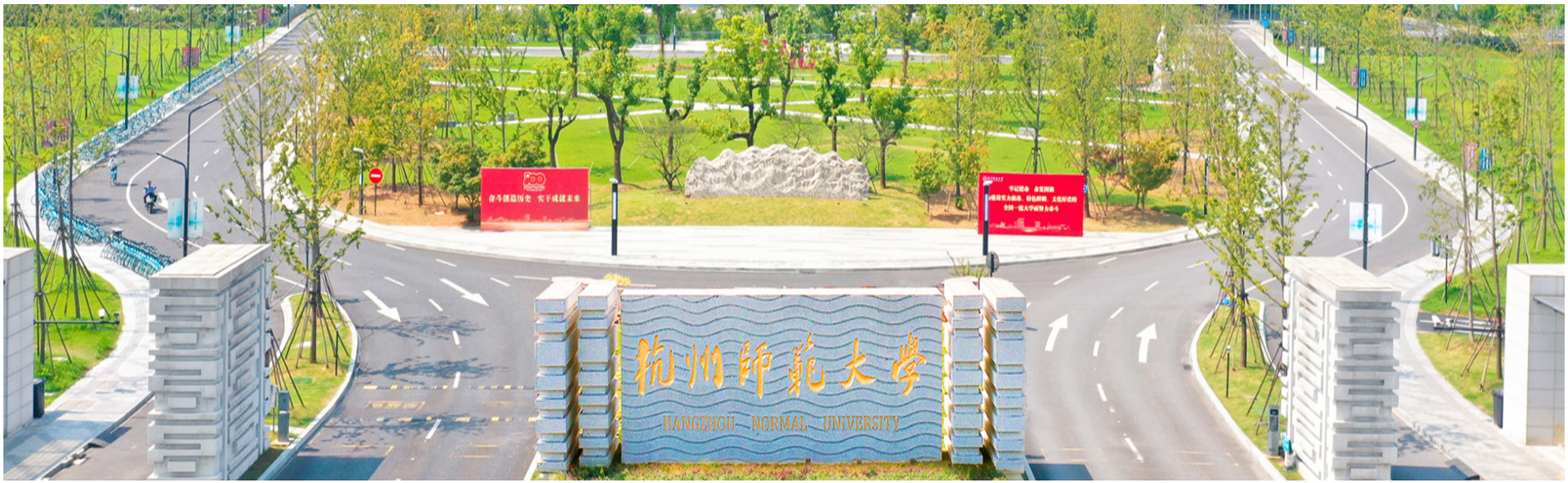
第八讲 - 循环神经网络

张建章

阿里巴巴商学院
杭州师范大学

2025-02-01



目录

1. 语言的时间性

语言天生具有时序性：无论是口语的声音信号，还是书面文本，都以序列的形式随时间展开。

2. 传统模型对时间性的处理及其局限

大多数此前学习的方法（如文本分类任务中的模型）都假设能够同时访问输入的所有部分，缺乏对时间顺序的内在建模。第7讲中介绍的前馈神经网络语言模型，也是通过固定大小的滑动窗口来捕捉上下文；每次只看定长的词序列，然后沿序列滑动，独立地预测下一个词。这种“窗口”方法虽然简单，但**限制了模型对较长距离依赖的感知**。

3. 循环网络（RNN）与长短期记忆（LSTM）

本讲介绍一种替代“滑动窗口”的深度学习架构——循环神经网络（RNN）及其变体（如LSTM）。RNN通过在隐藏层引入循环连接，**将前一时刻的隐状态作为当前输入的一部分**，从而无需人为设定上下文长度，即可**在理论上**让模型**记忆跨越任意距离的序列信息**，并将其应用于语言建模、文本分类乃至序列标注（如词性标注）等多种任务中。

目录

1. 循环连接（Recurrence）

RNN是指网络中**存在环路（cycle）**的神经网络，某个单元的值直接或间接地依赖于其自身先前的输出作为输入。

2. Elman简单循环网络

简单循环网络（Simple RNN, 本讲之后内容中使用RNN特指 Simple RNN）又称Elman Networks，其结构如下图所示：在输入层到隐藏层的前馈连接之外，隐藏层还接收来自上一步隐藏层的输出作为额外输入。

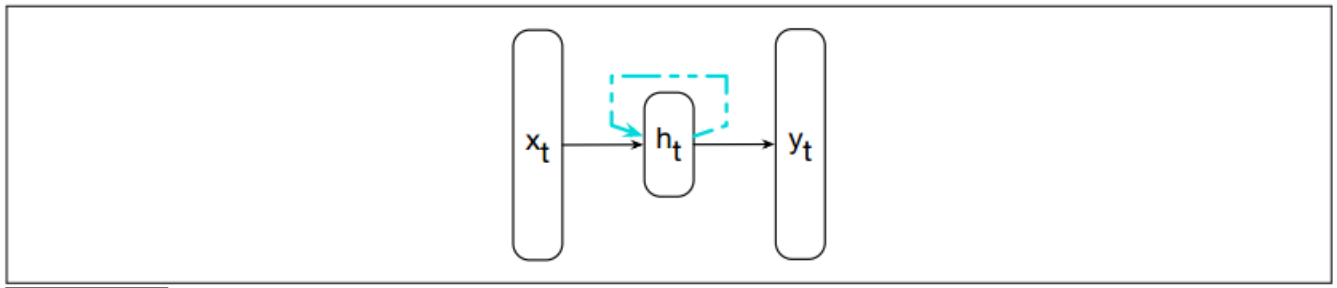


Figure 8.1 Simple recurrent neural network after Elman (1990). The hidden layer includes a recurrent connection as part of its input. That is, the activation value of the hidden layer depends on the current input as well as the activation value of the hidden layer from the previous time step.

3. 隐藏层记忆功能

上一时刻的隐藏层状态 h_{t-1} 相当于对序列先前信息的编码，不受固定窗口长度限制，可**理论上保留**从序列开头到当前的所有上下文。

RNN的前向推理（Inference）

1. 逐步处理

序列按时序逐元素输入：在时刻 t 仅使用当前输入 x_t 与上一步隐藏状态 h_{t-1} 计算当前隐藏状态 h_t，再计算输出 y_t。

2. 计算公式

$$\begin{cases} h_t = g(U h_{t-1} + W x_t) \\ y_t = f(V h_t) \end{cases}$$

其中，$W \in \mathbb{R}^{d_h \times d_{in}}$、$U \in \mathbb{R}^{d_h \times d_h}$、$V \in \mathbb{R}^{d_{out} \times d_h}$。输出通常经softmax归一化：

$$y_t = \text{softmax}(V h_t)$$

3. 时间展开（Unrolling）

将网络在各时刻复制并展开，可视为一个深层前馈网络，但权重 W, U, V 在各时刻共享，权重反向传播时需考虑跨时步的依赖。

前向推理的计算过程算法伪代码和示意图如下所示：

function FORWARDRNN($\mathbf{x}$, *network*) **returns** output sequence $\mathbf{y}$

$\mathbf{h}_0 \leftarrow 0$
for $i \leftarrow 1$ **to** LENGTH($\mathbf{x}$) **do**
  $\mathbf{h}_i \leftarrow g(\mathbf{U}\mathbf{h}_{i-1} + \mathbf{W}\mathbf{x}_i)$
  $\mathbf{y}_i \leftarrow f(\mathbf{V}\mathbf{h}_i)$
return y

Figure 8.3 Forward inference in a simple recurrent network. The matrices **U**, **V** and **W** are shared across time, while new values for **h** and **y** are calculated with each time step.

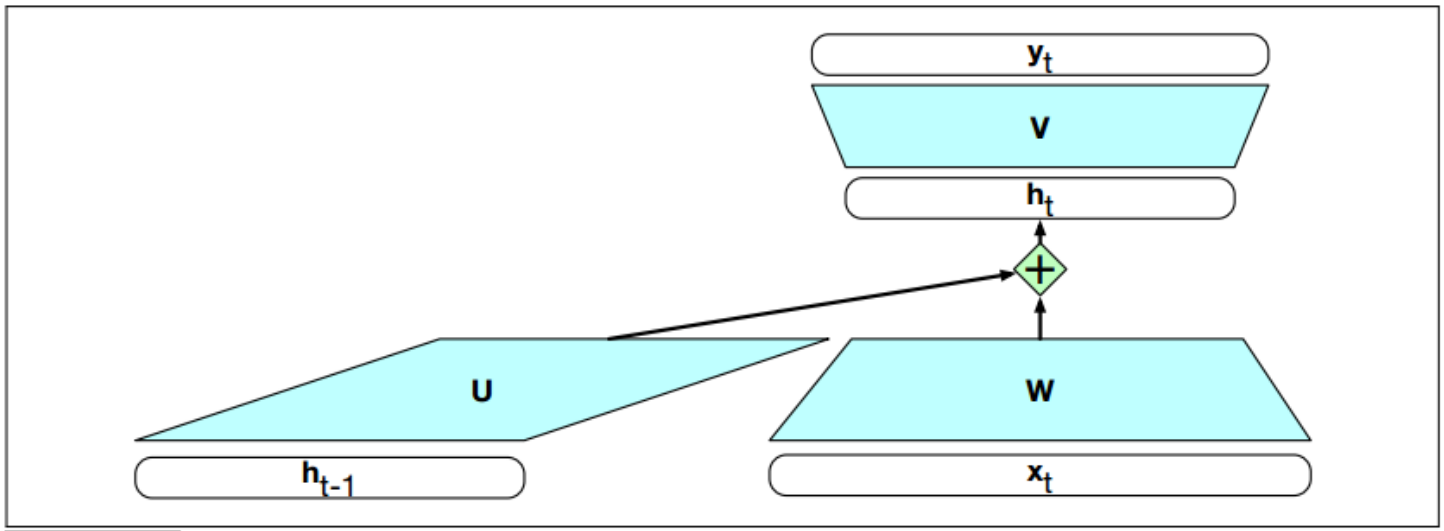


Figure 8.2 Simple recurrent neural network illustrated as a feedforward network. The hidden layer $\mathbf{h}_{t-1}$ from the prior time step is multiplied by weight matrix **U** and then added to the feedforward component from the current time step.

