9. 深度学习架构用于序列处理

Time will explain.

Jane Austen, Persuasion

语言是一种固有的时间现象。当我们理解和产生口语时，我们处理不确定长度的连续输入流。即使在处理书面文本时，我们通常也按顺序处理。语言的时代性反映在我们使用的隐喻中;我们谈论的是对话流、新闻源和推特流，所有这些都唤起了这样一种观念：语言是一个随时间展开的序列。

这种时间性质反映在我们用于处理语言的算法中。例如，当应用于词类标记问题时，Viterbi 算法通过一次输入一个单词来逐步实现自己的方式，从而沿途收集信息。另一方面，我们为情感分析而研究的机器学习方法和其他文本分类任务不具有这种时间性，它们假定可以同时访问其输入的所有方面。前馈神经网络尤其如此，包括它们在神经语言模型中的应用。这些完全连接的网络使用固定大小的输入以及关联的权重来一次捕获示例的所有相关方面。这使得难以处理长度可变的序列，并且无法捕获语言的重要时间方面的信息。

这些问题的解决方法是神经语言模型采用的滑动窗口方法。这些模型通过接受固定大小的符记窗口作为输入来运行。长度大于窗口大小的序列通过遍历输入进行预测来处理，最终结果是跨越输入的一系列预测。重要的是，在一个窗口中做出的决定不会影响后续的决定。从第 7 章图 7-13 复制而来的图 9.1，描述了使用这种方法的神经语言模型的操作，该方法的窗口大小为 3。在这里，根据输入“for all the”，我们将预测下一个单词。通过将窗口一次向前滑动一个单词来预测后续单词。

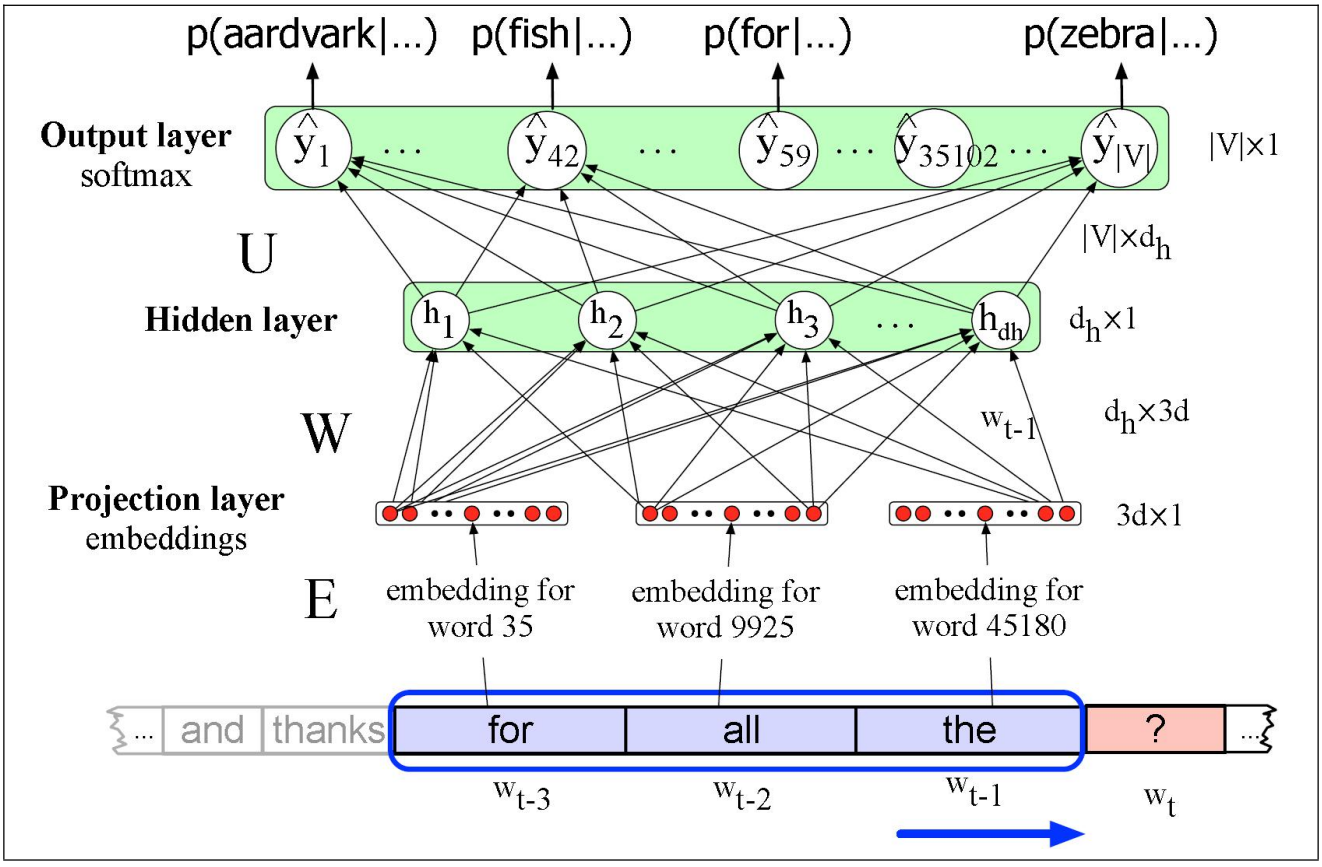

图 9-1：前馈神经语言模型的简化视图

图注：在文本中移动的前馈神经语言模型的简化视图。在每个时间步 t，网络获取 3 个上下文词，将每个上下文词转换为 d 维嵌入，并将 3 个嵌入拼接在一起，以获得网络的 $1\times N_d$ 单元输入层 x。网络的输出是词汇表上的概率分布，代表相对于每个单词为下一个可能单词的模型置信度。

由于多种原因，这种通用方法存在问题。首先，它与我们早期的马尔可夫 N-gram 方法一样，具有主要的缺点，即它限制了可以从中提取信息的上下文。上下文窗口之外的任何内容都不会影响所做出的决定。这是一个问题，因为有许多任务需要访问可能与处理发生地点(point)任意距离的信息。其次，窗口的使用使网络很难学习像成分等现象引起的系统模式。例如，在图 9.1 中，短语“all the”出现在两个单独的窗口中：首先显示为窗口中的第二和第三位置，然后再次出现在下一步中显示为第一和第二位置，从而迫使网络学习两种不同模式（这两种模式原本是一样的）。

本章涵盖了旨在解决这些挑战的两个紧密相关的深度学习架构：递归神经网络(RNN)和 Transformer 网络。两种方法都具有直接处理语言的顺序性质的机制，这些机制允许它们在不使用任何固定大小的窗口的情况下处理可变长度的输入，并捕获和利用语言的时间性质。

9.1. 语言模型回顾

在本章中，我们将主要通过概率语言模型的镜头(lens)来探讨这两种架构。回顾第三章，概率语言模型在给定某些先前上下文的情况下预测序列中的下一个单词。例如，如果前面的上下文是“Thanks for all the”，并且我们想知道下一个单词是“fish”的可能性，我们将计算：

P(fish|Thanks for all the)

语言模型让我们能够为每个可能的下一个单词分配这样的条件概率，给我们整个词汇的分布。我们也可以将这些条件概率与链式规则结合起来分配给整个序列的概率:

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

这个公式产生了广泛的序列标记应用，正如我们将看到的，它提供了一个明确的训练目标，该目标基于模型对序列中下一个单词的预测有多好。

我们已经看到了两种实例化概率语言模型的方法:第 3 章的 N-gram 模型和第 7 章的带滑动窗口的前馈神经网络。不幸的是，这两种方法都受到如下公式中马尔可夫假设的约束：

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{(n-N+1):(n-1)})$$

也就是说，预测基于一个大小为 N 的固定的先前上下文;在此之前发生的任何输入都与结果无关。我们在本章中探索的方法将会放宽这个假设，允许模型使用更大的上下文。

我们通过检查语言模型对与训练数据来自同一来源的看不见的数据进行预测的能力来评估语言模型。直观地讲，好的模型是为看不见的数据分配更高概率的模型。为了使这种直觉具体化，我们使用困惑度作为模型质量的度量。相对于看不见的测试集，模型θ的困惑度(PP)是模型分配给它的概率，并通过其长度进行归一化。

$$PP_\theta(w_{1:n}) = P(w_{1:n})^{\frac{1}{n}}$$

受信息论启发，观察困惑度的另一种方法是熵。

$$\begin{aligned} PP(w_{1:n}) &= 2^{H(w_{1:n})} \\ &= 2^{-\frac{1}{n}\sum_1^n log_2 m(w_n)} \end{aligned}$$

在这个公式中，指数中的值是我们当前模型相对于真实分布的交叉熵。

评估语言模型的另一种方法是使用它来生成新的序列。生成的序列反映训练数据的程度表明了模型的质量。我们在第 3 章中了解了如何通过采用同时由 Claude Shannon(Shannon，1951)和心理学家 George Miller 和 Selfridge(1950)提出的技术来做到这一点。首先，我们根据一个单词是否适合作为一个序列的开头对一个单词进行随机采样以开始一个序列。在对第一个单词进行采样之后，我们对以先前选择为条件的其他单词进行采样，直到达到预定长度，或者生成序列符记的结尾。今天，这种方法称为**自动回归生成(autoregressive generation)**。我们将介绍其在诸如机器翻译和文本摘要之类的问题中的实际应用。

9.2. 循环神经网络

循环神经网络(RNN)是在其网络连接中包含循环的任何网络，即任何一个单元的值直接或间接取决于其自身较早的输出作为输入的网络。尽管功能强大，但难以推理和训练。但是，在通用的循环网络类别中，有一些受约束的体系结构已被证明在应用于口语和书面语言时非常有效。在本节中，我们考虑一类称为**Elman Networks**(Elman，1990)的循环网络或**简单循环网络(simple recurrent networks)**。这些网络本身是有用的，并且是更复杂方法的基础，例如本章稍后讨论的**长短期记忆(LSTM)**网络。展望未来，当我们使用术语 RNN 时，我们将指的是这些更简单、更受约束的网络。

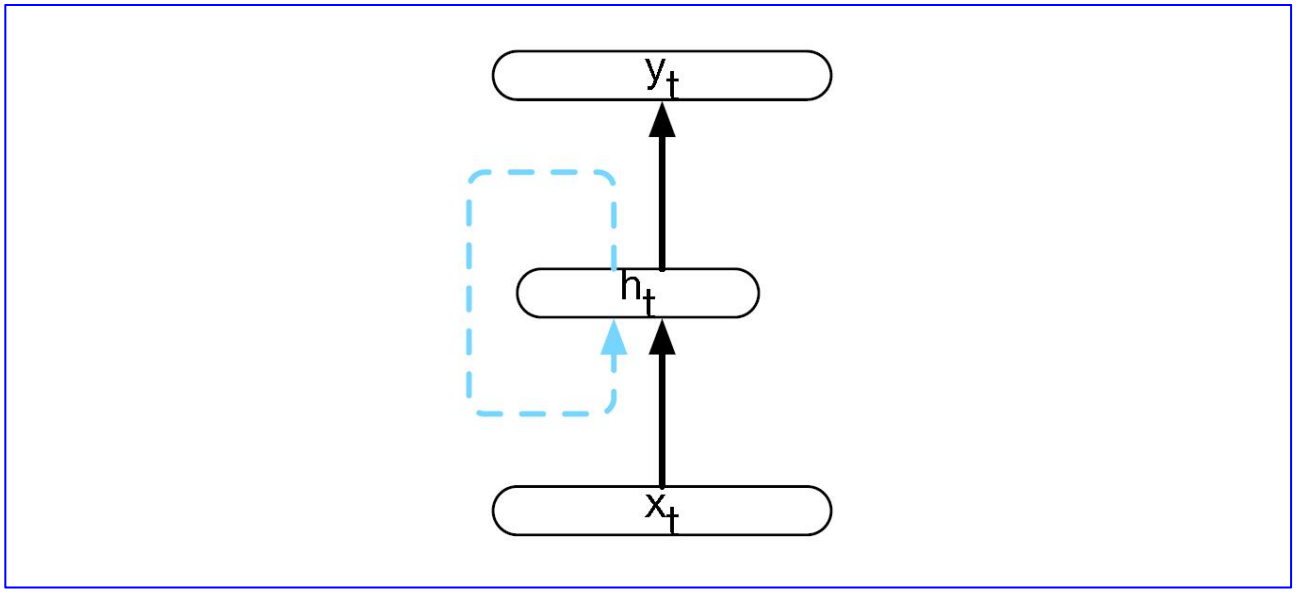

图 9-2：Elman 的简单循环神经网络

图注：Elman 的简单循环神经网络(Elman, 1990)。隐藏层包括一个循环连接，作为其输入的一部分。即隐含层的激活值既取决于当前输入，也取决于上一个时间步的隐含层激活值。