RNN的训练——时序反向传播（BPTT）

1. 损失函数与误差传播

对每步输出 y_t 计算交叉熵损失，并对隐藏层在当前及后续时刻的影响进行求和，得到针对 h_t 的总误差。

2. 双向遍历

前向：逐时刻计算 h_t, y_t，累积每一步的损失，并保存中间隐藏状态；

反向：从序列末端向前回传误差，计算各权重梯度。此即“通过时间的反向传播”（Backpropagation Through Time, BPTT）算法。

3. 梯度消失与网络深度

由于误差信号多次相乘，RNN易出现梯度消失或爆炸，导致**难以学习长距离依赖**。LSTM等门控结构的RNN变体可以缓解此类问题。

将RNN网络按时间步展开后，如下图所示，可视为一个深层前馈神经网络，可使用第七讲中的反向传播算法训练RNN的参数。

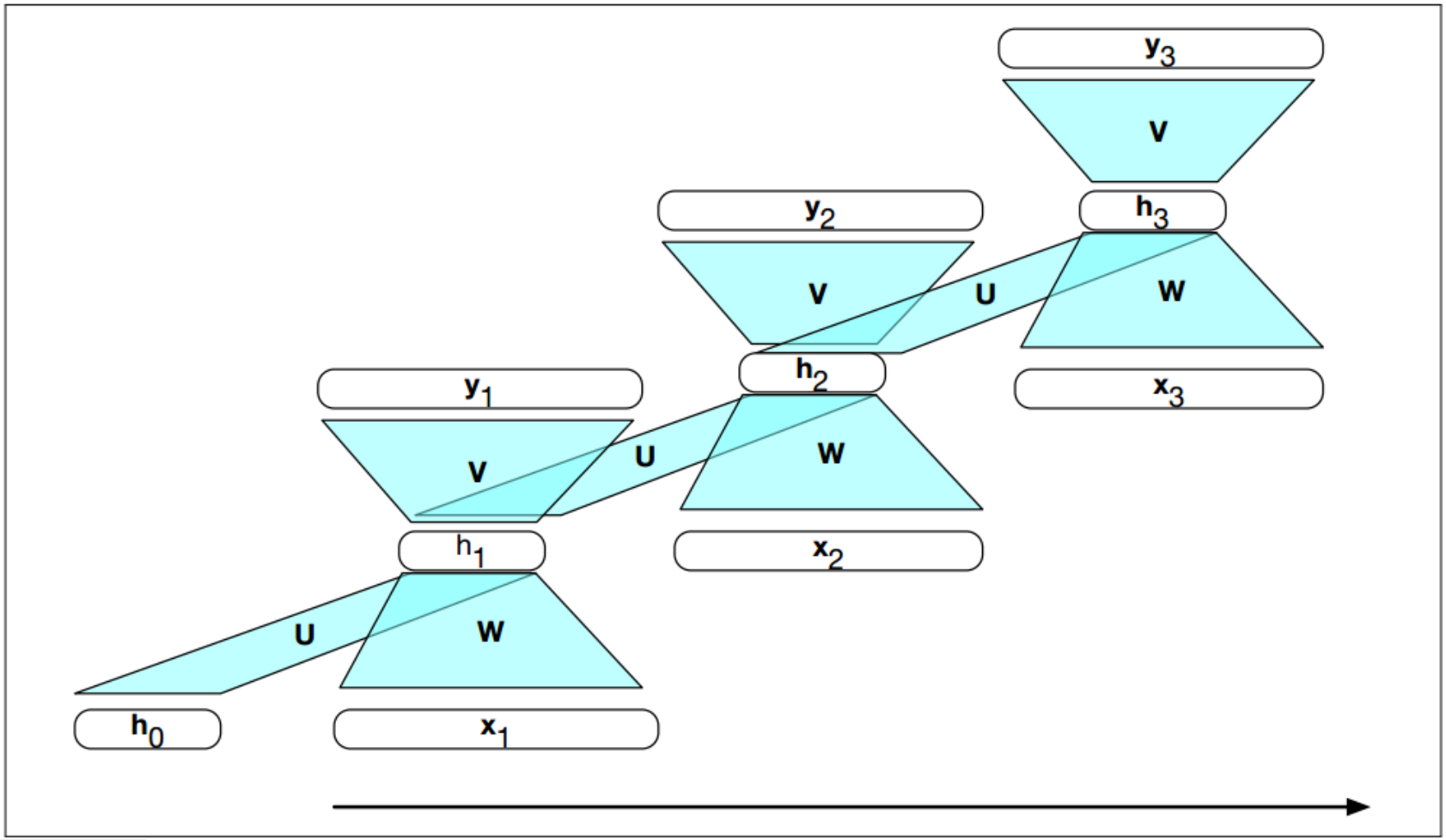


Figure 8.4 A simple recurrent neural network shown unrolled in time. Network layers are recalculated for each time step, while the weights **U**, **V** and **W** are shared across all time steps.

目录

语言模型中的RNN框架

任务目标：语言模型旨在根据先行上下文预测序列中下一个词的条件概率：

$$P(\text{fish} \mid \text{“Thanks for all the”})$$

链式法则：对整句概率建模时，将条件概率连乘：

$$P(w_{1:n}) = \prod_{i=1}^{n} P(w_i \mid w_{<i})$$

模型优势：与只依赖固定n-gram上下文的统计模型或前馈神经语言模型相比，RNN通过其隐状态 h_{t-1} **理论上**可保留从序列开头到当前的一切信息，无需任意窗口长度限制，**克服了**长距离依赖受限的问题。

使用前馈网络和RNN构建语言模型的差异

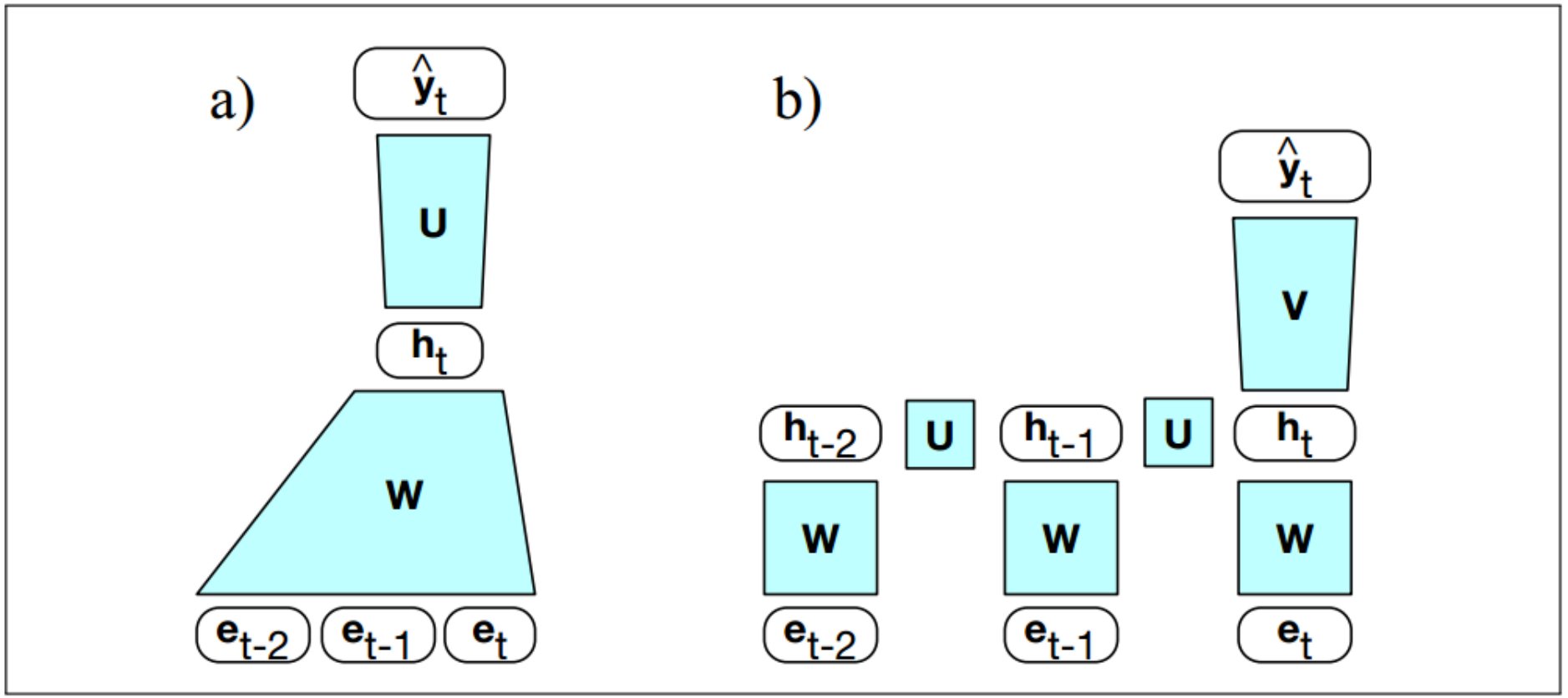


Figure 8.5 Simplified sketch of two LM architectures moving through a text, showing a schematic context of three tokens: (a) a feedforward neural language model which has a fixed context input to the weight matrix **W**, (b) an RNN language model, in which the hidden state $\mathbf{h}_{t-1}$ summarizes the prior context.

RNN语言模型前向推理

1. 输入表示：将当前词 x_t 编码为独热向量后，通过嵌入矩阵 E 映射到词向量

$$e_t = E\,x_t$$

2. 隐状态更新：结合前一步隐状态 h_{t-1} 与当前嵌入，计算新隐状态

$$h_t = g\big(U\,h_{t-1} + W\,e_t\big)$$

3. 输出层与概率：再经输出权重 V 与 softmax 得到对全词表的预测分布

$$\hat{y}_t = \mathrm{softmax}(V\,h_t)$$

4. 概率解释：词表中第 k 个词为下一个词的概率

$$P(w_{t+1} = k \mid w_{1:t}) = \hat{y}_t[k],$$

整句概率：各步预测概率连乘：$\prod_{i=1}^{n} \hat{y}_i[w_i]$。

RNN语言模型训练

自监督学习：直接利用原文序列进行训练，无需额外标注；

损失函数：在每个时刻对下一词 w_{t+1} 的负对数概率做交叉熵损失：

$$L_{\mathrm{CE}}(t) = -\log \hat{y}_t\left[w_{t+1}\right]$$

Teacher Forcing：训练过程中**始终使用真实前序词作为下一步输入**，而非模型预测，以加速收敛；

优化方式：通过梯度下降最小化平均交叉熵损失，端到端更新参数。

$$L = \frac{1}{T}\sum_{t=1}^{T} L_{CE}(t)$$

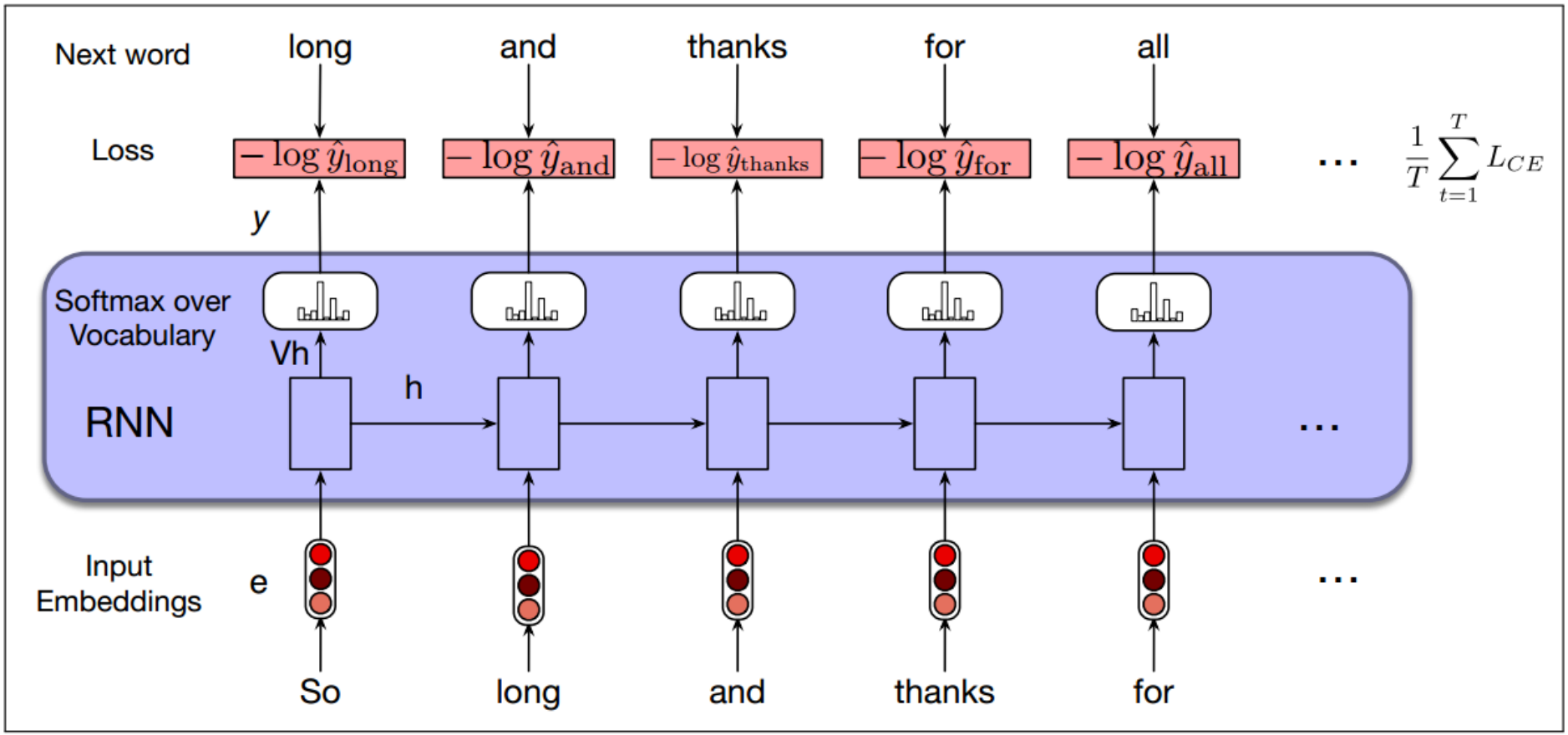


Figure 8.6 Training RNNs as language models.

权重绑定

动机：输入嵌入矩阵 E $(d \times |V|)$ 与输出层权重矩阵 V $(|V| \times d)$ 维度相同且语义相似，分别学习两套矩阵既浪费参数又易过拟合；

使用RNN构建语言模型时，假定隐藏层的维度与词嵌入的维度相同(即，d)，E 的每一列对应一个词语的词嵌入向量，Vh 计算的结果是词表中每一个词语的logit得分，V 的每一行对应词表中的一个词语。因此，V 相当于是RNN语言模型学到的第二个词嵌入矩阵。为避免冗余和提高训练效率，可省去参数矩阵 V，使用词嵌入矩阵代替，即，在输入层使用词嵌入矩阵 E，在输出层使用词嵌入矩阵的转置 E^T。

方法：让两者共享同一矩阵（输出时取其转置），即

$$\hat{y}_t = \text{softmax}\left(E^\top h_t\right)$$

同时输入映射仍为 $e_t = E x_t$。

效益：在减少参数量的同时通常也能进一步降低模型困惑度。

目录

序列标注（Sequence Labeling）

1. 任务定义：对序列中每个元素（如句子中的每个词）分配一个固定标签，如词性标注（POS tagging），需从标签集（如NOUN、VERB等）中选择最优标签。

2. 模型结构输入：每个时刻的词通过预训练词嵌入映射为向量；

RNN处理：依次将词嵌入输送入RNN（或LSTM），生成各时刻的隐藏状态；

输出层：对每个隐藏状态，使用softmax产生对应标签的概率分布；

解码：每时刻取概率最高标签作为预测。

3. 训练与损失：采用交叉熵损失，对每个时刻的标签预测**独立计算并求和** (同语言模型)；整体模型通过反向传播（含时间展开）端到端训练。

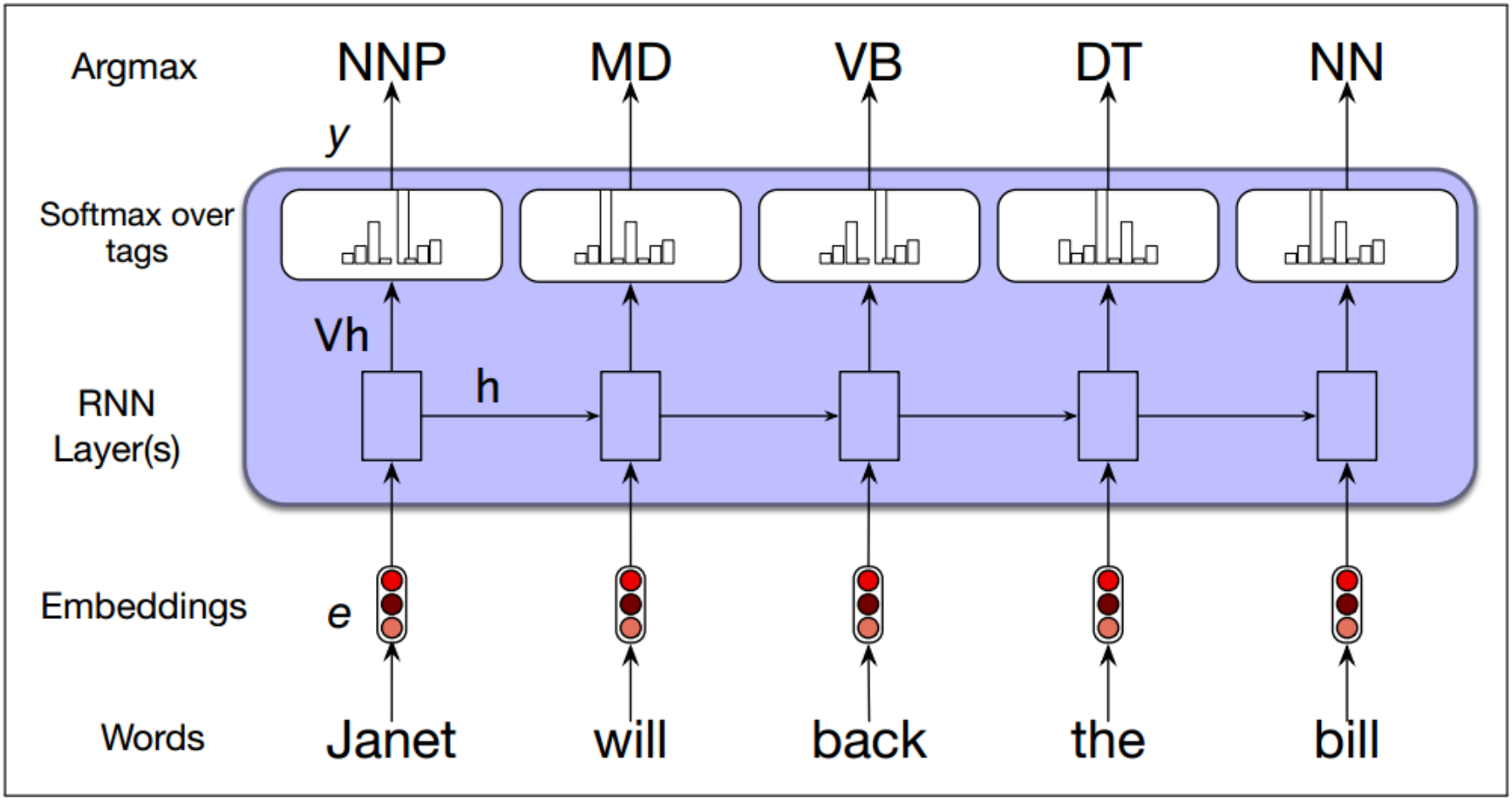


Figure 8.7 Part-of-speech tagging as sequence labeling with a simple RNN. The goal of part-of-speech (POS) tagging is to assign a grammatical label to each word in a sentence, drawn from a predefined set of tags. (The tags for this sentence include NNP (proper noun), MD (modal verb) and others; we'll give a complete description of the task of part-of-speech tagging in Chapter 17.) Pre-trained word embeddings serve as inputs and a softmax layer provides a probability distribution over the part-of-speech tags as output at each time step.

序列分类（Sequence Classification）

1. 任务定义：对整个文本序列生成一个整体标签，如情感分析（正/负面）、主题分类或垃圾邮件检测。

2. 模型结构

- 序列输入同上，逐词生成隐藏状态；

- 最后时刻隐状态 h_n 或所有时刻隐状态**池化（如取均值/最大值）**作为整句表示；

- 将该表示输入至一级或多级前馈神经网络，最终经softmax输出序列级类别概率。

3. 训练与损失：仅对序列级输出计算交叉熵损失 (同多类别逻辑回归)；误差信号通过前馈分类器及整个RNN网络端到端反向传播。

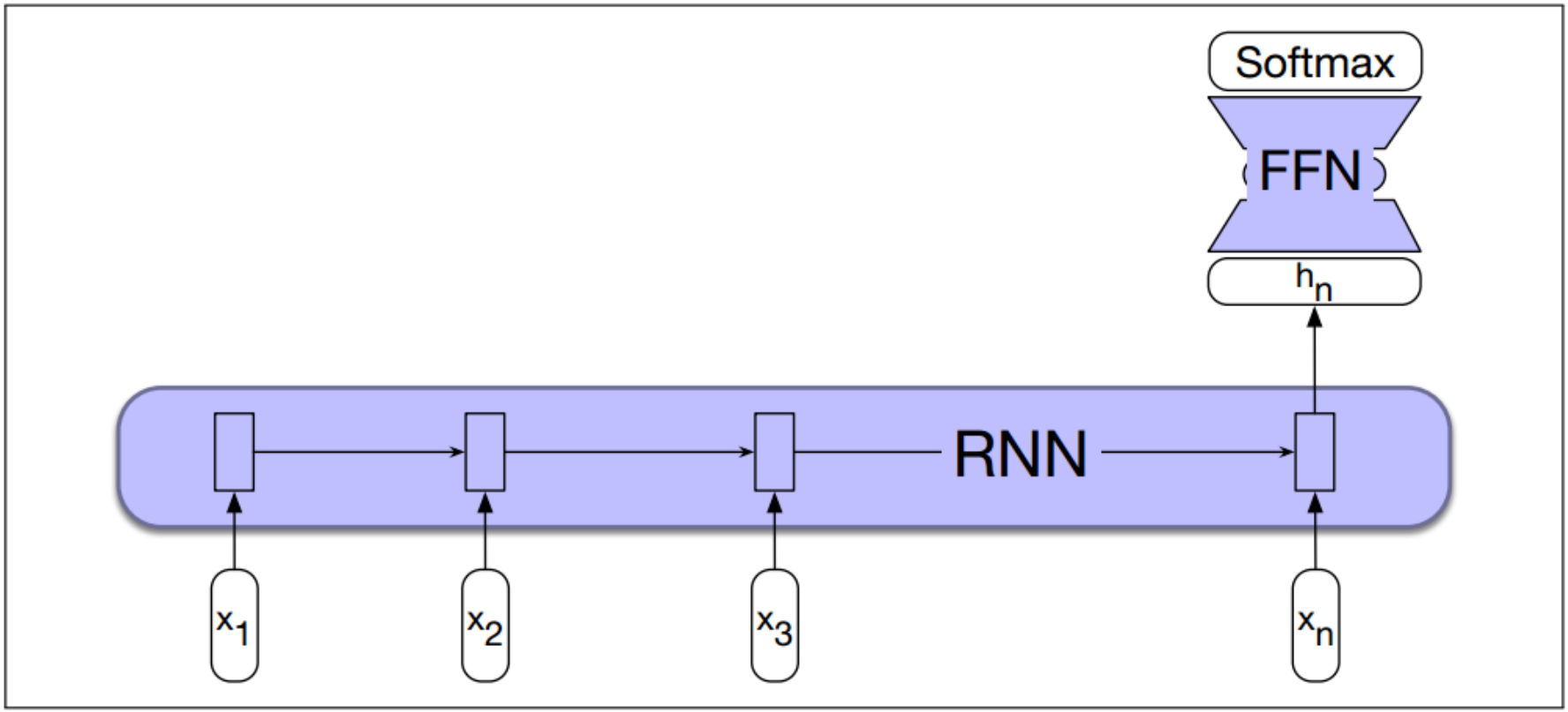


Figure 8.8 Sequence classification using a simple RNN combined with a feedforward network. The final hidden state from the RNN is used as the input to a feedforward network that performs the classification.