图 9.2 说明了 RNN 的结构。与普通前馈网络一样，将代表当前输入 x_t 的输入向量乘以权重矩阵，然后传递给非线性激活函数以计算隐藏单元层的值。然后将此隐藏层用于计算相应的输出 y_t。与我们以前的基于窗口的方法不同，通过一次向网络展示一项来处理序列。与前馈网络的主要区别在于图中虚线所示的循环连接。此连接将输入增加到隐藏层的计算中，该隐藏层的值“来自前一个时间点”。

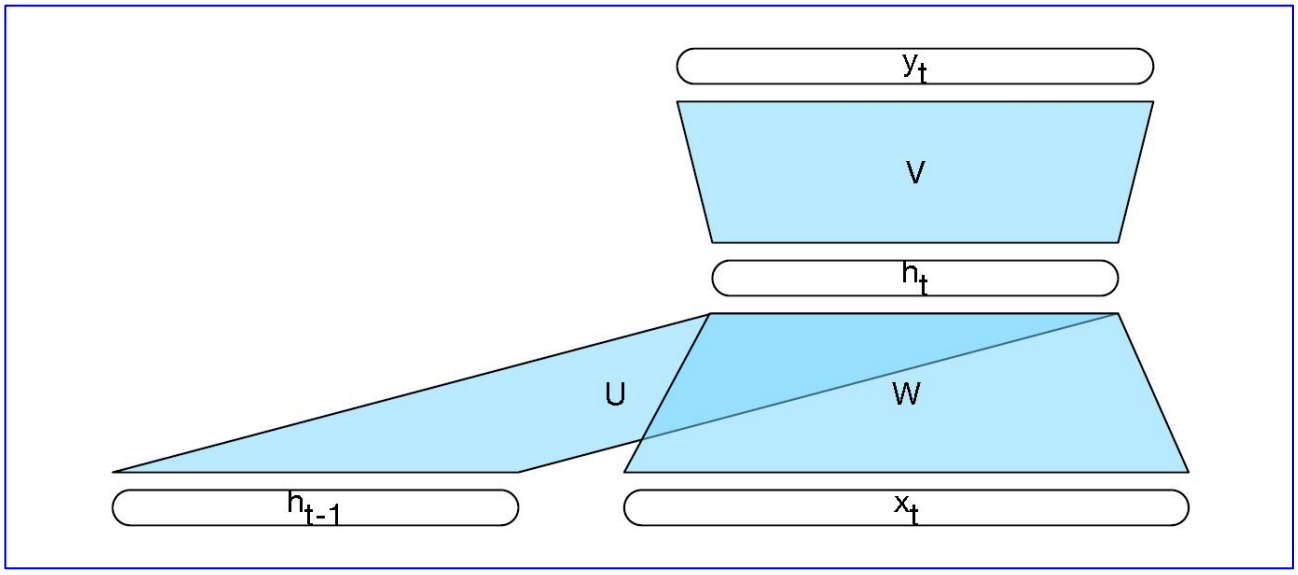

图 9-3：前馈网络的简单循环神经网络

上一个时间步中的隐藏层提供了一种记忆或上下文形式，该形式对较早的处理进行编码并通知要在较晚的时间点做出的决策。至关重要的是，这种方法不会在此先验上下文中施加固定长度的限制。前一个隐藏层中嵌入的上下文包括延伸回到序列开头的信息。

添加这个时间维度会使 RNN 看起来比非循环体系结构更复杂。但实际上，它们并没有那么不同。给定输入向量和前一个时间步的隐含层的值，我们仍然执行第七章中介绍的标准前馈计算。为了看到这一点，考虑图 9.3，它阐明了循环的性质，以及它如何在隐藏层的计算中发挥影响(factors)。最重要的变化是新的一组权值 U，它连接了从上一个时间步的隐藏层到当前的隐藏层。这些权重决定了网络如何利用过去的上下文来计算当前输入的输出。与网络中的其他权值一样，这些连接是通过反向传播进行训练的。

9.2.1. RNN 中的推理

RNN 中的前向推理(将输入序列映射到输出序列)与我们在前馈网络中看到的几乎相同。为了计算输入 x_t 对应的输出 y_t，我们需要隐藏层 h_t 的激活值。为了计算这一点，我们将输入 x_t 乘以权重矩阵 W，并将上一个时间步 h_{t-1} 的隐藏层乘以权重矩阵 U。我们将这些值加在一起，然后将它们传递给合适的激活函数 g，以得出当前隐藏层的激活值 h_t。一旦有了隐藏层的值，就可以进行通常的计算以生成输出向量。

$$h_t=g(Uh_{t-1}+Wx_t)$$
$$y_t=f(Vh_t)$$

这里值得仔细指定输入层、隐藏层和输出层的维数，以及权值矩阵，以确保这些计算是正确的。

输入层、隐藏层和输出层的尺寸分别为 d_{in}、d_h 和 d_{out}。因此，我们的三个参数矩阵是:

$$W \in \mathbb{R}^{d_h \times d_{in}}, \quad U \in \mathbb{R}^{d_h \times d_h}, \quad V \in \mathbb{R}^{d_{out} \times d_h}$$

在软分类的常见情况下，计算 y_t 由 softmax 计算组成，该计算提供了可能的输出类别上的概率分布。

$$y_t=\text{softmax}(Vh_t)$$

在 t 时刻的计算要求从 t-1 时刻开始的隐含层的值，这一事实要求一种从序列开始到结束的增量推理算法，如图 9.4 所示。通过及时展开网络，也可以看出简单循环网络的顺序性质，如图 9.5 所示。在这个图中，每个时间步都复制了不同的单元层，以说明它们随着时间的推移会有不同的值。然而，不同的权重矩阵在时间上是共享的。

```
function FORWARDRNN(x, network) returns output sequence y
h_0 ←0
for i←1 to LENGTH(x) do
    h_i←g(U h_{i-1} + W x_i)
    y_i← f(V h_i)
return y
```

图 9-4：简单循环网络中的正向推理

图注：矩阵 U，V 和 W 在时间上是共享的，而 h 和 y 的新值在每个时间步中计算。

9.2.2. 训练

与前馈网络一样，我们将使用训练集，损失函数和反向传播来获得调整这些循环网络中权重所需的梯度。如图 9.3 所示，我们现在有 3 组权重要更新：W，从输入层到隐藏层的权重；U，从前一个隐藏层到当前隐藏层的权重；V，从隐藏层到输出层的权重。

图 9.5 突出了我们在前馈网络中不需要担心的两个问题。首先，为了计算 t 时刻输出的损失函数，我们需要从 t-1 时刻开始的隐含层。第二，t 时刻的隐含层影响 t 时刻的输出和 t+1 时刻的隐含层(因此也影响 t +1 时刻的输出和损失)。由此可以得出，要评估 h_t 产生的误差，我们需要知道它对当前输出以及随后输出的影响。

针对这种情况量身定制反向传播算法会导致在 RNN 中训练权重的两遍算法。第一遍，我们执行前向推断，计算 h_t 和 y_t，在每个时间步累加损失，在每个步骤保存隐藏层的值以供下一时间步使用。第二遍，我们以相反的顺序处理序列，随着时间的推移计算所需的梯度，计算并保存误差项，以供在时间上向后的

每一步用于隐藏层。这种通用方法通常称为**时间反向传播(Backpropagation Through Time)**(Werbos 1974，Rumelhart 等人 1986，Werbos 1990)。

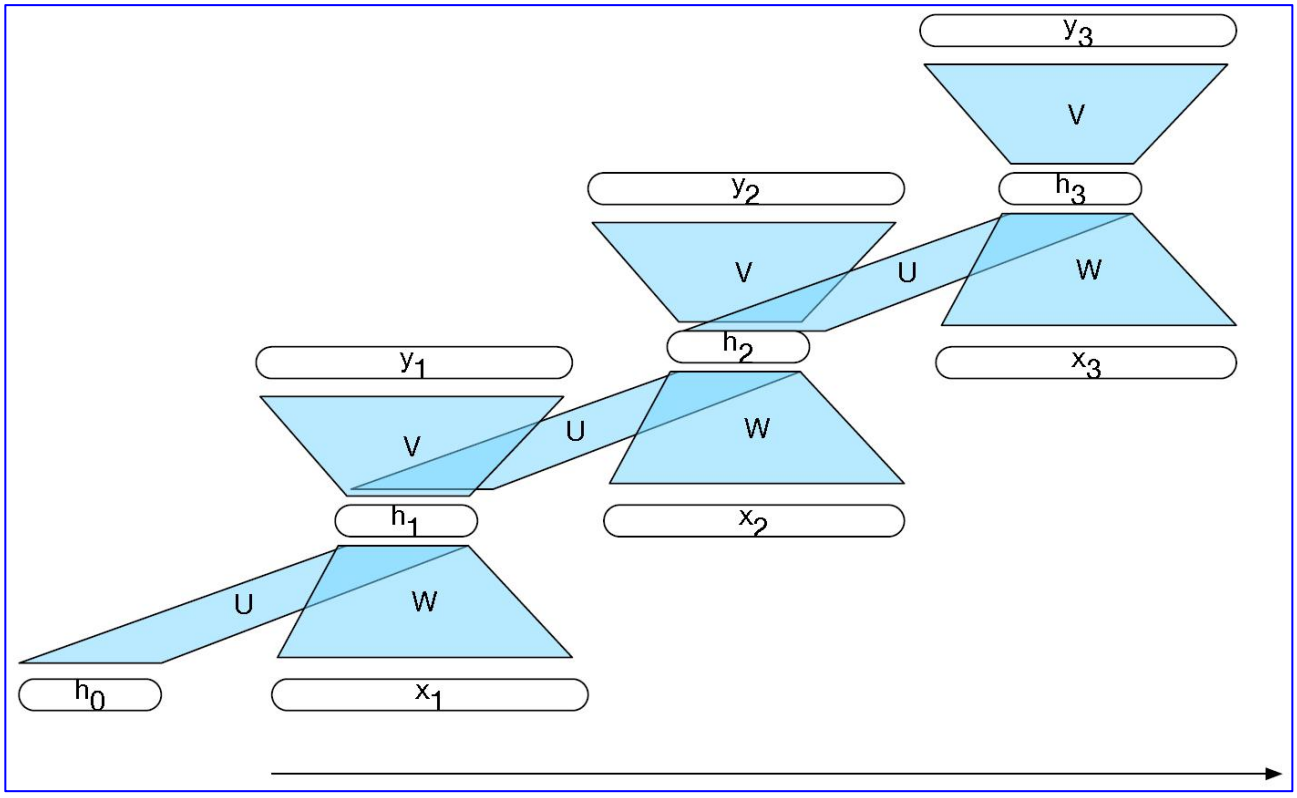

图 9-5：在时间上展开的简单循环神经网络

图注：每个时间步都复制网络层，而所有时间步都共享权值 U、V 和 W。

幸运的是，有了现代的计算框架和足够的计算资源，就不需要专门的方法来训练 RNN。如图 9.5 所示，将循环网络显式展开为前馈计算图可消除任何显式循环，从而可以直接训练网络权重。在这种方法中，我们提供了一个模板，用于指定网络的基本结构，包括输入、输出和隐藏层的所有必要参数、权重矩阵以及要使用的激活和输出函数。然后，当呈现特定的输入序列时，我们可以生成特定于该输入的展开前馈网络，并使用该图通过普通的反向传播执行前向推断或训练。

对于涉及更长输入序列的应用程序，例如语音识别，字符级处理或连续输入流，展开整个输入序列可能不可行。在这些情况下，我们可以将输入展开为可管理的固定长度段，并将每个段视为不同的训练项目。

9.2.3. RNN 作为语言模型

基于 RNN 的语言模型一次处理一个单词的序列，试图通过使用当前单词和之前的隐藏状态作为输入来预测序列中的下一个单词(Mikolov 等人，2010)。N-gram 模型中固有的有限上下文约束被避免了，因为隐藏状态包含了关于前面所有单词的信息，一直到退回到序列的开始。