文本生成（Generation with RNN Based Language Models）

1. 任务定义：基于已生成词或给定上下文，**逐步生成**后续文本，用于机器翻译、摘要、对话等。

2. 自回归生成（Autoregressive Generation）流程

- 以特定起始符（如“<s>”）为第一输入，计算隐藏状态并通过softmax抽样或贪心选择下一个词；

- 将该词的嵌入作为下一时刻输入，迭代以上过程；

- 直至达到长度阈值或生成结束符（“</s>”）。

3. 模型训练

- 与语言建模一致，使用Teacher Forcing：训练时每步输入真实前序词；

- 计算每步对下一个词的交叉熵损失并平均，端到端优化所有参数。

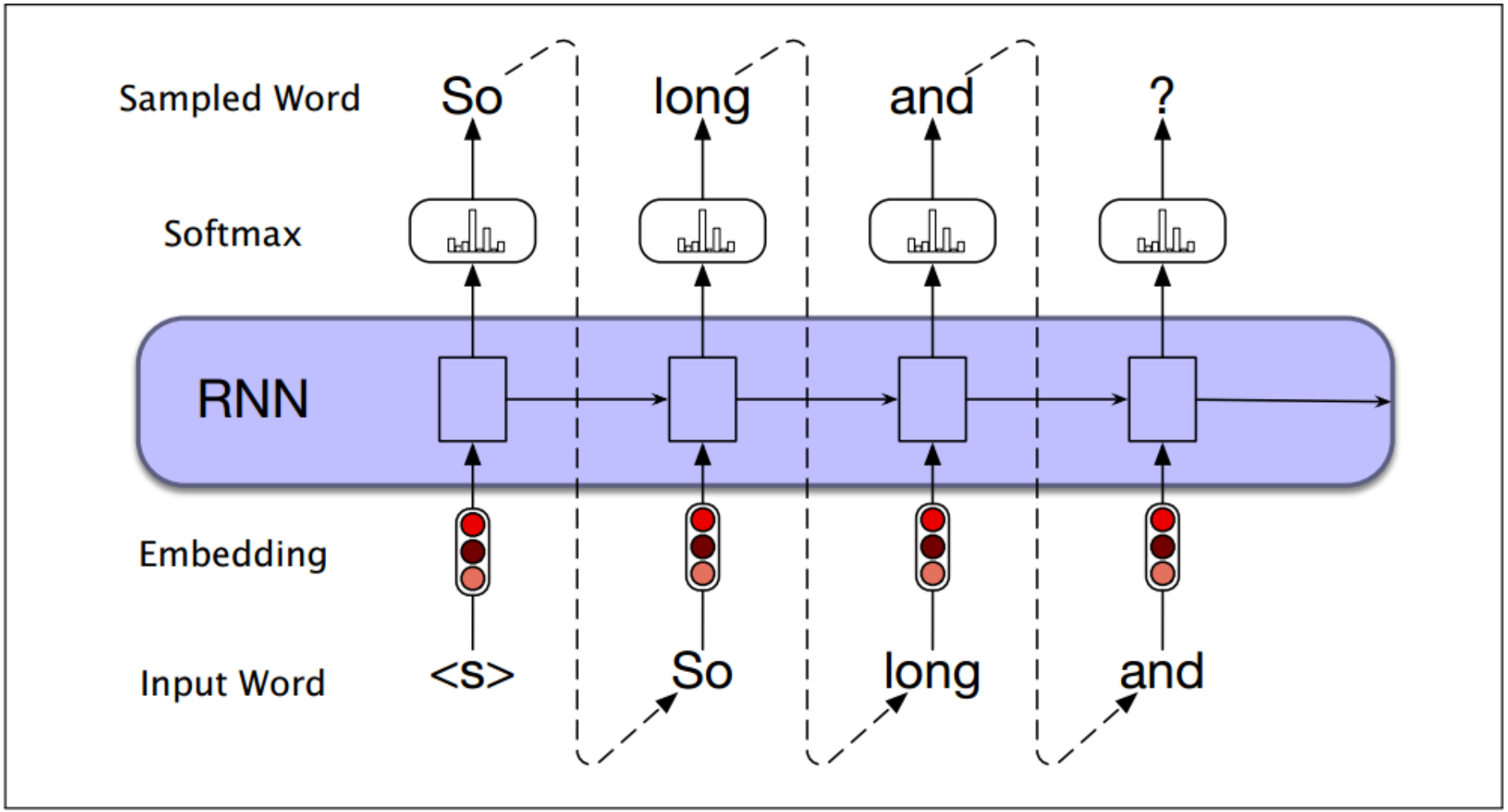


Figure 8.9 Autoregressive generation with an RNN-based neural language model.

目录

Stacked RNNs（多层堆叠循环网络）

1. 定义：将多个循环网络按层级串联起来，前一层的输出序列作为后一层的输入序列。

2. 结构示意

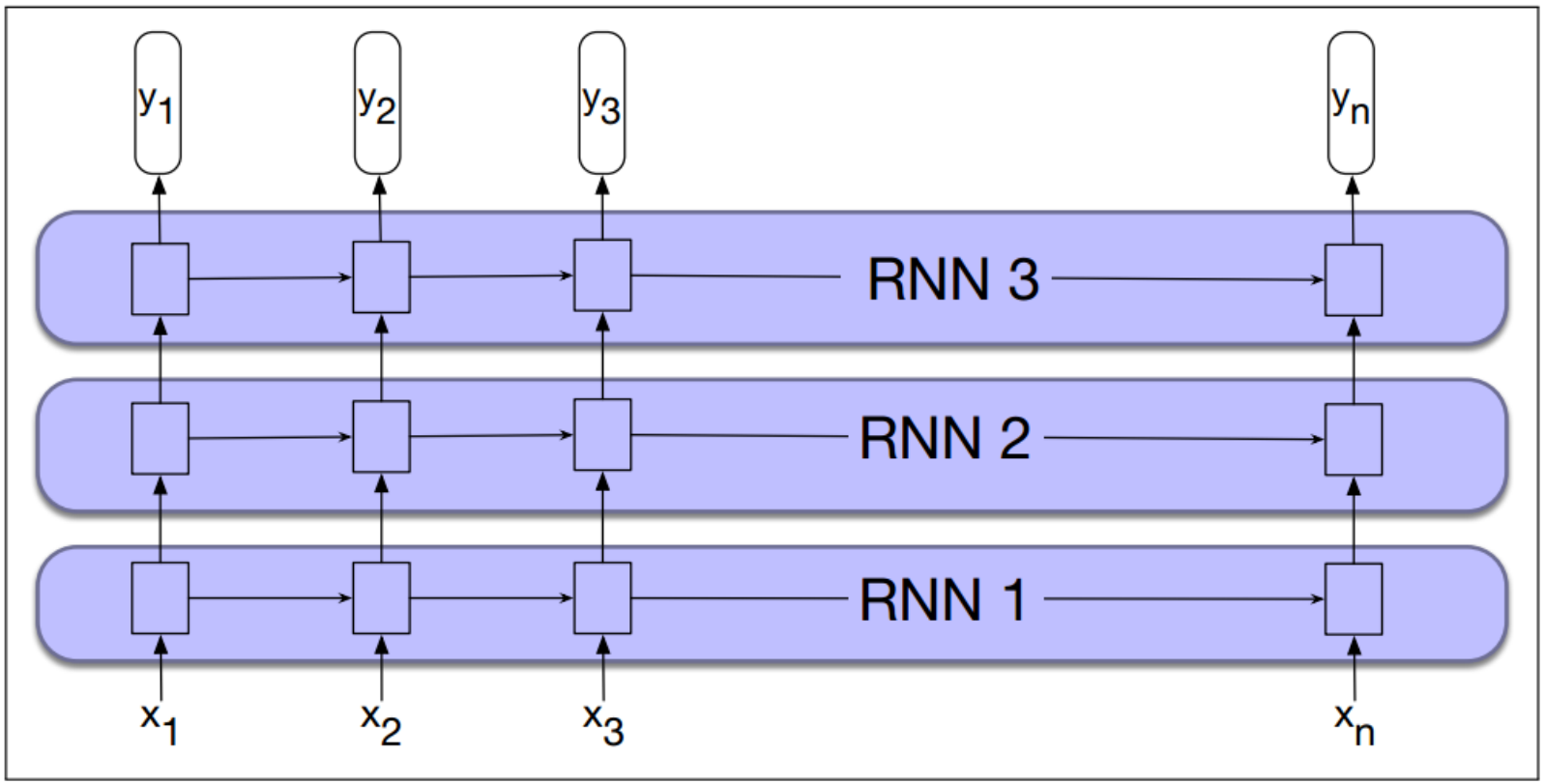


Figure 8.10 Stacked recurrent networks. The output of a lower level serves as the input to higher levels with the output of the last network serving as the final output.

Bidirectional RNNs（双向循环网络）

1. 定义：同时训练两个方向相反的循环网络——一个从序列开头向末尾遍历（正向），一个从末尾向开头遍历（反向），并在每个时刻将二者的隐状态拼接 (级联, concatenation)或融合。

2. 公式

$$
\begin{aligned}
h_t^{\mathsf{f}} &= \mathrm{RNN}_{\mathsf{forward}}(x_1, \ldots, x_t), \\
h_t^{\mathsf{b}} &= \mathrm{RNN}_{\mathsf{backward}}(x_t, \ldots, x_n), \\
h_t &= [\, h_t^{\mathsf{f}};\, h_t^{\mathsf{b}} \,] \quad (\text{或 } \oplus).
\end{aligned}
$$

将正向隐状态 h_t^{f} 与反向隐状态 h_t^{b} 级联，形成新的双向表示。

3. 优点

- 全上下文感知：对当前时刻的预测，可同时利用左右两侧的信息，显著提升标注和分类任务的准确性；

- 应用扩展：在序列标注任务中，每个词的标签决策均可参考后续词汇；在序列分类中，可将前后两个方向的最终隐状态拼接后再分类，避免单向RNN对末尾信息过度偏重。

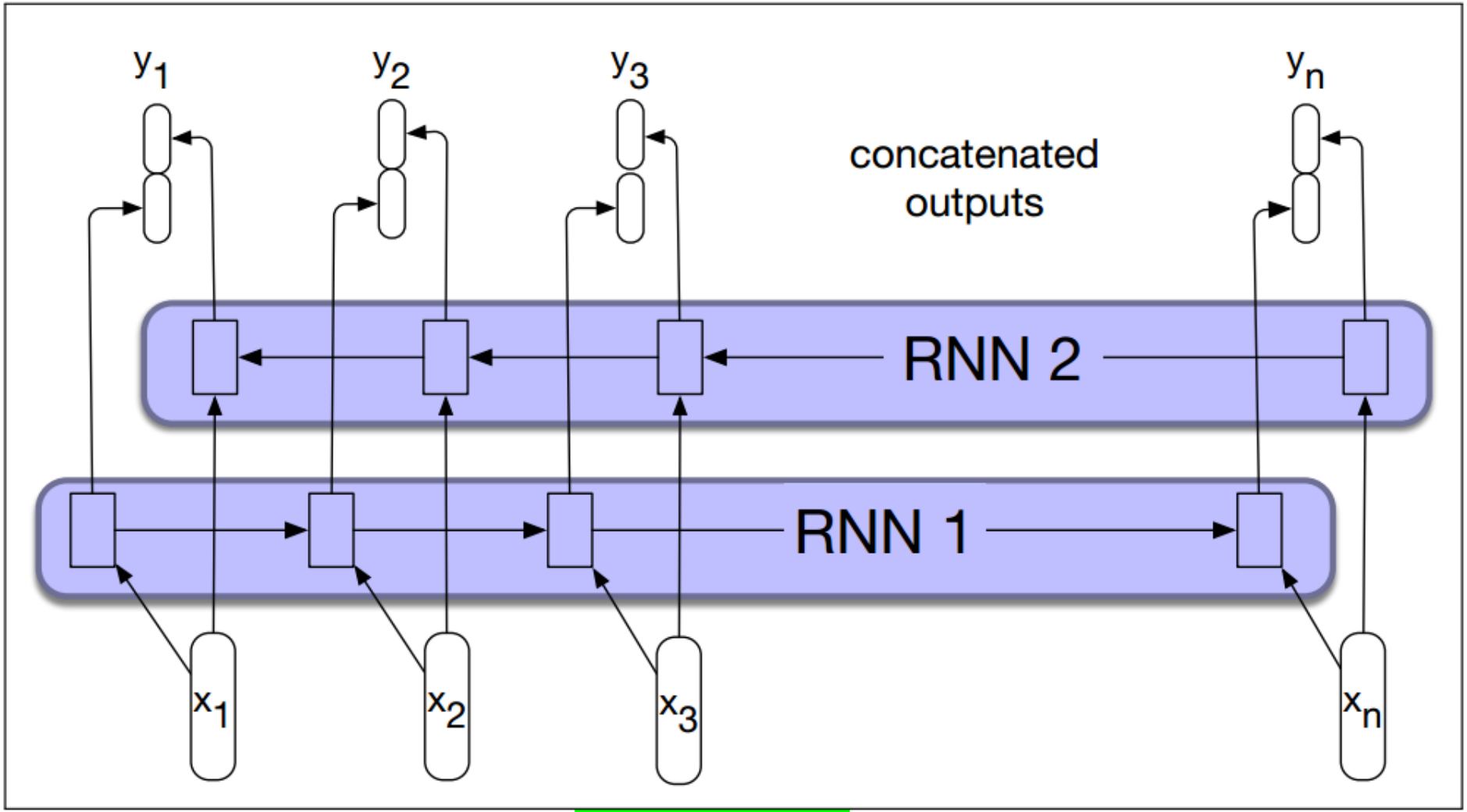


Figure 8.11 A bidirectional RNN. Separate models are trained in the forward and backward directions, with the output of each model at each time point concatenated to represent the bidirectional state at that time point.

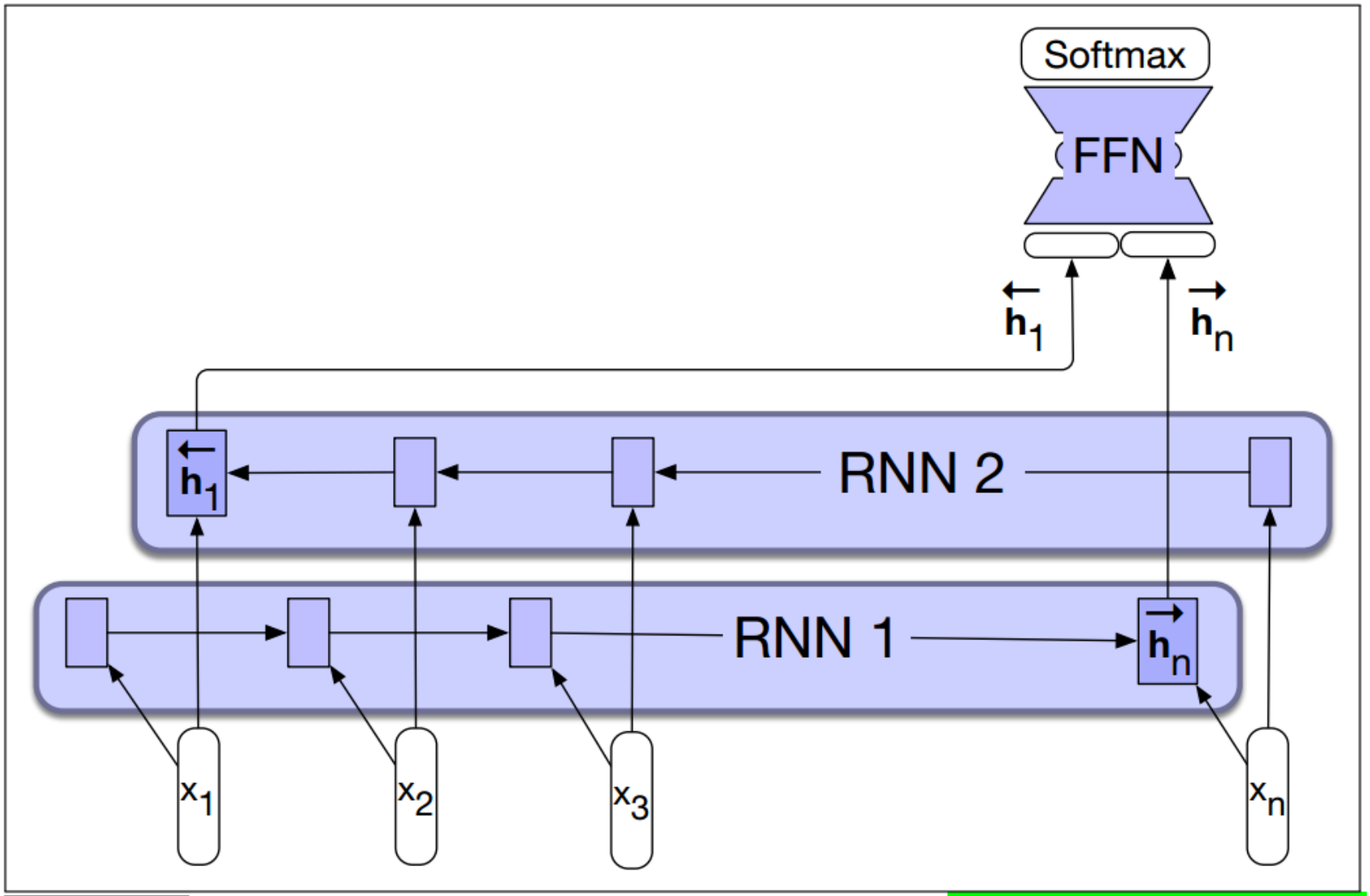


Figure 8.12 A bidirectional RNN for sequence classification. The final hidden units from the forward and backward passes are combined to represent the entire sequence. This combined representation serves as input to the subsequent classifier.

目录

1. 背景与动机

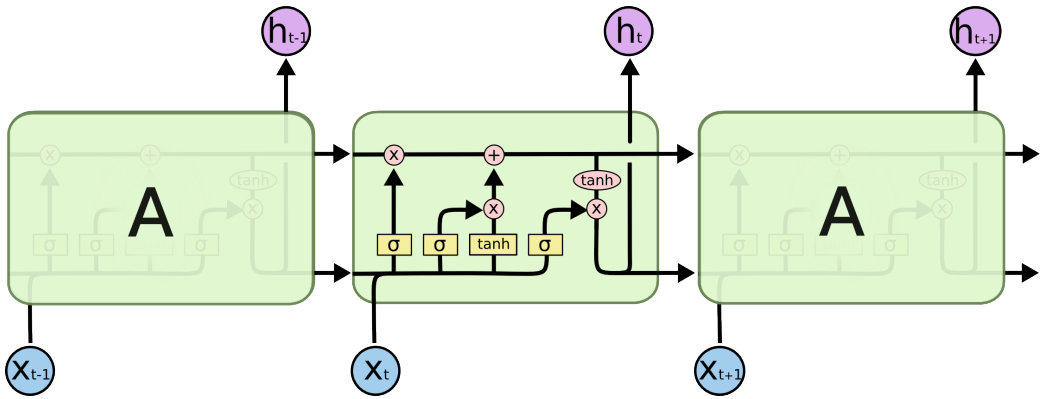
在长序列建模中，普通RNN虽然理论上能够访问任意远的历史信息，但由于两方面原因常常难以捕捉远距离依赖：

信息干扰：隐藏层既要提供当前决策所需信息，又要更新并保留未来需要的信息，二者相互冲突；

梯度消失：反向传播时，误差需跨越多步相乘，易导致梯度指数级衰减。

2. LSTM架构概述

长短期记忆网络（LSTM）通过在传统RNN中引入显式的“细胞状态”（context）和门控机制，将“忘记无用信息”与“保留重要信息”两个职责分离，从而有效管理长期依赖。



四类门及其计算

所有门单元均采用“线性变换→Sigmoid”产生[0,1]范围的掩码，再与相应向量做逐元素乘法，以控制信息流：

1. 忘记门（Forget Gate）

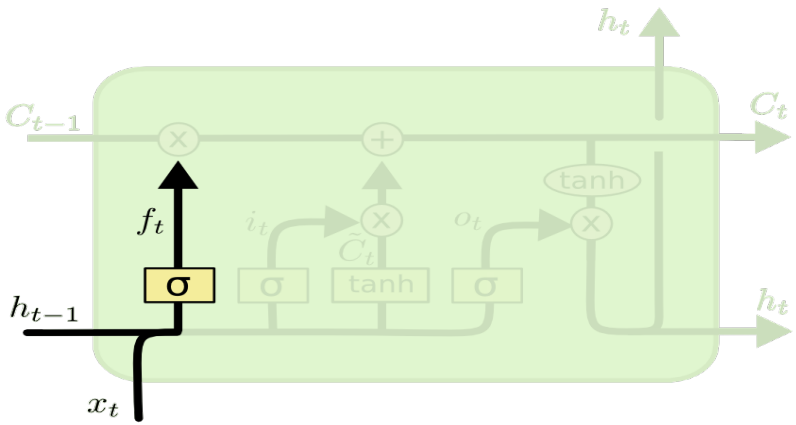
$$f_t = \sigma\big(U_f h_{t-1} + W_f x_t\big), \quad k_t = c_{t-1} \odot f_t$$

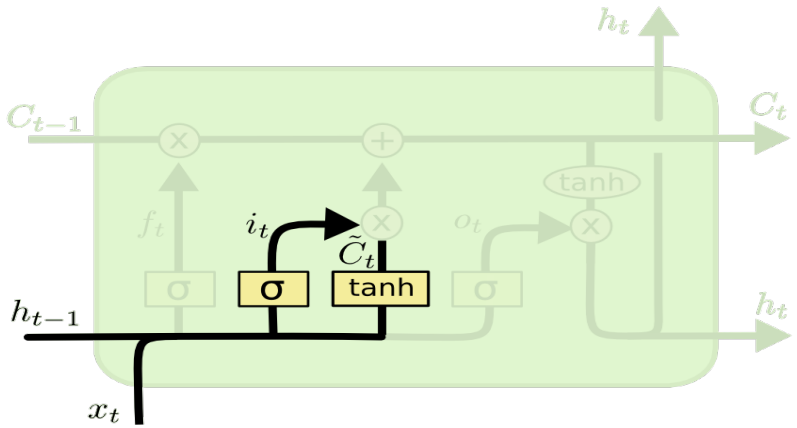
用以删除先前细胞状态中不再需要的信息。

2. 新增信息门（Input/Add Gate）

$$g_t = \tanh\big(U_g h_{t-1} + W_g x_t\big), \quad i_t = \sigma\big(U_i h_{t-1} + W_i x_t\big), \quad j_t = g_t \odot i_t$$

计算当前时刻应当写入细胞的新信息。





3. 细胞状态更新

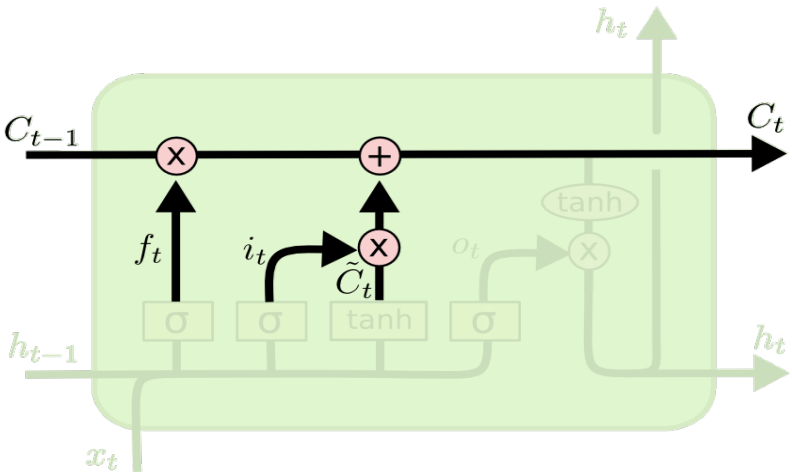
$$c_t = k_t + j_t$$

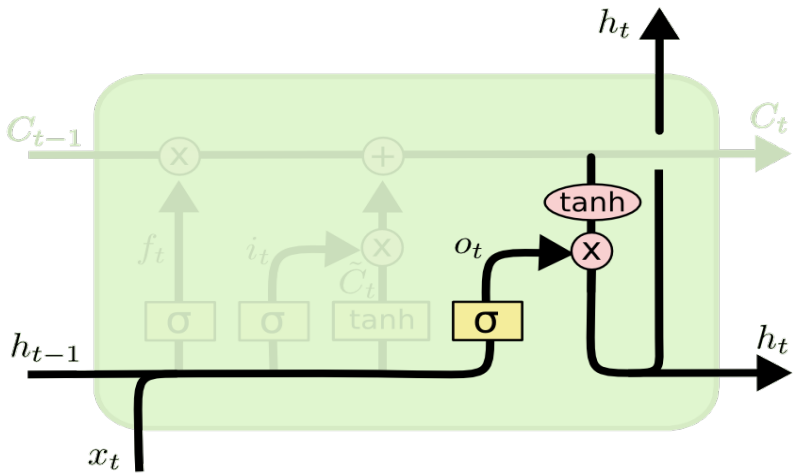
将保留信息和新增信息合并，形成新的细胞状态。

4. 输出门（Output Gate）

$$o_t = \sigma\big(U_o h_{t-1} + W_o x_t\big), \quad h_t = o_t \odot \tanh(c_t)$$