循环语言模型中的前向推理完全按照第 9.2.1 节中的描述进行。输入序列 x 由单词嵌入组成，这些单词嵌入表示为大小为|V|×1 的独热向量。并且输出预测 y 表示为向量，表示在词汇表上的概率分布。在每个步骤中，模型都使用单词嵌入矩阵 E 检索当前单词的嵌入，然后将其与上一步中的隐藏层进行组合以计算新的隐藏层。然后，该隐藏层用于生成输出层，该输出层将穿过 softmax 层以在整个词汇表上生成概率分布。也就是说，在时间 t：

$$e_t = E^T x_t$$
$$h_t = g(Uh_{t-1}+We_t)$$
$$y_t = \text{softmax}(Vh_t)$$

给定 h 中提供的证据，可以将 Vh 产生的向量视为词汇量的一组分数。将这些分数传递给 softmax 会将分数归一化为概率分布。给定 y，则词汇中特定单词 i 的概率(即下一个单词)就是 y 的相应分量。

$$P(w_{t+1} = i \mid w_{1:t}) = y_t^i$$

由此可以得出，整个序列的概率就是序列中每一项的概率的乘积。

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

为了将 RNN 作为一种语言模型进行训练，我们使用一个文本语料库作为训练材料，并结合名为**教师强制(teacher forcing)**的训练方案。使用交叉熵作为损失函数，其任务是最小化预测训练序列中下一个单词的误差。回想一下，交叉熵损失度量的是预测概率分布与正确分布之间的差异。

$$L_{CE} = -\sum_{w \in V} y_w^t \log \hat{y}_w^t$$

在语言建模的情况下，正确的分布 y 来自于知道下一个单词。这表示为对应词汇表的一个独热向量，其中实际下一个单词的条目为 1，所有其他条目为 0。因此，语言建模的交叉熵损失是由模型分配给正确的下一个单词的概率决定的。具体来说，在时刻 t, CE 损失是分配给训练序列中下一个单词的负对数概率。

$$L_{CE}(\hat{y}_t, y_t) = -\log \hat{y}_{w_{t+1}}^t \quad (9.1)$$

在实践中，通过梯度下降调整网络中的权值，使训练序列的平均 CE 损失最小。图 9.6 说明了这个训练方案。

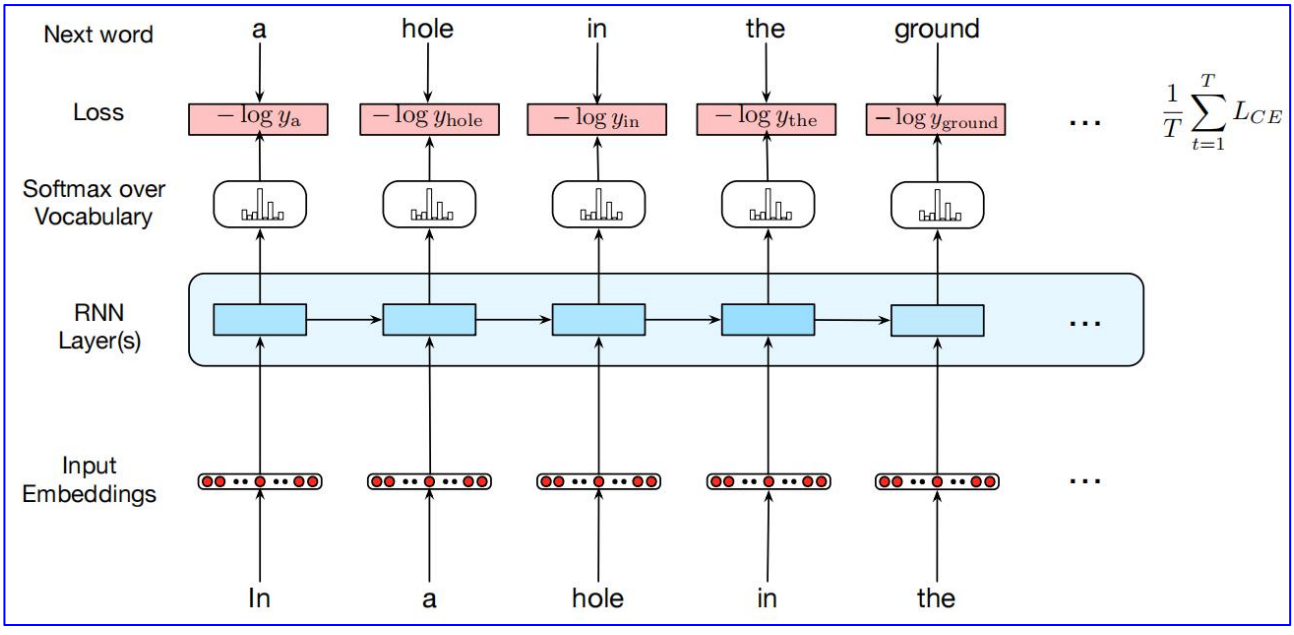

图 9-6：训练 RNN 作为语言模型

细心的读者可能已经注意到，输入的嵌入矩阵 E 和最后一层矩阵 V(V 的输出馈入 softmax 函数)非常相似。E 的行表示训练过程中学习到的词汇中每个单词的嵌入度，目的是让含义和功能相似的单词具有相似的嵌入度。由于这些嵌入的长度对应于隐藏层 d_h 的大小，因此嵌入矩阵形状 E 为$|V| \times d_h$。

最终层矩阵 V 提供了一种对词汇表中每个单词的似然度进行评分的方法，该方法通过 V_h 的计算，给出存在于网络最终隐藏层中的证据。这要求它也具有维度$|V| \times d_h$。也就是说，V 的行提供了第二组学习的单词嵌入，这些单词嵌入捕获了单词含义和功能的相关方面。这就引出了一个明显的问题：两者是否都需要？**权重绑定(Weight Tying)**是一种方法，它免除了此冗余，并在输入层和 softmax 层使用一组嵌入。也就是说，E=V。为了做到这一点，我们将最终隐藏层的维数设置为相同的 d_h，(或添加一个额外的投影层以执行相同的操作)，并简单地对两个层使用相同的矩阵。除了提供改善的困惑度结果之外，这种方法还大大减少了模型所需的参数数量。

基于 RNN 的语言模型生成

就像第 3 章中的概率莎士比亚生成器一样，了解语言模型的一个有用方法是使用训练过的模型来生成随机的新句子。这个过程和第 3.3 节描述的基本一样。

•首先，从 softmax 分布的输出中选取一个单词作为第一个输入，这是使用句子标记开头<s>作为第一个输入的结果。

•使用第一个单词的词嵌入作为下一个时间步的网络输入，然后以同样的方式采样下一个单词。

•继续生成，直到采样到句子标记</s>或达到固定长度限制为止。

这种技术被称为自回归生成，因为每一步生成的词都是以网络在前一步中选择的词为条件的。图 9.7 说明了这种方法。在这个图中，RNN 的隐藏层和循环连接的细节隐藏在蓝色块中。

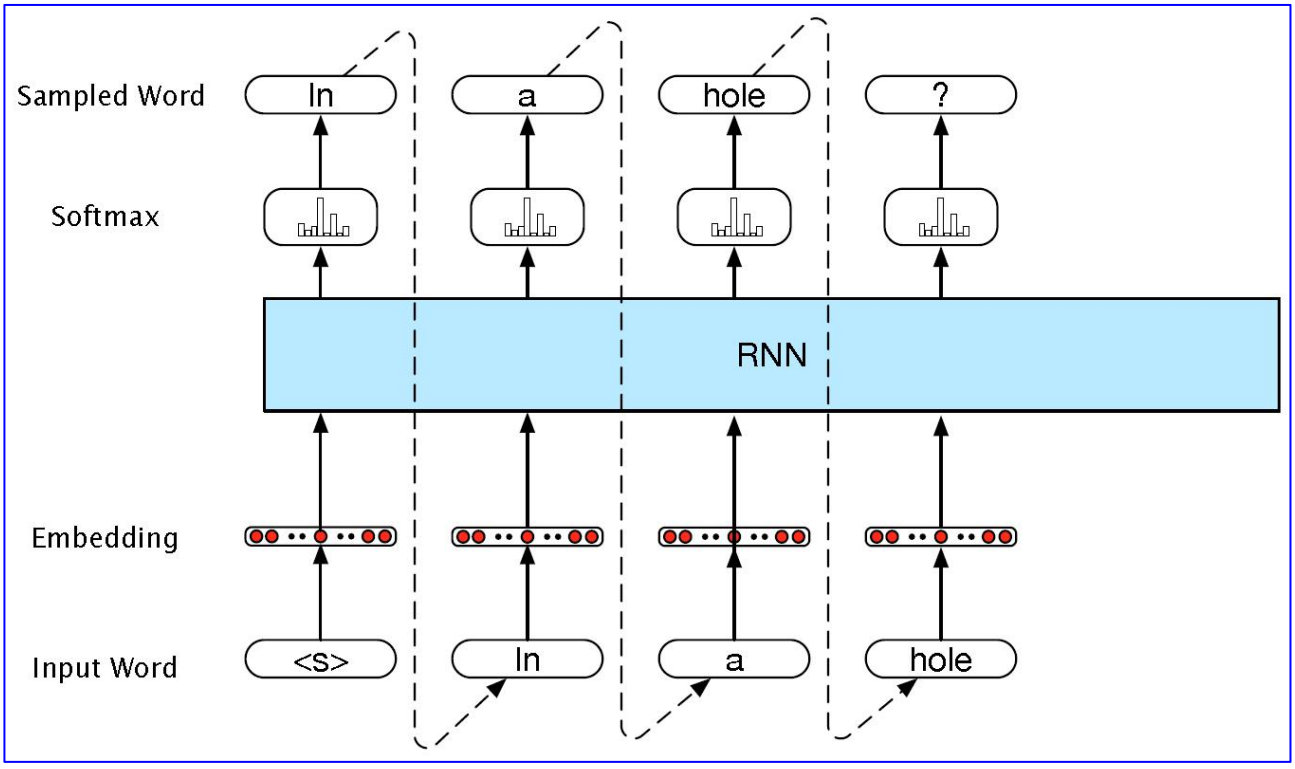

图 9-7：基于 RNN 的神经语言模型自回归生成

虽然这是一个有趣的练习，但该体系结构启发了最新的应用方法，例如机器翻译，摘要和问答。这些方法的关键是在适当的上下文中填充生成组件。也就是说，除了简单地使用<s>来开始工作之外，我们还可以提供更丰富的适合任务的上下文。在第 9.4 节中，我们将在基于 Transformer 的语言模型的背景下讨论上下文生成在摘要问题上的应用。

9.2.4. RNN 的其它应用

事实证明，循环神经网络是一种有效的方法，可用于语言建模，序列标签任务(例如词类标记)以及序列分类任务(例如情感分析和主题分类)。正如我们将在第 11 章中看到的那样，它们构成了摘要，机器翻译和问答的逐个序列方法的基础。

序列标签

在序列标签中，网络的任务是将一个从固定的小型标签集中选择的标签分配给序列的每个元素。序列标签的规范示例包括词类标记和命名实体识别，已在第 8 章中详细讨论了。在 RNN 序列标签方法中，输入是词嵌入，而输出是由给定标记集上的 softmax 层生成的标记概率 ，如图 9.8 所示。

在该图中，每个时间步的输入都是与输入符记相对应的预训练词嵌入。RNN 块是一个抽象概念，代表展开的简单循环网络，该网络由每个时间步的输入层，隐藏层和输出层以及组成该网络的共享 U，V 和 W 权重矩阵组成。网络在每个时间步的输出表示为一个分布，该分布置于 softmax 层生成的 POS 标签集上。

为了为给定的输入生成标签序列，我们对输入序列进行前向推理，并在每个步骤中从 softmax 中选择最可能的标记。由于我们在每个时间步都使用 softmax 层在输出标签集上生成概率分布，因此我们将在训