决定哪些细胞状态将影响当前隐藏状态输出。





单元运算总结

每个LSTM单元输入当前外部输入 x_t、上一步隐藏状态 h_{t-1} 及上一步细胞状态 c_{t-1}，经过四类门的并行运算后，输出新隐藏状态 h_t 与新细胞状态 c_t；该模块化单元可随意嵌入堆叠或双向网络中。

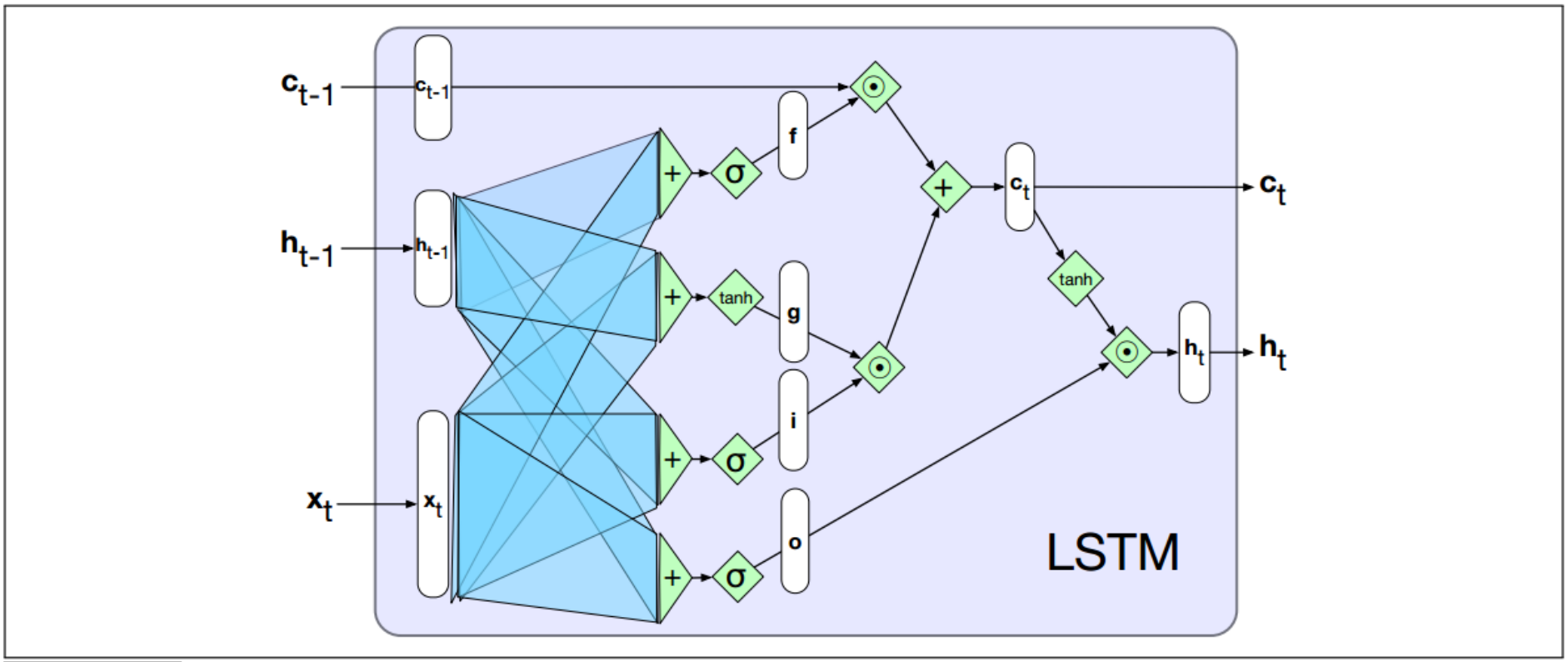


Figure 8.13 A single LSTM unit displayed as a computation graph. The inputs to each unit consists of the current input, x, the previous hidden state, h_{t-1}, and the previous context, c_{t-1}. The outputs are a new hidden state, h_t and an updated context, c_t.

模块化与实际应用

- **高度模块化**：LSTM单元内部封装复杂门控逻辑，对上层网络保持“黑盒”接口；

- **灵活替换**：可直接替换任何RNN单元，用于堆叠（stacked）或双向（bidirectional）架构中；

- **主流标准**：由于其在长距依赖建模和梯度稳定性方面的显著优势，LSTM已成为现代序列建模的默认单元。

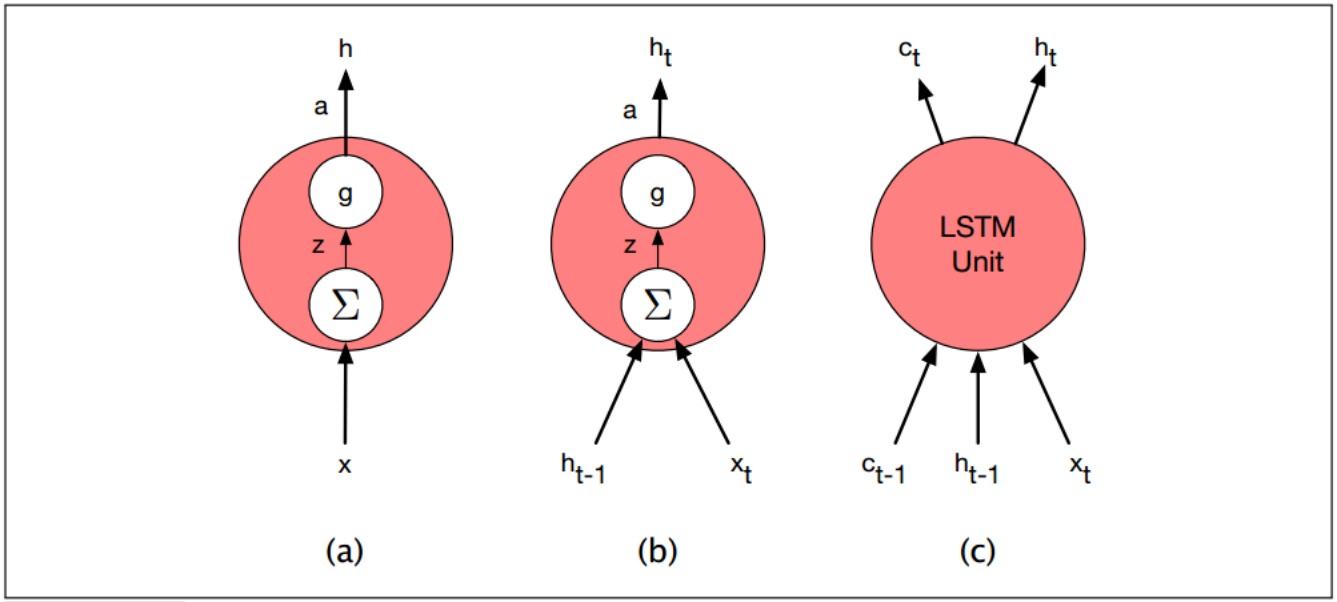


Figure 8.14 Basic neural units used in feedforward, simple recurrent networks (SRN), and long short-term memory (LSTM).

目录

1. 序列标注（Sequence Labeling）

- 任务性质：对输入序列中每个元素（如每个词）分配一个标签，典型例子为词性标注（POS tagging）。

- 模型结构：将词嵌入依次送入 RNN（或 LSTM），在每个时刻输出一个隐藏状态，再通过 softmax 层预测该时刻的标签。

- 训练目标：对每个时刻的标签预测计算交叉熵损失，并对所有时刻损失求和，端到端反向传播更新 RNN 参数。

2. 序列分类（Sequence Classification）

- 任务性质：对整个序列生成一个整体标签，常见应用包括情感分析、主题分类、垃圾邮件检测等。

- 模型结构：RNN 逐词计算隐藏状态后，通常取最后时刻的隐藏状态 h_n（或对所有时刻隐藏状态做池化，如均值/最大值），再经一层前馈网络和 softmax 输出类别分布。

- 训练目标：仅对序列级别的预测结果计算交叉熵损失，误差通过前馈层再传回 RNN，实现端到端训练。

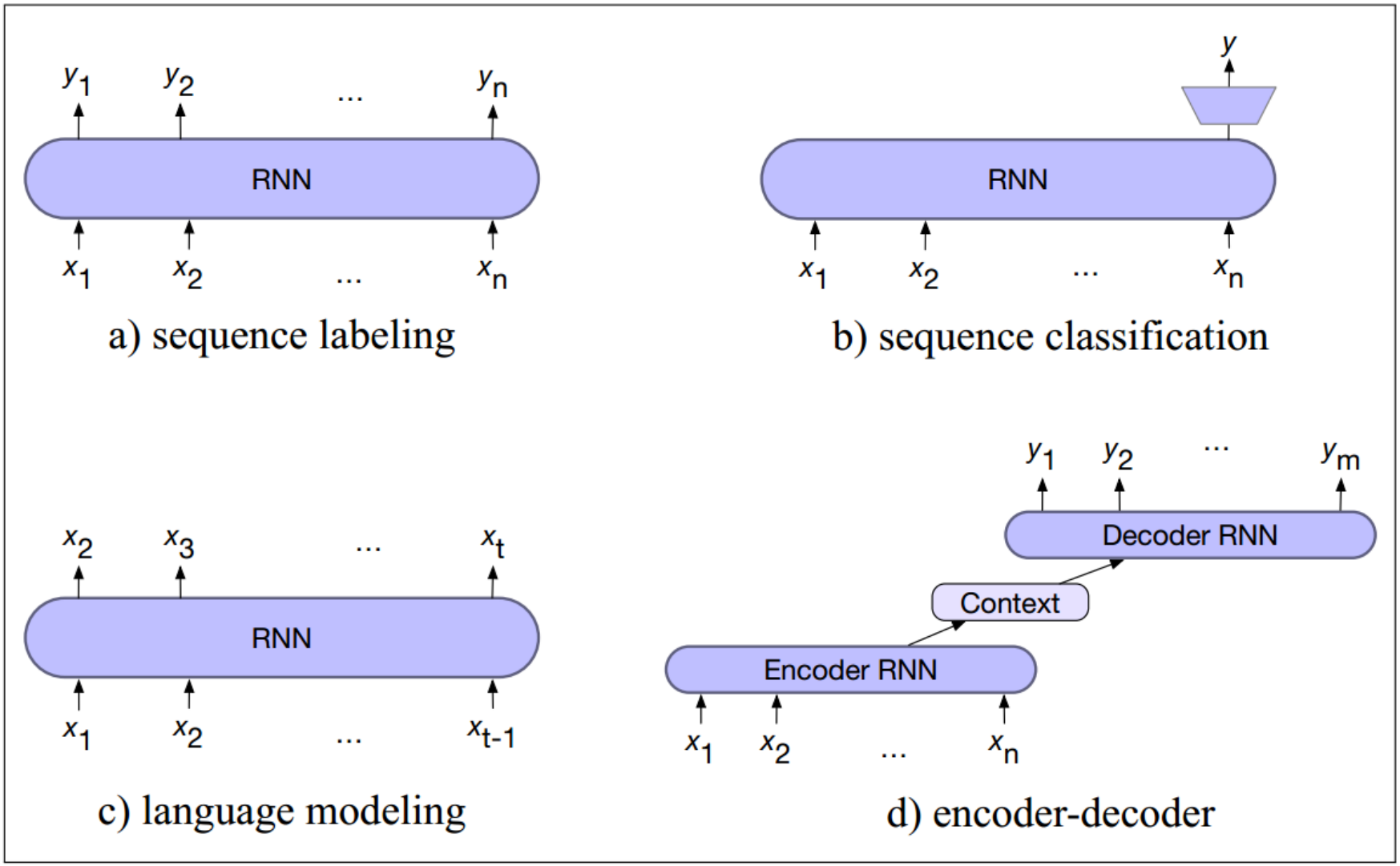


Figure 8.15 Four architectures for NLP tasks. In sequence labeling (POS or named entity tagging) we map each input token x_i to an output token y_i. In sequence classification we map the entire input sequence to a single class. In language modeling we output the next token conditioned on previous tokens. In the encoder model we have two separate RNN models, one of which maps from an input sequence **x** to an intermediate representation we call the **context**, and a second of which maps from the context to an output sequence **y**.

目录

Encoder-Decoder模型的基本概念

- **任务描述：**Encoder-Decoder模型主要用于处理**输入序列与输出序列长度不一致且映射关系复杂的任务**，尤其适用于机器翻译等任务。与传统的序列标注任务（如词性标注）不同，Encoder-Decoder模型的输入序列与输出序列的长度不匹配，且每个输入词与输出词之间没有简单的一一对应关系。

- **应用领域：**该模型广泛应用于机器翻译、文本摘要、问答系统和对话生成等任务。在机器翻译中，输入序列和输出序列不仅可能有不同长度，而且词汇的排列顺序也可能大不相同。

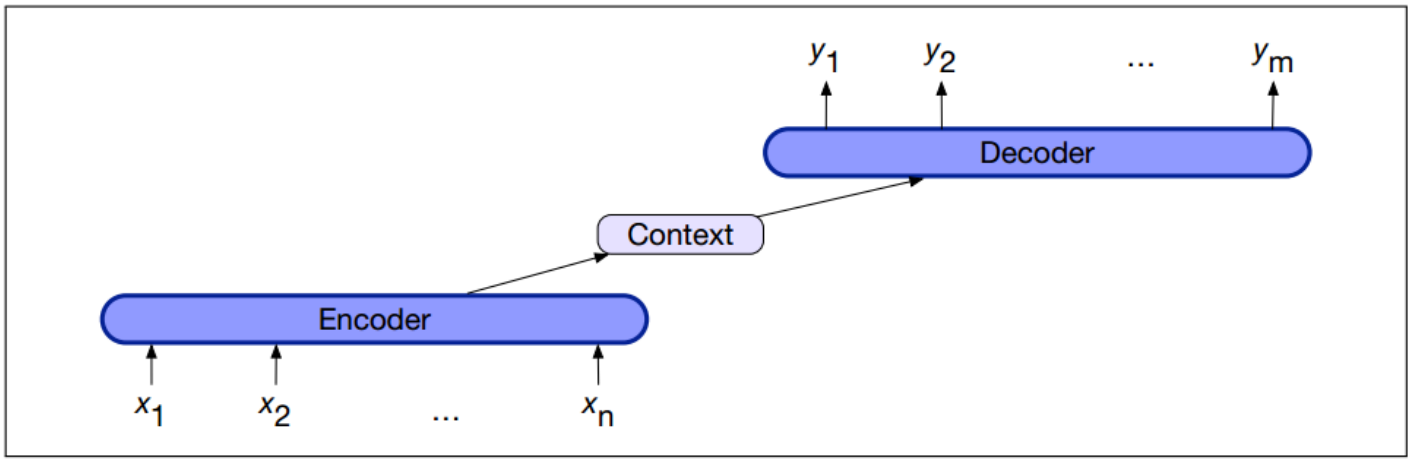


Figure 8.16 The encoder-decoder architecture. The context is a function of the hidden representations of the input, and may be used by the decoder in a variety of ways.

模型结构与工作原理

Encoder-Decoder模型由三个主要部分组成：

1. Encoder： 接受输入序列 $x_1, x_2, \ldots, x_n$，并生成一系列上下文化的表示 $h_1, h_2, \ldots, h_n$，这些表示捕捉了输入序列的语义信息。Encoder通常使用RNN、LSTM或GRU等网络。

2. Context Vector： Encoder的最后隐藏状态 h_n 被用作上下文向量（context vector），这个向量包含了输入序列的核心信息，并传递给Decoder作为生成输出的依据。

3. Decoder： Decoder接受Context向量作为输入，并生成一个长度可变的输出序列 $y_1, y_2, \ldots, y_m$，这些输出通过Decoder的隐状态和前一时刻的输出生成。

在Encoder-Decoder模型中，给定输入序列 x 下生成目标序列 y 的条件概率计算为：

$$p(y|x) = p(y_1|x)p(y_2|y_1, x)p(y_3|y_1, y_2, x) \ldots p(y_m|y_1, \ldots, y_{m-1}, x)$$

RNN Encoder-Decoder架构

在基础的RNN Encoder-Decoder模型中：

- **Encoder**：将输入序列转换为一个上下文表示（context），通过一系列的隐状态 $h_1, h_2, \ldots, h_n$ 来表征整个输入序列的语义。

- **Decoder**：Decoder在每一步生成一个新的输出 y_t，每个输出依赖于当前的Decoder隐状态 h_t 和前一步的生成输出。最简单的Decoder版本通过Context向量初始化其第一个隐状态 h_0，然后递归地生成输出。

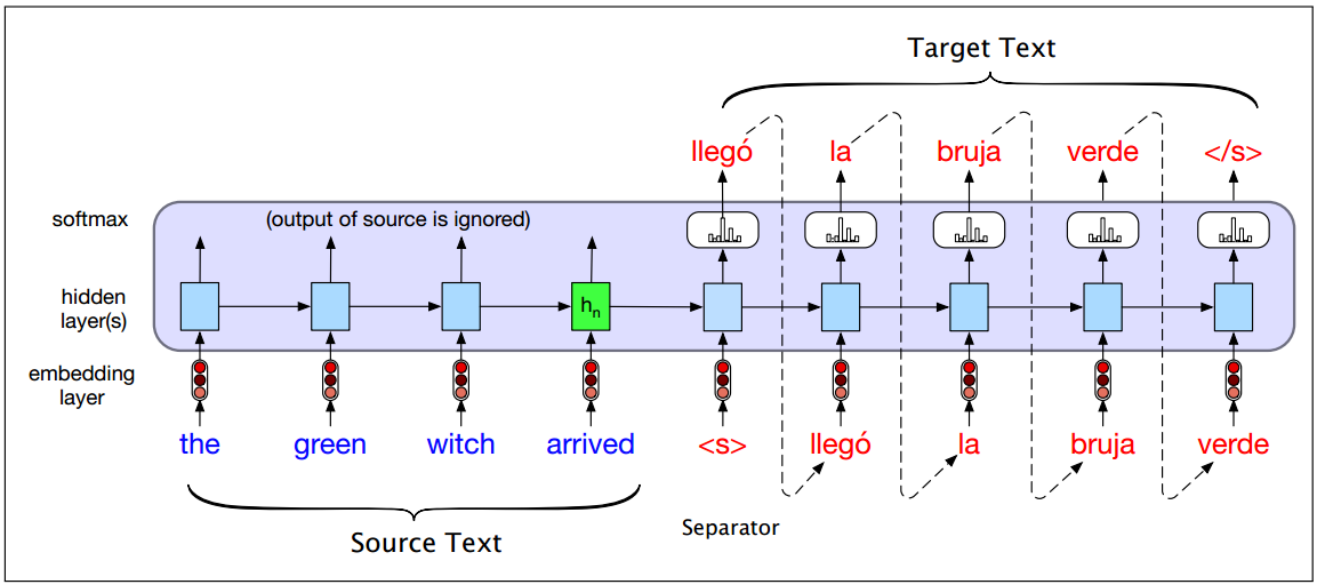


Figure 8.17 Translating a single sentence (inference time) in the basic RNN version of encoder-decoder approach to machine translation. Source and target sentences are concatenated with a separator token in between, and the decoder uses context information from the encoder's last hidden state.

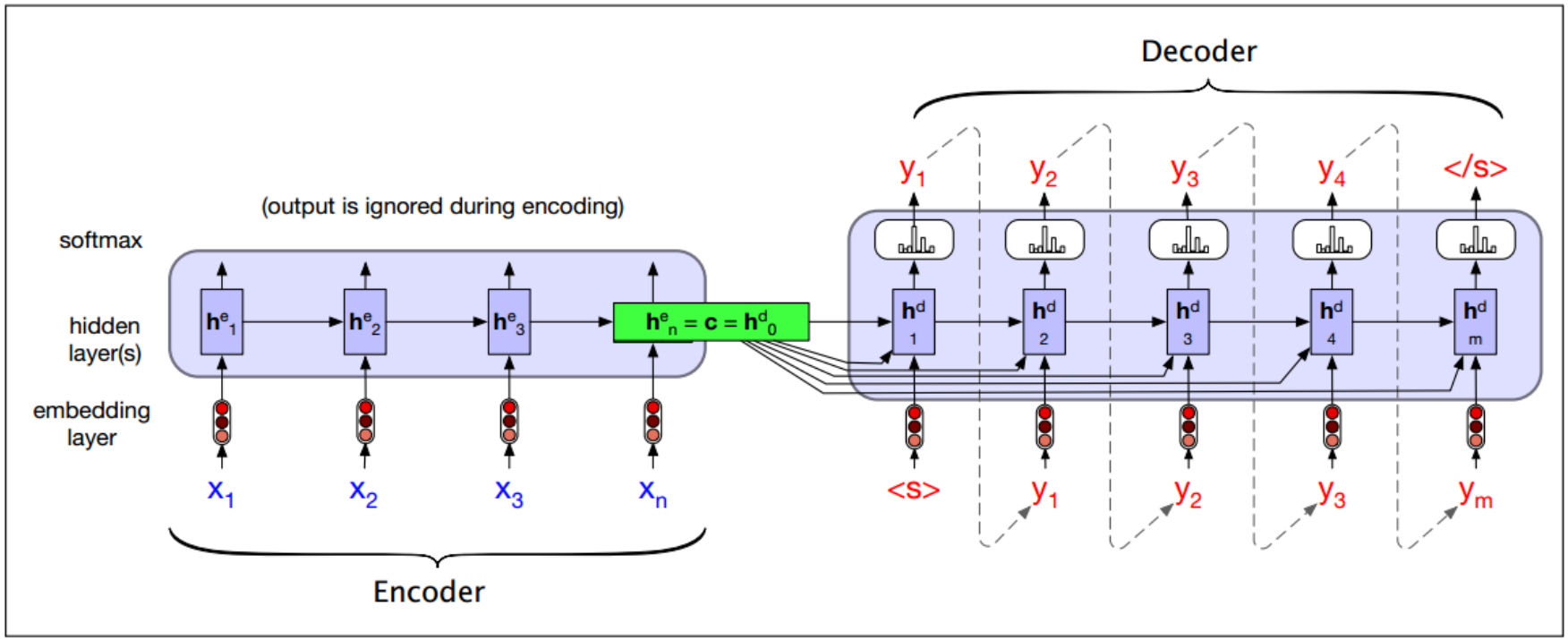


Figure 8.18 A more formal version of translating a sentence at inference time in the basic RNN-based encoder-decoder architecture. The final hidden state of the encoder RNN, h_n^e, serves as the context for the decoder in its role as h_0^d in the decoder RNN, and is also made available to each decoder hidden state.