练过程中再次采用交叉熵损失。

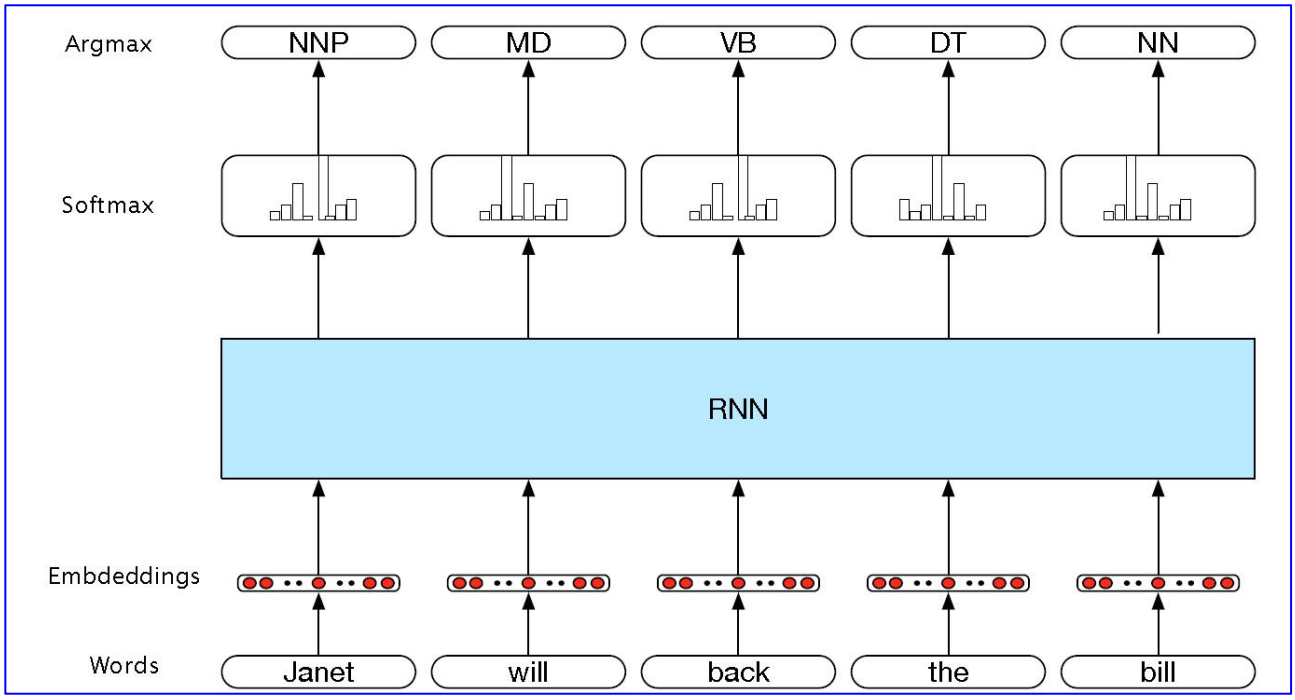

图 9-8：使用简单的 RNN 将词类标记当作序列标签

图注：预先训练的单词嵌入用作输入，并且 softmax 层在每个时间步上输出的词类标记上都提供了概率分布。

9.2.5. RNN 用于序列分类

RNN 的另一个用途是对整个序列进行分类，而不是对其中的符记进行分类。在第 4 章中，我们已经对情感分析进行了讨论。其他示例包括文档级主题分类，垃圾邮件检测，针对客户服务应用程序的邮件路由以及欺骗检测。在所有这些应用中，文本序列被分类为属于少数类别之一。

要在此设置中应用 RNN，将要分类的文本一次通过一个单词传递到 RNN，并在每个时间步生成一个新的隐藏层。文本的最后元素 h_n 的隐藏层被用来构成整个序列的压缩表示。在最简单的分类方法中，h_n 用作后续前馈网络的输入，该网络通过 softmax 在可能的类别上选择类别。图 9.9 说明了这种方法。

注意，在这种方法中，最后一个元素之前的序列中的单词没有中间输出。因此，没有与这些元素相关的损失项。取而代之的是，用于训练网络中权重的损失函数完全基于最终的文本分类任务。具体来说，softmax 的输出和前馈分类器的输出，连同交叉熵损失一起，驱动了训练的进行。

分类的误差信号通过前馈分类器中的权值一直反向传播到其输入，然后再反向传播到前面第 9.2.2 节中描述的 RNN 中的三组权值。这种简单循环网络与前馈分类器的组合是我们深度神经网络的第一个例子。使用来自下游应用程序的损失在整个网络中调整权值的训练方案称为**端到端训练(end-to-end training)**。

9.2.6. 堆叠和双向 RNN

从图 9.9 所示的序列分类体系结构可以看出，循环网络是非常灵活的。通过将展开的计算图的前馈性质与作为公共输入和输出的向量相结合，可以将复杂网络视为可以以创造性方式组合的模块。本节介绍在 RNNs 语言处理中使用的两种更常见的网络架构。

堆叠 RNNs

到目前为止，在我们的示例中，RNN 的输入由单词或字符嵌入(矢量)的序列组成，而输出则是可用于预测单词、标记或序列标签的矢量。但是，没有什么可以阻止我们将一个 RNN 的整个输出序列用作到另一个 RNN 的输入序列。**堆叠(Stacked)RNN** 由多个网络组成，其中一层的输出充当下一层的输入，如图 9.10 所示。

它已经在许多任务中得到了证明，堆叠 RNN 可以比单层网络的性能更好。这一成功的原因之一与网络在不同的抽象层上诱导(induce)表示的能力有关。就像早期的人类视觉系统检测边缘,然后用于发现更大的区域和形状,多层网络的初始层可以诱导表示,作为进一步有用的抽象层,表示可能在一个 RNN 难以诱导。

堆栈的最优数量是针对每个应用和每个训练集而定的。但随着堆栈数量的增加，训练成本迅速上升。

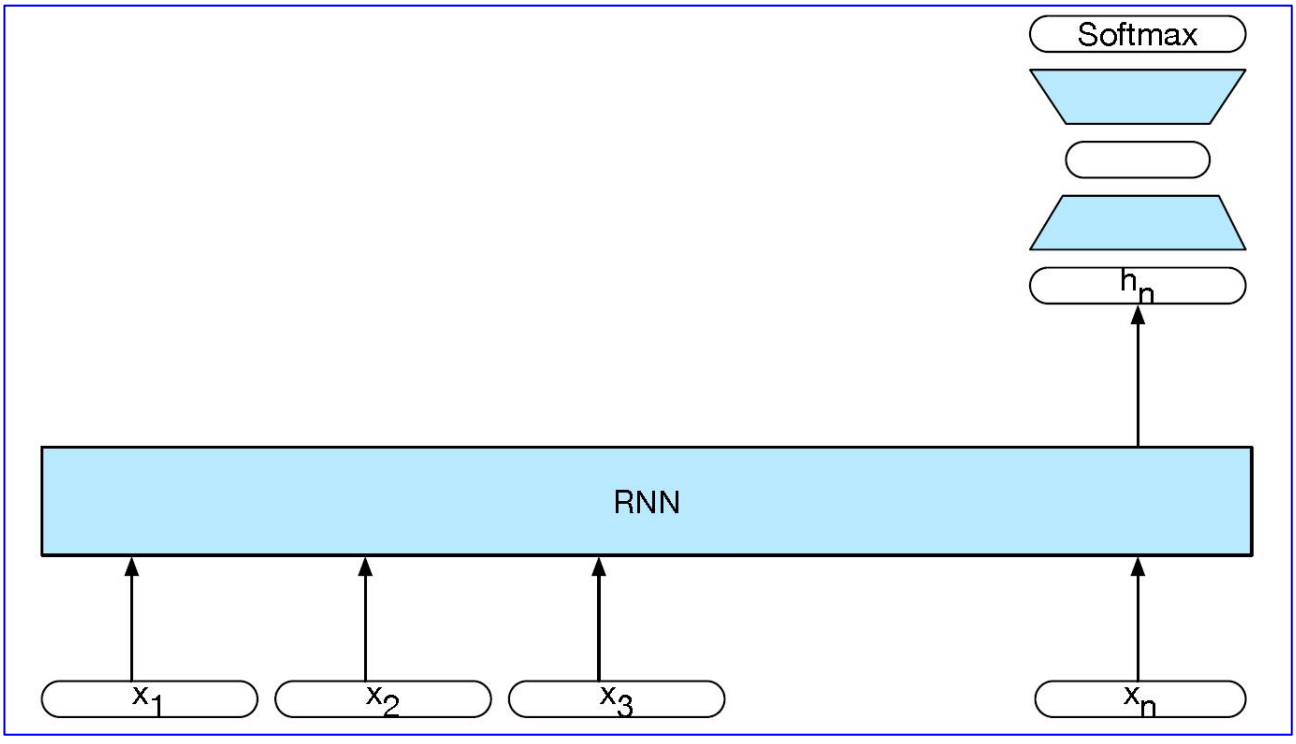

图 9-9：使用简单的 RNN 和前馈网络进行序列分类

图注：来自 RNN 的最终隐藏状态用作执行分类的前馈网络的输入。

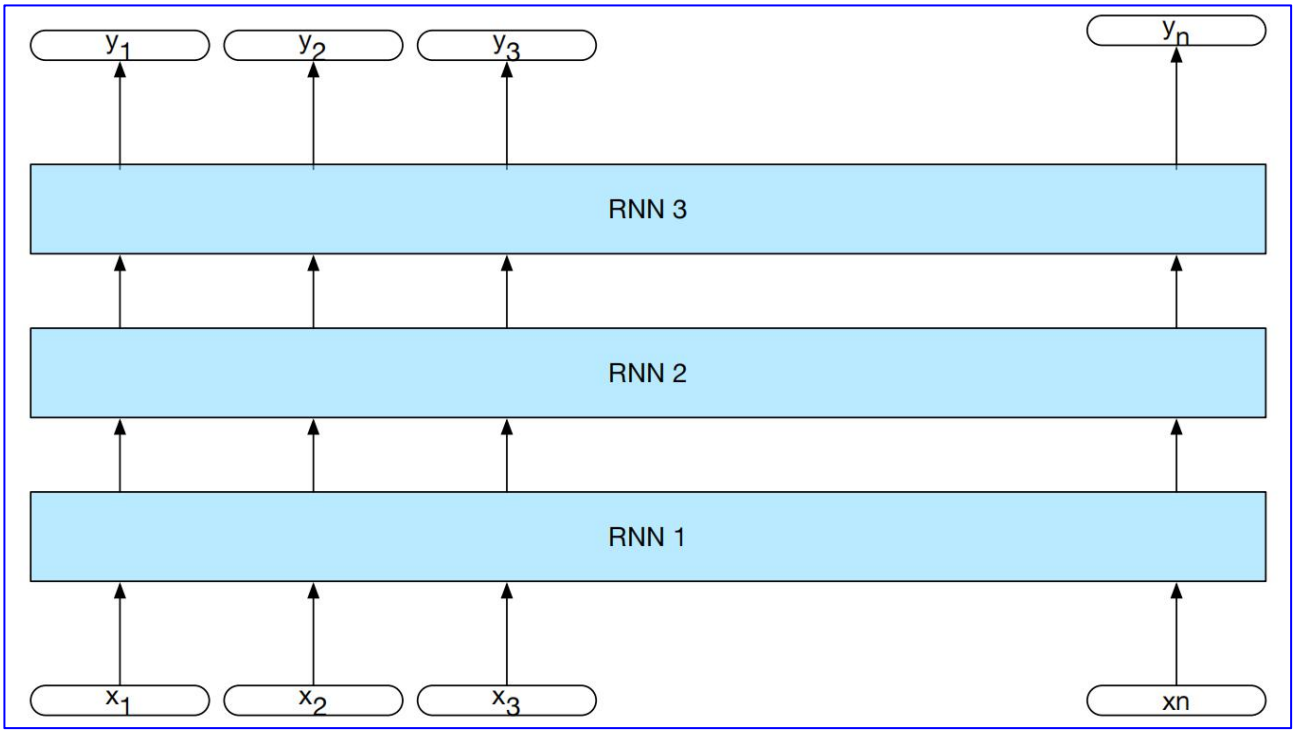

图 9-10：堆叠式 RNN

图注：较低一级的输出作为较高一级的输入，最后一个网络的输出作为最终输出。

双向 RNNs

在简单的循环网络中，给定时间 t 的隐藏状态展现网络在序列中直到该点为止所知道的一切。即，在时间 t 的隐藏状态是从开始到时间 t 的输入的函数的结果。我们可以将其视为当前时间左侧的网络上下文。

$$h_t^f = \text{RNN}_{\text{forward}}(x_1^t)$$

其中h_t^f对应于 t 时刻的正常隐藏状态，表示网络从序列到该点收集到的所有信息。

在许多应用程序中，我们可以一次访问整个输入序列。我们可能会问，利用当前输入右边的上下文是否有帮助。恢复此类信息的一种方法是，使用与我们一直在讨论的完全相同的网络，在输入序列上反向训练 RNN。通过这种方法，时间 t 的隐藏状态现在表示有关当前输入右侧的序列的信息。

$$h_t^b = \text{RNN}_{\text{backward}}(x_t^n)$$

这里，隐藏状态h_t^b表示我们已经识别的从 t 到序列末尾的所有信息。

将正向网络和反向网络相结合产生了**双向 RNN**(Schuster 和 Paliwal, 1997)。一个双向 RNN 由两个独立的 RNN 组成，一个从开始到结束处理输入，另一个从结束到开始处理输入。然后，我们将两个网络的输出组合成一个单独的表示形式，在每个时间点捕捉输入的左上下文和右上下文。

$$h_t = h_t^f \oplus h_t^b$$

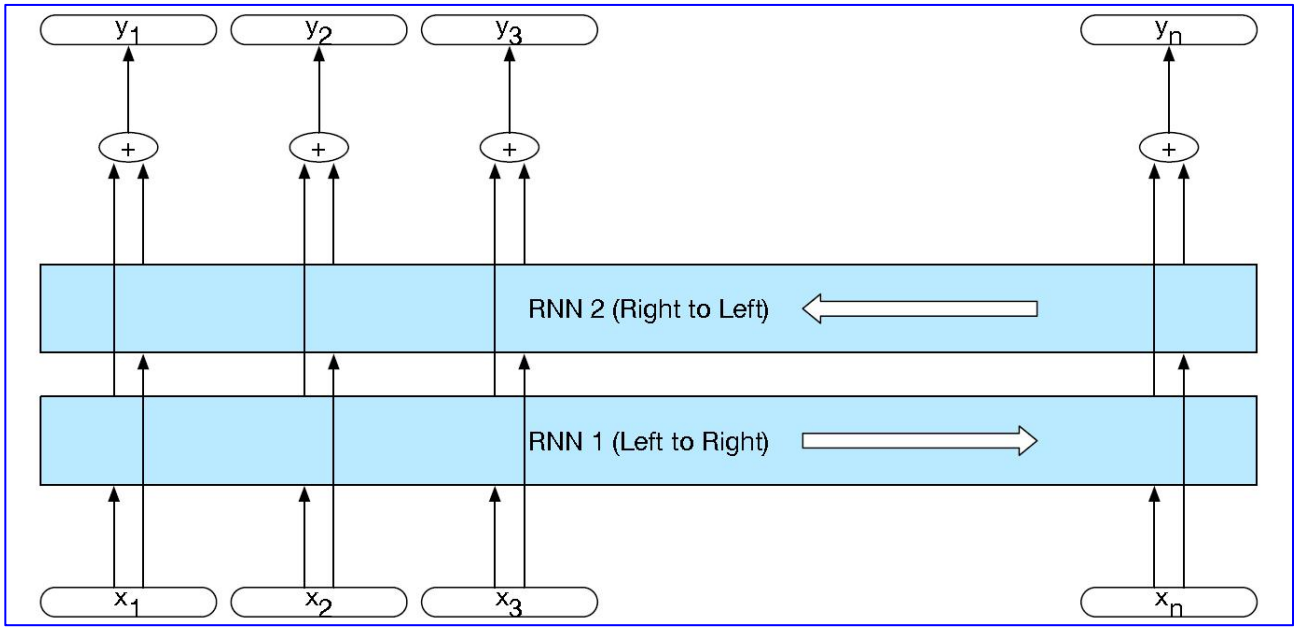

图 9-11：一个双向 RNN

图注：分别的模型在前进和后退方向进行训练，每个模型在每个时间点的输出拼接起来，以表示该时间点的事件状态。包裹在前向和后向网络周围的方框强调了这种架构的模块化本质。

图 9.11 说明了一个双向网络，其中前向和后向通过的输出是拼接在一起的。组合前向和后向上下文的其他简单方法包括逐元素加法或乘法。因此，每个时间步的输出都会在当前输入的左侧和右侧捕获信息。在序列标签应用中，这些拼接的输出可以用作本地标签决策的基础。

双向 RNN 也被证明对序列分类非常有效。回顾图 9.10，对于序列分类，我们使用 RNN 的最终隐藏状态作为后续前馈分类器的输入。这种方法的困难在于，最终状态自然会反映出比句子开头更多的有关句子结尾的信息。双向 RNN 为这个问题提供了一个简单的解决方案。如图 9.12 所示，我们仅将前进和后退路径中的最终隐藏状态组合在一起，并将其用作后续处理的输入。同样，拼接是组合两个输出的常用方法，但也使用逐元素求和、相乘或求平均。

9.3.在 RNN 中管理上下文：LSTM 和 GRU

在实践中，有一个非常困难的训练 RNN 的任务：要求网络利用远离当前处理点的信息。尽管可以访问整个前面的序列，但是，被编码在隐藏状态里的信息，往往是相当局部的，而且与输入序列的最新部分和最新决策更相关。然而，通常情况下，遥远的信息对于许多语言应用程序都是至关重要的。为此，请在

语言建模的上下文中考虑以下示例。

(9.2) The flights the airline was cancelling were full.

分配一个高概率给“airline”后面的“was”是很直截了当的，因为“airline”为语法上的单数一致性提供了强大的局部上下文。但是，很难给“were”分配适当的概率，这不仅是因为复数“flights”距离很远，而且还因为介入的上下文涉及单数成分。理想情况下，网络应该能够保留有关复数“flights”的遥远信息，直到需要它为止，同时仍能正确处理序列的中间部分。

RNN 无法传递关键信息的原因之一是隐藏层和权重(权重决定了隐藏层中的值)，它们同时被要求执行两个任务：提供对当前决策有用的信息，以及更新和转发未来决策所需的信息。

训练 SRN 的第二个困难来自需要反向传播误差信号。从 9.2.2 节回想起，时间 t 的隐藏层会参与下一计算步，因此会导致下一步骤的损失。结果，在训练的后退过程中，隐藏层要经受重复乘法，这由序列的长度确定。该过程的一个常见结果是，梯度最终被驱动为零-所谓的**消失梯度(vanishing gradients)**问题。

为了解决这些问题，已经设计了更复杂的网络体系结构来显式管理随时间推移维护相关上下文的任务。更具体地说，网络需要学会忘记不再需要的信息，并记住仍然需要做出决策的信息。

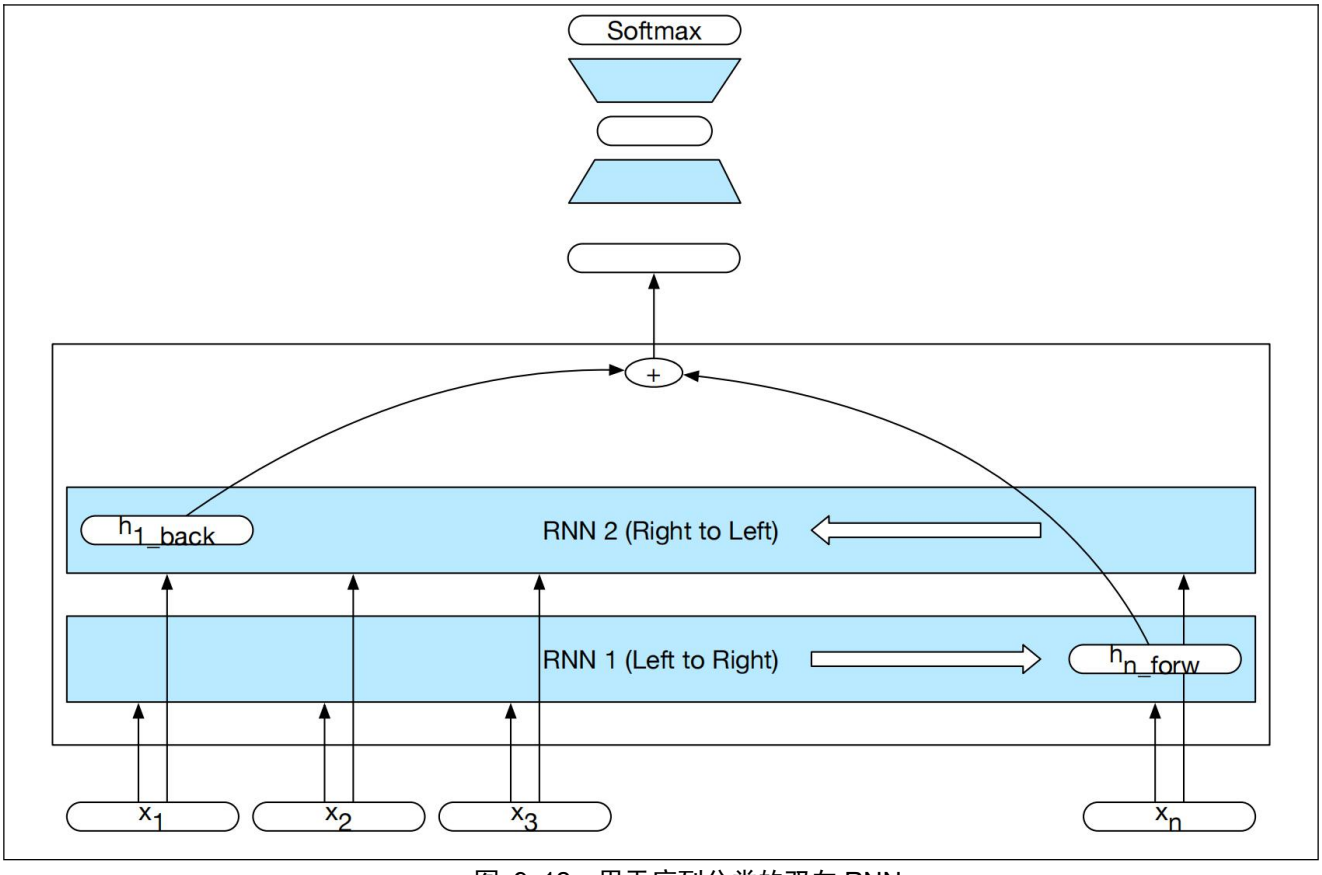

图 9-12：用于序列分类的双向 RNN

图注：向前和向后遍历的最终隐藏单元被组合以代表整个序列。该组合表示用作后续分类器的输入。

9.3.1. 长短期记忆(LSTM)

长短期记忆(LSTM)网络(Hochreiter 和 Schmidhuber，1997 年)将上下文管理问题分为两个子问题：从上下文中删除不再需要的信息，以及添加以后进行决策时可能需要的信息。解决这两个问题的关键是学习如何管理此上下文，而不是将策略硬编码到体系结构中。LSTM 通过首先在体系结构中添加一个显式的上下文层(除了通常的循环隐藏层之外)，并通过使用专用的神经单元，这些单元利用**门(gate)**来控制信息流入和流出，这些单元组成了网络层。这些门是通过使用其他权重实现的，这些权重依次对输入、先前的隐藏层和先前的上下文层进行操作。

LSTM 中的门具有相同的设计模式。每一个都由前馈层组成，其后是 S 型激活函数，然后是与该选通的层进行逐点乘法。选择 S 型作为激活函数是因为它倾向于将其输出推到 0 或 1。将其与逐点乘法相结合

具有类似于二进制**掩码(mask)**的效果。一些值几乎保持不变(即=1)，这些值是指：在选通的层中，与掩码中近似 1 的值对齐的那些值；另一些值基本被抹去(即=0)，这些值是指：与较低值对应的那些值。

我们要考虑的第一个门是**遗忘门(forget gate)**。这个门的目的是从上下文中删除不再需要的信息。遗忘门计算先前状态的隐藏层和当前输入的加权和，并通过一个 sigmoid。然后将此掩码与上下文向量相乘，以从上下文中删除不再需要的信息。

$$f_t = \sigma(U_f\, h_{t-1} + W_f\, x_t)$$

$$k_t = c_{t-1} \odot f_t$$

下一个任务是计算我们需要从之前的隐藏状态和当前输入中提取的实际信息——这与我们在所有循环网络中使用的基本计算方法相同。

$$g_t = tanh(U_g\, h_{t-1} + W_g\, x_t) \tag{9.3}$$

接下来，我们为**加法门(add gate)**生成掩码，以选择要添加到当前上下文的信息。

$$i_t = \sigma(U_i\, h_{t-1} + W_i\, x_t) \tag{9.4}$$

$$j_t = g_t \odot i_t \tag{9.5}$$

接下来，我们将它添加到修改后的上下文向量中，以获得新的上下文向量。

$$c_t = j_t + k_t \tag{9.6}$$

我们将使用的最后一个门是**输出门(output gate)**，它用于决定当前隐藏状态需要什么信息(而不是为未来的决策需要保留什么信息)。

$$o_t = \sigma(U_o\, h_{t-1} + W_o\, x_t) \tag{9.7}$$

$$h_t = o_t \odot tanh(c_t) \tag{9.8}$$

$$\tag{9.9}$$

图 9.13 给出了单个 LSTM 单元的完整计算。给定各种门的适当权值，LSTM 接受上下文层、前一时间步的隐藏层以及当前输入向量作为输入。然后它生成更新的上下文和隐藏向量作为输出。隐藏层 h_t 可以作为堆叠 RNN 中后续层的输入，也可以为网络的最后一层生成输出。