Encoder通过对输入序列的处理，生成一个上下文向量 c，它总结了整个输入序列的语义信息。这个上下文向量 c 会在Decoder的每个步骤中被用来生成输出，确保Decoder对整个输入序列的信息都有所了解：

$$h_d^t = g(y_{t-1}^d, h_{t-1}^d, c)$$

目录

背景：Encoder-Decoder模型的瓶颈问题

- **瓶颈问题：**在传统的Encoder-Decoder模型中，Encoder将整个输入序列映射到一个固定的上下文向量（context vector），通常是Encoder最后一个时间步的隐藏状态 h_n。这个上下文向量必须总结输入序列的所有信息，以便Decoder使用它来生成目标序列。然而，这种做法存在局限性，尤其是对于长句子，输入序列的早期信息可能无法充分传递给Decoder。

- **解决方案：**为了解决这一瓶颈问题，引入了Attention机制，该机制使得Decoder能够从Encoder的所有隐藏状态中获取信息，而不仅仅是最后一个隐藏状态。

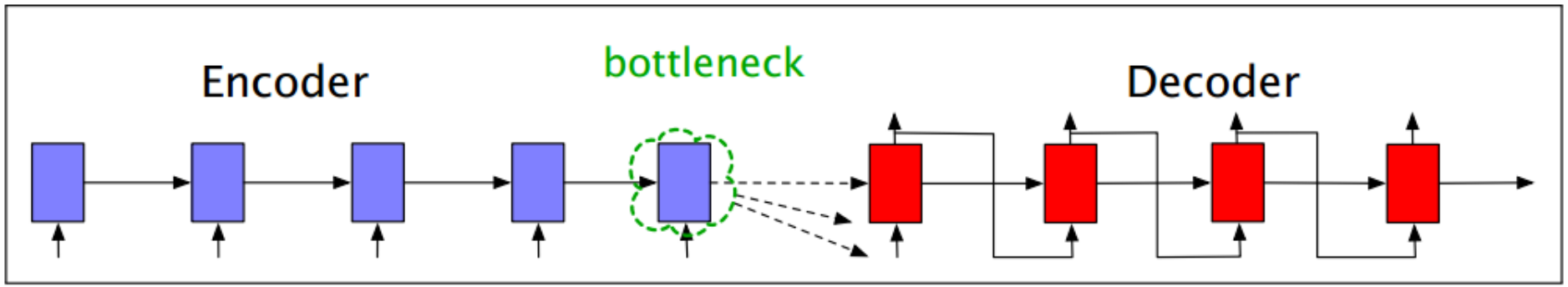


Figure 8.20 Requiring the context c to be only the encoder's final hidden state forces all the information from the entire source sentence to pass through this representational bottleneck.

Attention机制的工作原理

- **动态上下文向量：** 与传统的Encoder-Decoder模型不同，Attention机制中的上下文向量不再是固定的，而是根据Decoder当前的状态动态生成。具体来说，每个时间步的上下文向量 c_i 是Encoder所有隐藏状态的加权平均。

- **计算方法：** 上下文向量 c_i 的生成依赖于Decoder当前的隐藏状态 h_d^{i-1} 和Encoder的所有隐藏状态 $h_e^1, h_e^2, ..., h_e^n$，通过计算每个Encoder隐藏状态与当前Decoder隐藏状态之间的相关性，得到加权系数。

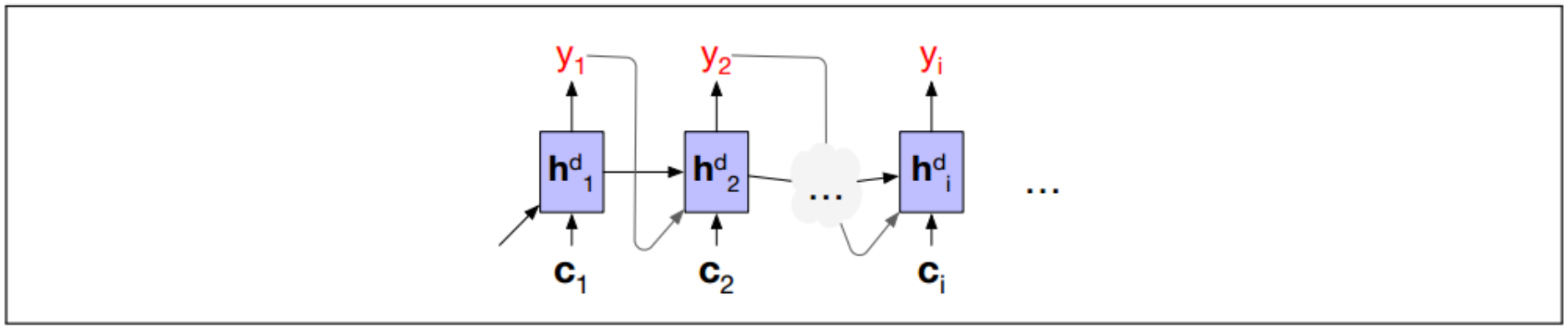


Figure 8.21 The attention mechanism allows each hidden state of the decoder to see a different, dynamic, context, which is a function of all the encoder hidden states.

计算相关性与加权平均

- **相关性评分：**在每一步解码时，通过计算Decoder当前隐藏状态 h_d^{i-1} 与每个Encoder隐藏状态 h_e^j 之间的相关性来确定关注的区域。最简单的计算方法是点积注意力（dot-product attention），通过计算两个向量的点积来度量它们的相似性：

$$\text{score}(h_d^{i-1}, h_e^j) = h_d^{i-1} \cdot h_e^j$$

这个点积结果表示Decoder和Encoder在当前时刻的相关性。

- **归一化与加权：将这些相关性分数通过softmax归一化，**得到加权系数 α_{ij}，这些系数表示每个Encoder隐藏状态对当前Decoder状态的重要性。然后，通过对所有Encoder隐藏状态加权平均，计算得到当前的上下文向量 c_i：

$$c_i = \sum_j \alpha_{ij} h_e^j$$

这个上下文向量 c_i 会被用作当前Decoder状态的输入。

Decoder的更新

- **动态更新：**每次解码时，Decoder的当前隐藏状态 h_d^i 依赖于之前的隐藏状态 h_d^{i-1}、当前生成的词 y_{i-1} 以及从Attention机制得到的上下文向量 c_i。具体的更新公式为：

$$h_d^i = g(y_{i-1}, h_d^{i-1}, c_i)$$

其中，g 是激活函数，通常是RNN单元或其变种。

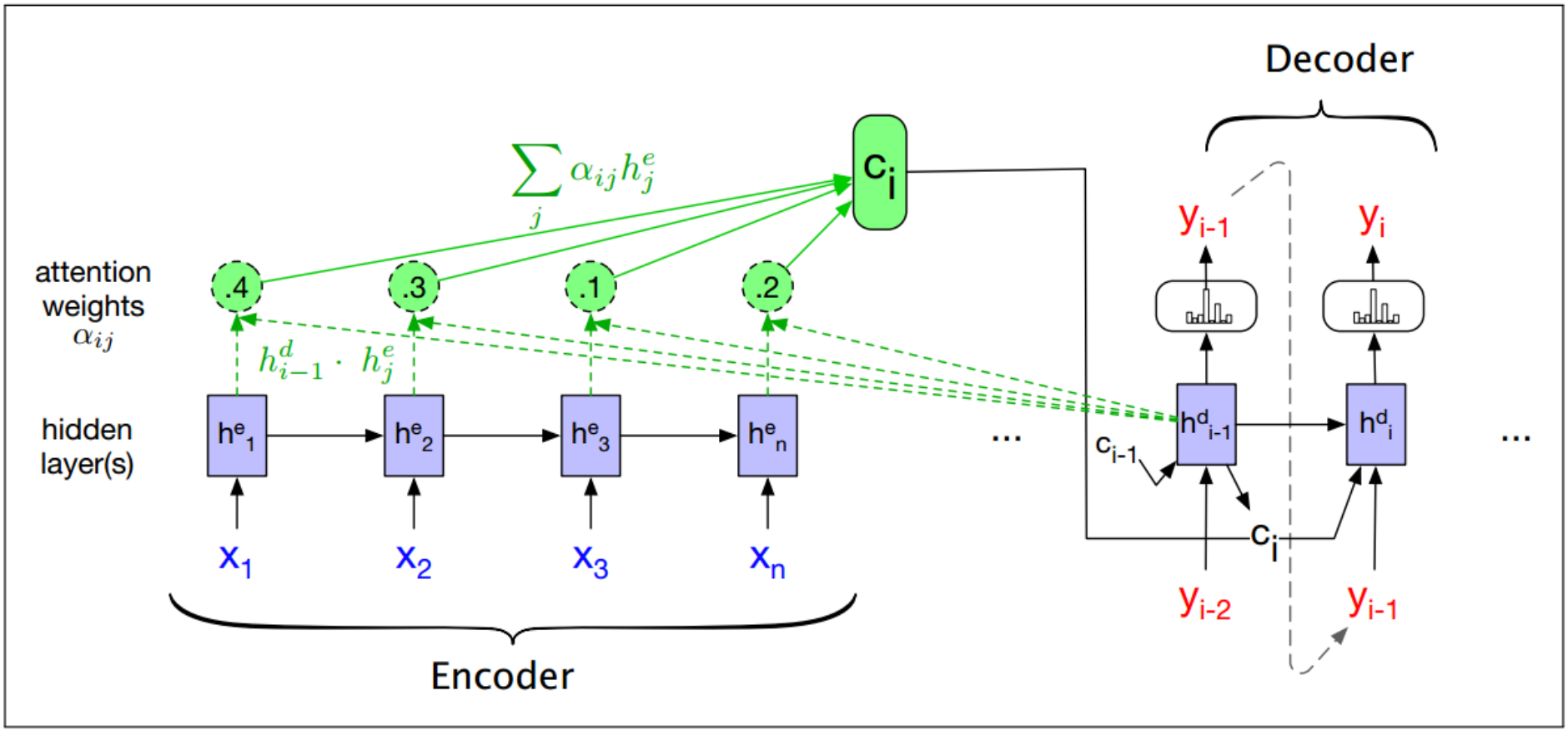


Figure 8.22 A sketch of the encoder-decoder network with attention, focusing on the computation of $\mathbf{c}_i$. The context value $\mathbf{c}_i$ is one of the inputs to the computation of $\mathbf{h}_i^d$. It is computed by taking the weighted sum of all the encoder hidden states, each weighted by their dot product with the prior decoder hidden state $\mathbf{h}_{i-1}^d$.

注意力机制扩展

参数化评分函数：除了点积注意力，Attention机制的评分函数还可以采用基于参数化的加权和，使网络能够学习在当前应用中，**解码器和编码器状态之间哪些方面的相似性对任务最为重要**。

$$\text{score}(h_d^{i-1}, h_e^j) = h_d^{i-1} W_s h_e^j$$

上式通过参数化的加权和计算解码器和编码器隐藏状态之间的相似性，W_s 是一个训练得到的权重矩阵。该扩展允许Encoder和Decoder使用不同维度的隐藏状态，而简单的点积注意力要求它们具有相同的维度。

Attention机制解决了传统Encoder-Decoder模型中存在的瓶颈问题，通过**动态生成上下文向量**，使得Decoder能够在每个时间步都关注输入序列中最相关的部分，从而显著提高了生成任务（如机器翻译）的表现。

THE END