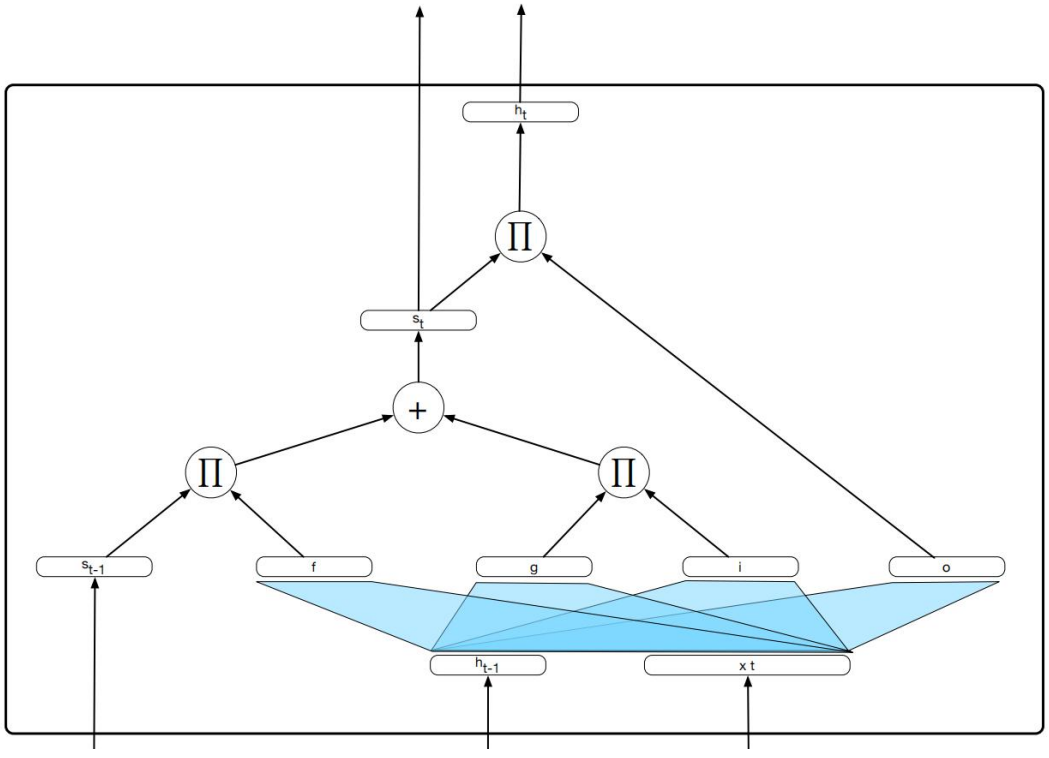

图 9-13：单个 LSTM 单元的计算图

图注：每个单元的输入由当前输入 x_t、之前的隐藏状态 h_{t-1} 和之前的上下文 c_{t-1} 组成。输出是一个新的隐藏状态 h_t 和一个更新的上下文 c_t。

9.3.2. 门控循环单元(GRU)

LSTMs 为我们的循环网络引入了相当多的附加参数。现在我们有 8 组权重需要学习(即每个单元中的 4 个门的 U 和 W)，而在简单的回归单元中我们只有 2 个。这些额外的参数增加了训练成本。门控循环单元(GRUs)(Cho 等人，2014)通过免除使用单独的上下文向量，并通过将门的数量减少到 2 个(重置门 r 和更新门 z)来减轻这一负担。

$$r_t = \sigma(U_r\, h_{t-1} + W_r\, x_t) \tag{9.10}$$

$$z_t = \sigma(U_z\, h_{t-1} + W_z\, x_t) \tag{9.11}$$

与 LSTM 一样，在这些门的设计中使用 S 形，产生的结果类似二进制掩码值。若掩码值≈0，则阻止；若掩码值≈1，则放行。重置门的目的是确定先前隐藏状态的哪些方面与当前上下文相关，以及哪些内容可以忽略。这是通过将 r 与先前的隐藏状态的值进行逐元素乘法来实现的。然后，我们在计算时间 t 的新隐藏状态的中间表示时使用此掩码值。

$$\tilde{h}_t = tanh(U(r_t \odot h_{t-1}) + W\, x_t) \tag{9.12}$$

更新门 z 的工作是确定将在新的隐藏状态下直接使用此新状态的哪些方面，以及需要保留先前状态的哪些方面以备将来使用。这可以通过使用 z 中的值在旧的隐藏状态和新的隐藏状态之间进行插值来实现。

$$h_t = (1-z_t)h_{t-1} + z_t\tilde{h}_t \tag{9.13}$$

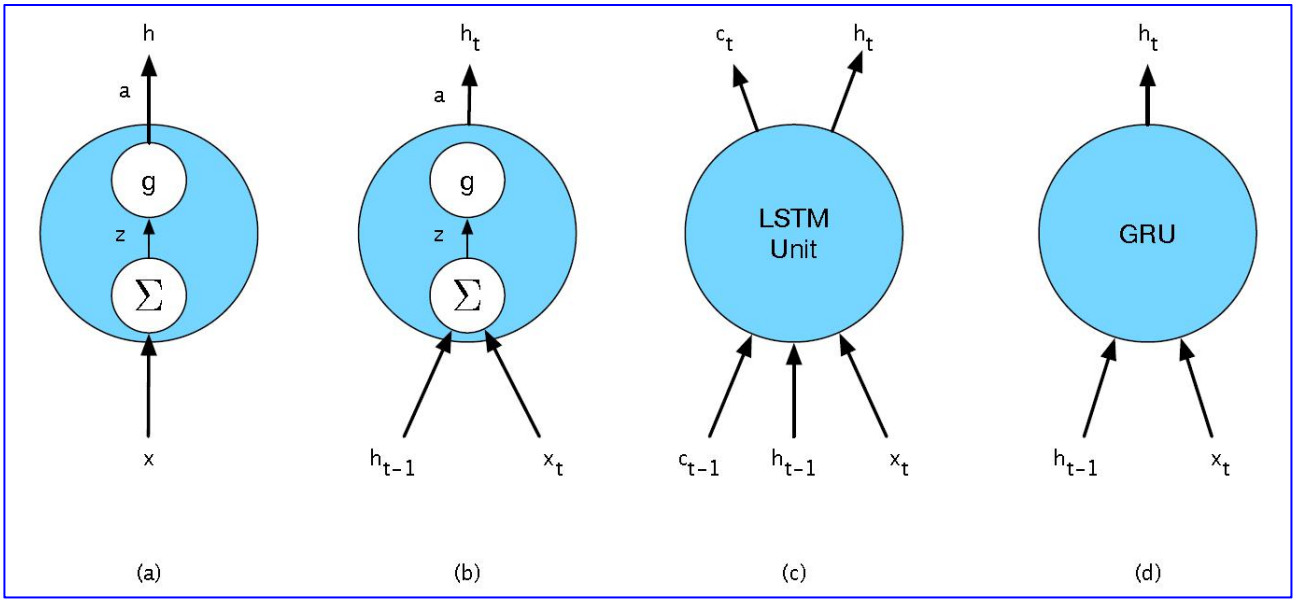

图 9-14：FF、SRN、LSTM 和 GRU

图注：(a)前馈(FF)、(b)简单循环网络(SRN)、(c)长短期记忆(LSTM)和(d)门循环单元(GRU)中使用的基本神经单元。

9.3.3. 门控单元、层和网络

LSTMs 和 GRUs 中使用的神经单元明显比基本的前馈网络中使用的神经单元复杂得多。幸运的是，这种复杂性被封装在基本处理单元中，允许我们保持模块化，并轻松地试验不同的架构。要了解这一点，考虑图 9.14，它说明了与每种单元相关的输入和输出。

最左边的(a)是基本前馈单元，由一组权值和一个激活函数决定其输出，在层中排列时，层内单元之间没有连接。(b)表示简单循环网络中的单元。现在有两个输入和一组额外的权重。然而，仍然有一个单独的激活函数和输出。右侧的 LSTM(c)和 GRU(d)单元增加了复杂性，被封装在这些单元本身中。在基本循环单元(b)上，LSTM 唯一的附加外部复杂性是附加上下文向量作为输入和输出的存在。GRU 单元具有与简单循环单元相同的输入和输出架构。

这种模块化是 LSTM 和 GRU 单元功能强大和广泛应用的关键。LSTM 和 GRU 单元可以替换为第 9.2.6 节中描述的任何网络体系结构。而且，与简单的 RNN 一样，可以将利用门控单元的多层网络展开为深度前馈网络，并通过反向传播以常规方式对其进行训练。

9.4. Self-Attention 网络：Transformer

尽管 LSTM 能够缓解由于 RNN 的循环而导致的远程信息丢失的问题，但潜在的问题仍然存在。通过扩展的一系列循环连接向前传递信息会导致相关信息的丢失和训练上的困难。此外，循环网络的固有顺序性质抑制了并行计算资源的使用。这些考虑导致了 **Transformers** 的发展：一种序列处理方法，该方法消除了循环连接，并返回到使人联想起第 7 章中所述的完全连接网络的体系结构。

Transformers 将输入向量($x_1, \ldots, x_n$)的序列映射到相同长度的输出向量($y_1, \ldots, y_n$)的序列。Transformers 由网络层的堆栈组成，这些网络层由简单的线性层、前馈网络以及围绕它们的自定义连接组成。除了这些标准组件之外，Transformers 的关键创新是使用**自注意力(self-attention)**层。我们将首先描述自注意力的工作原理，然后再回到如何适应更大的 Transformer 模块。自注意力允许网络直接从任意大小的上下文中提取和使用信息，而无需像 RNN 中那样通过中间的循环连接传递信息。在本章中，我们将重点关注自注意力在语言建模和自回归生成问题上的应用，这些问题要使用过去的上下文。在后面的章节中，我们将返回自注意力和 Transformer 的更广泛应用。

图 9.15 说明了单个因果关系或向后看的自注意力层中的信息流。与整个 Transformer 一样，自注意力层将输入序列(x_1，…，x_n)映射到相同长度(y_1，…，y_n)的输出序列。在处理输入中的每个项(item)时，模型可以访问所有输入(包括正在考虑的输入)，但是无法访问有关当前输入之外的输入信息。此外，为每个项执行的计算独立于所有其他计算。第一点确保我们可以使用这种方法创建语言模型，并将它们用于自回归生成；第二点意味着我们可以轻松地并行化此类模型的正向推理和训练。

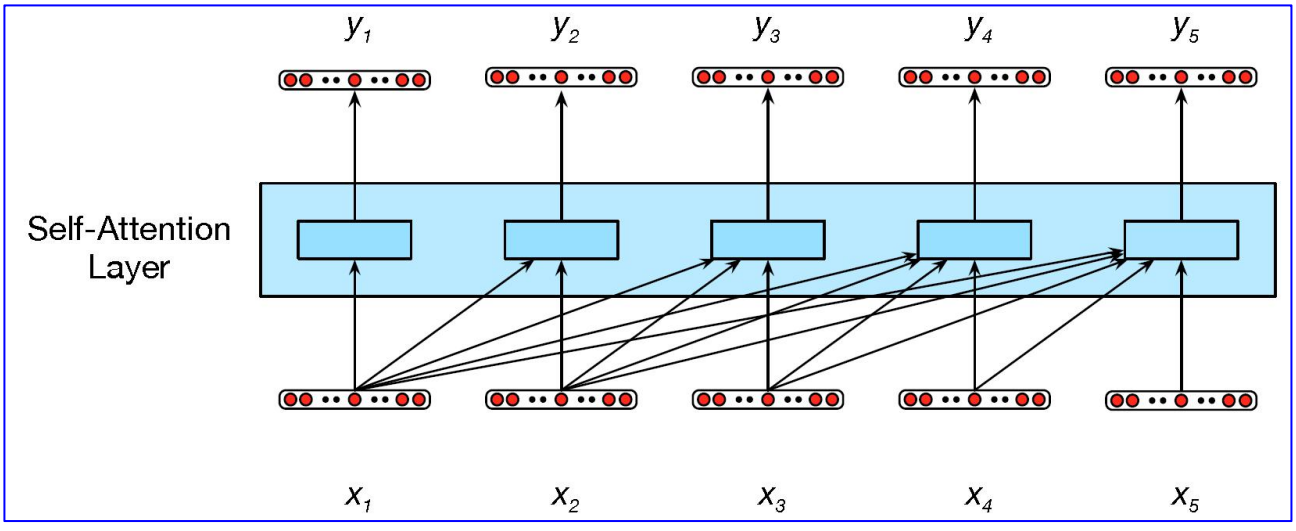

图 9-15：Self-Attention 模型

图注：因果(或掩盖)自注意力模型中的信息流。在处理序列中的每个元素时，模型都会处理直到当前(包括当前)的所有输入。与 RNN 不同，每个时间步的计算都独立于所有其他步骤，因此可以并行执行。

基于注意力的方法的核心是能够将感兴趣的项与其他项的集合进行比较，从而揭示它们在当前上下文中的相关性。在自注意力的情况下，这组比较是针对给定序列中的其他元素的。然后，将这些比较的结果用于计算当前输入的输出。例如，返回到图 9.15，y_3 的计算基于输入 x_3 与它的先前元素 x_1 和 x_2 以及 x_3 本身之间的一组比较。自注意力层中元素之间比较的最简单形式是点积。为了进行其他可能的比较，我们将这些比较的结果称为得分。

$$SCORE(x_i, x_j) = x_i \cdot x_j \tag{9.14}$$

点积的结果是在-∞到∞范围内的标量值，该值越大，正在比较的向量越相似。继续我们的示例，计算 y_3 的第一步将是计算三个分数：x_3x_1，x_3x_2 和 x_3x_3。然后，为了有效利用这些得分，我们将使用 softmax 对它们进行归一化，以创建权重向量 α_{ij}，它表示每个输入与输入元素 i(当前注意力焦点)的比例相关性。

$$\alpha_{ij} = \text{softmax}(score(x_i, x_j)), \quad \forall j \le i \tag{9.15}$$

$$= \frac{exp(score(x_i, x_j))}{\sum_{k=1}^{i} exp(score(x_i, x_k))}, \qquad \forall j \le i \tag{9.16}$$

给定α中的比例分数，然后将目前所见的输入求总和，按各自的α值加权，生成一个输出值 y_i。

$$y_i = \sum_{j \le i} \alpha_{ij} x_j \tag{9.17}$$

公式 9.14 至 9.17 中体现的步骤表示基于注意力的方法的核心：在某些情况下对相关项进行的一组比较，对这些分数进行归一化以提供概率分布，然后使用该分布进行加权求总和。输出 y 是对输入进行直接计算的结果。

不幸的是，这种简单的机制没有提供学习的机会，一切都直接基于原始输入值 x。特别是，没有机会学习单词可以有助于较长输入表示的多种方式。为了进行这种学习，Transformers 以一组权重矩阵的形式包含其他参数，这些参数对输入嵌入进行操作。为了激发这些新参数，请考虑每个输入嵌入在注意力过程中扮演的不同角色(即 Q、K、V)。

- 和其他所有*先前输入*比较时，扮演***当前注意力的焦点***角色，称此为 query(即 Q)。
- 与*当前注意力焦点*比较时，扮演***先前输入***角色，称此为 key(即 K)。
- 用于计算*当前注意力焦点*的输出时，扮演***值***角色，称此为 value(即 V)。

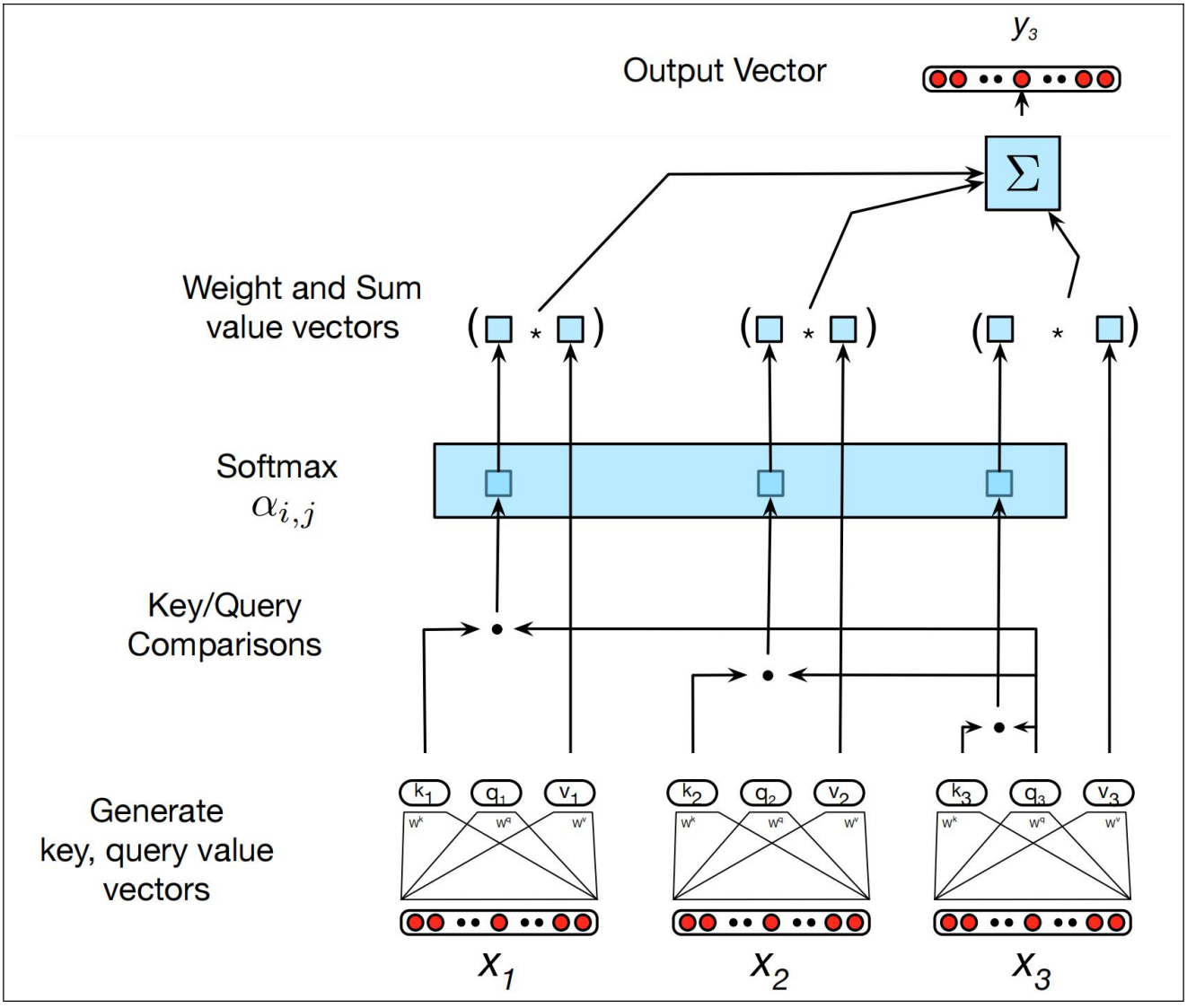

图 9-16：自注意力的计算

图注：使用因果自注意力，对序列中第三个元素的值进行计算。

为了捕获输入嵌入在每个步骤中扮演的不同角色，Transformer 引入了三组权重，我们将其称为 W^Q、W^K 和 W^V。这些权重将用于计算每个输入 x 的线性转换，其结果值将用于后续计算中各自的角色。

$$q_i = W^Q x_i; \qquad k_i = W^K x_i; \qquad v_i = W^V x_i$$

给定大小为 d_m 的输入嵌入，这些矩阵的维数分别为 $d_q \times d_m$, $d_k \times d_m$, $d_v \times d_m$。在原始 Transformer 著作(Vaswani 等人,2017)中，d_m 为 1024，d_q 和 d_v 为 64。

给定这些预测，即当前注意力焦点 x_i 与先前上下文中的元素 x_j 之间的分数，由其 Q 向量 q_i 和先前元素 K 向量 k_j 之间的点积组成。让我们更新之前的比较计算以反映这一点。

$$SCORE(x_i, x_j) = q_i \cdot k_j \tag{9.18}$$

随之产生的 softmax 计算结果α_{ij} 保持不变，但是 y_i 的输出计算现在基于值向量 v 的加权和。

$$y_i = \sum_{j \le i} \alpha_{ij} v_j \tag{9.19}$$

图 9.16 演示了在计算序列的第三个输出 y_3 时的这种计算。

在计算α_{ij} 时出现的实际考虑是由于将点积与 softmax 中的指数结合起来用作比较。点积的结果可以是任意大(正或负)值。对如此大的值求幂可能会导致数值问题，并导致训练过程中梯度的实际损失。为了避免这种情况，需要以适当的方式缩放点积。缩放点积方法将点积的结果除以与嵌入大小相关的因子，然后再将其通过 softmax。一种典型的方法是将点积除以查询(Q)和键(K)向量的维数的平方根，从而导致我们再次更新评分函数。

$$SCORE(x_i, x_j) = \frac{q_i \cdot k_j}{\sqrt{d_k}} \tag{9.20}$$

自注意力过程的描述是从在特定时间点计算单个输出的角度出发的。但是，由于每个输出 y_i 都是独立计算的，因此可以通过使用有效的矩阵乘法例程来并行化整个过程，方法是将输入的嵌入内容打包到单个矩阵中，并将其乘以键(K)、查询(Q)和值(K)矩阵，以生成包含所有键、查询和值向量的矩阵。

$$Q = W^Q X; \quad K = W^K X; \quad V = W^V X$$

给定这些矩阵，我们可以通过在单个矩阵乘法中将 K 和 Q 相乘来同时计算所有必要的 query-key 比较。更进一步，我们可以缩放这些分数，取 softmax，然后将结果乘以 V，从而将整个序列的整个自注意力步骤减少到以下计算：

$$SelfAttention(Q, K, V) = \text{softmax}\frac{QK^T}{\sqrt{d_k}}V \tag{9.21}$$

不幸的是，这个过程有点过头了，因为 QK^T 中的比较计算会为每个查询值对每个键值(包括查询后面的键值)产生一个分数。这在语言建模的设置中是不合适的，因为如果您已经知道下一个单词，那么猜测它是非常简单的。为了解决这个问题，比较矩阵的上三角部分中的元素被清零(设为-∞)，从而消除序列后面的单词的任何知识。

Transformer 模块

自注意力的计算是所谓的 Transformer 模块的核心，该模块除了自注意力层之外，还包括其他前馈层、残余连接和归一化层。图 9.17 展示了一个典型的 Transformer 模块，该模块由一个单一的注意力层，一个完全连接的前馈层，一个残余连接和每个层之后的归一化层组成。然后可以像堆叠 RNN 一样堆叠这些块。

多头注意力

句子中不同的单词可以同时以多种不同的方式相互关联。例如，动词及其在句子中的论元之间可以保持明显的句法、语义和话语关系。单个 Transformer 模块很难学习捕获其输入之间的所有不同种类的并联关系。Transformer 使用多头自注意力层解决了这个问题。这些层是一组自注意力层，称为头，它们位于层中，平行于模型中相同深度的层，每个层都有自己的一组参数。给定这些不同的参数集，每个头都可以学习在同一抽象级别上存在的输入之间的关系的不同方面。

为了实现此概念，在自注意层中的每个头 i 都拥有自己的一组键，查询和值矩阵：W_i^K，W_i^Q 和 W_i^V。它们被用于将每个头的输入分别投影到层 x_i，其余的自注意力计算保持不变。具有 h 个头的多头层的输出包含相同长度的 h 个向量。为了在进一步处理中使用这些矢量，将它们组合在一起，然后减小到原始输入维度 d_m。这是通过拼接来自每个头的输出，然后使用另一个线性投影将其减小到原始输出维度来实现的。

$$MultiHeadAttn(Q, K, V) = W^O(head_1 \oplus head_2 \ldots \oplus head_h)$$

$$head_i = SelfAttention(W_i^Q X, W_i^K X, W_i^V X)$$

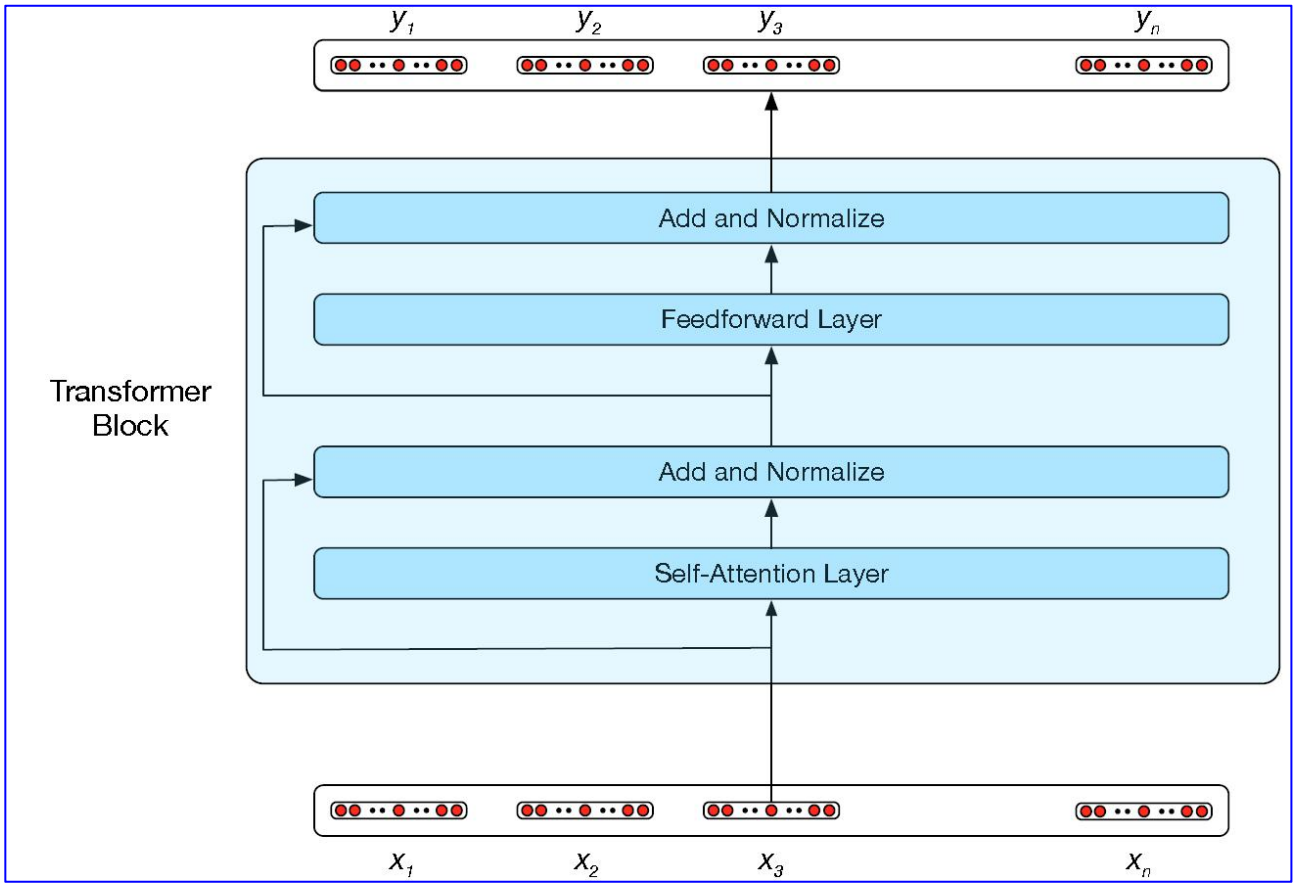

图 9-17：典型的 Transformer 模块

图 9.18 用 4 个自注意力头说明了这种方法。该多头层取代了图 9.17 中所示的 Transfomer 模块中的单个自注意力层，Transfomer 模块的其余部分及其前馈层、残余连接层和层归一化保持不变。

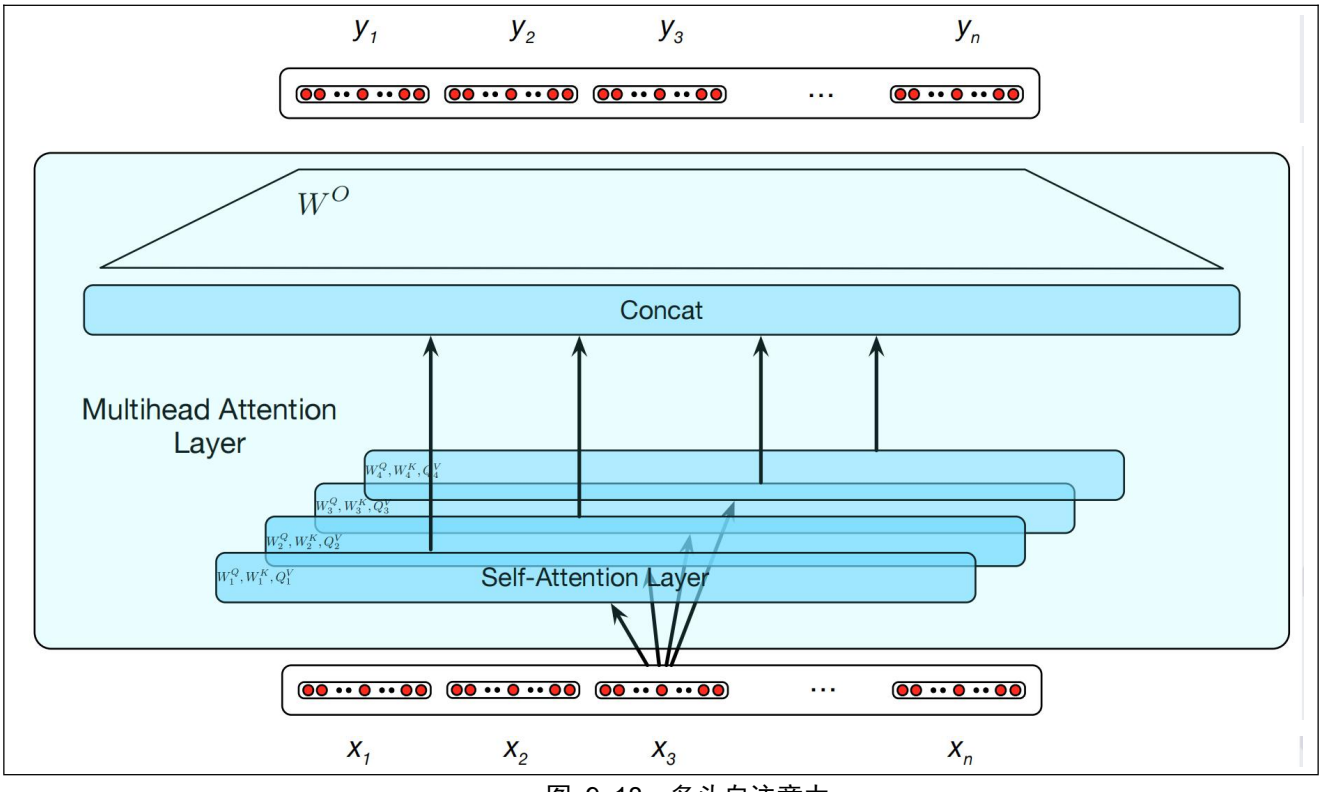

图 9-18：多头自注意力

图注：每个多头自注意层力都有自己的一组键(K)、查询(Q)和值(V)权重矩阵。每个层的输出被拼接起来，然后向下投影到 d_{model} 中，从而产生正确大小的输出。

位置嵌入

利用 RNN，有关输入顺序的信息被纳入模型的性质。不幸的是，Transformer 并非如此。没有什么可以让这种模型利用有关输入序列元素相对或绝对位置的信息。从以下事实可以看出这一点：如果您在前面说明的注意力计算中对输入的顺序进行了打乱，您将得到完全相同的答案。为了解决这个问题，将 Transformer 输入与特定于输入序列中每个位置的**位置嵌入(positional embedding)**相结合。

我们在哪里得到这些位置嵌入？ 一种简单有效的方法是从随机初始化的嵌入开始，这些嵌入对应于每个可能的输入位置，直到某个最大长度。例如，就像有一个单词 fish 嵌入一样，我们也为位置 3 嵌入了一个词。与单词嵌入一样，这些位置嵌入是在训练过程中与其他参数一起学习的。为了产生捕获位置信息的输入嵌入，我们只需将每个输入的词嵌入添加到其对应的位置嵌入中。这种新的嵌入用作进一步处理的输入。

这种方法的潜在问题是，我们输入中的初始位置将有大量的训练样本，而在长度极限之外的位置相应的样本会更少。因此，后面的这些嵌入可能训练不足，并且在测试过程中可能无法很好地泛化。位置嵌入的另一种方法是选择一个静态函数，该函数以捕获位置之间固有关系的方式将整数输入映射到实值向量。也就是说，它捕获了一个事实：相比于输入中的位置 4 与位置 17 的关系，位置 4 与位置 5 的关系更紧密。原始 Transformer 工作中使用了具有不同频率的正弦和余弦函数的组合。

9.4.1. Transformer 作为自回归语言模型

现在，我们已经了解了 Transformers 的所有主要组件，下面我们来研究如何通过半监督学习将它们部署为语言模型。为此，我们将像基于 RNN 的方法一样继续进行操作：给定纯文本训练语料库，训练模型用教师强制方法来预测序列中的下一个单词。图 9.19 说明了一般方法。在每个步骤中，给定所有前面的单词，最终的 Transformer 层会在整个词汇表上产生输出分布。在训练过程中，分配给正确单词的概率用于计算序列中每个项的交叉熵损失。与 RNN 一样，训练序列的损失是整个序列的平均交叉熵损失。

请注意，该图与先前的基于 RNN 的版本之间的关键区别如图 9.6 所示。给定隐藏状态的计算循环，此处输出的计算和每一步的损失本质上都是串行的。使用 Transformer，可以并行处理每个训练项，因为序列中每个元素的输出是分别计算的。训练后，我们可以计算所得模型的困惑度，或像基于 RNN 的模型一样自动回归生成新颖的文本。

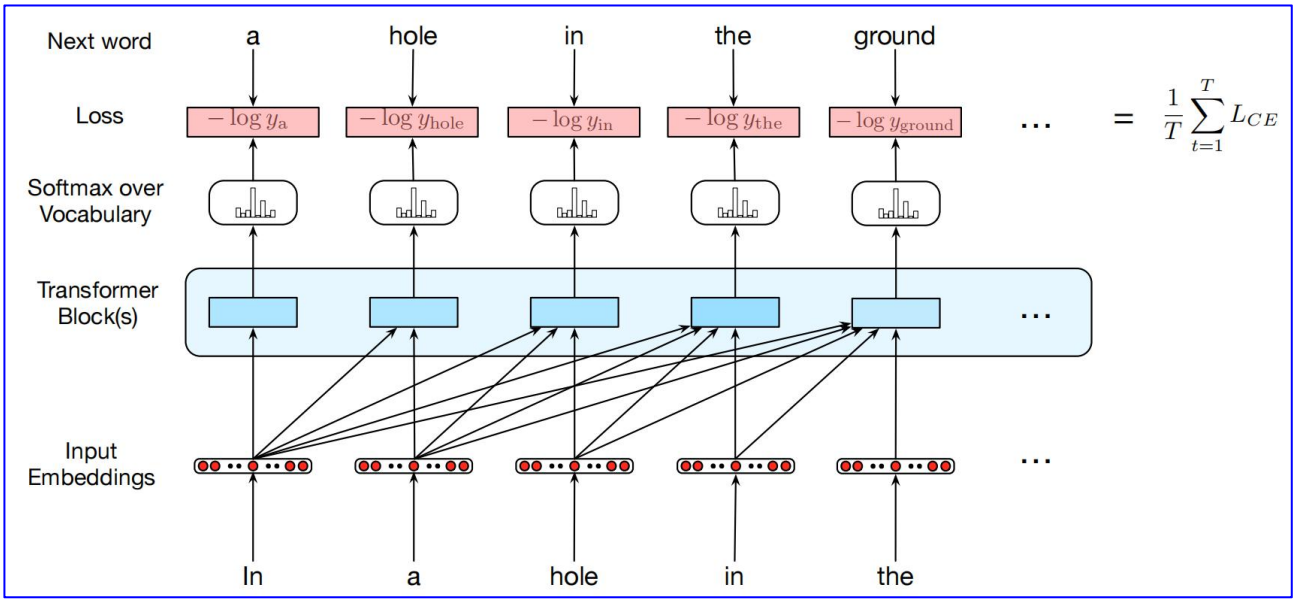

图 9-19：将 Transformer 训练为语言模型

上下文生成和摘要

自回归生成的一个简单变体是许多实际应用的基础，它使用先验上下文来准备自回归生成过程。图

9.20 用**文本补全(text completion)**任务说明了这一点。在这里，标准语言模型被赋予某些文本的前缀，并被要求为其生成可能的补全。请注意，随着生成过程的进行，模型可以直接访问头道(priming)上下文以及所有其自身随后生成的输出。在每个时间步整合整个较早上下文和生成的输出的能力是这些模型强大功能的关键。

文本摘要(text summarization)是基于上下文的自回归生成的实际应用。在这里，任务是获取全长文章，并提供有效的摘要。为了训练基于 Transformer 的自回归模型来执行此任务，我们从一个语料库开始，该语料库由全长文章及其相应的摘要组成。图 9.21 显示了来自广泛使用的摘要语料库的此类数据的示例，该摘要语料库由 CNN 和 Daily Mirror 新闻文章组成。

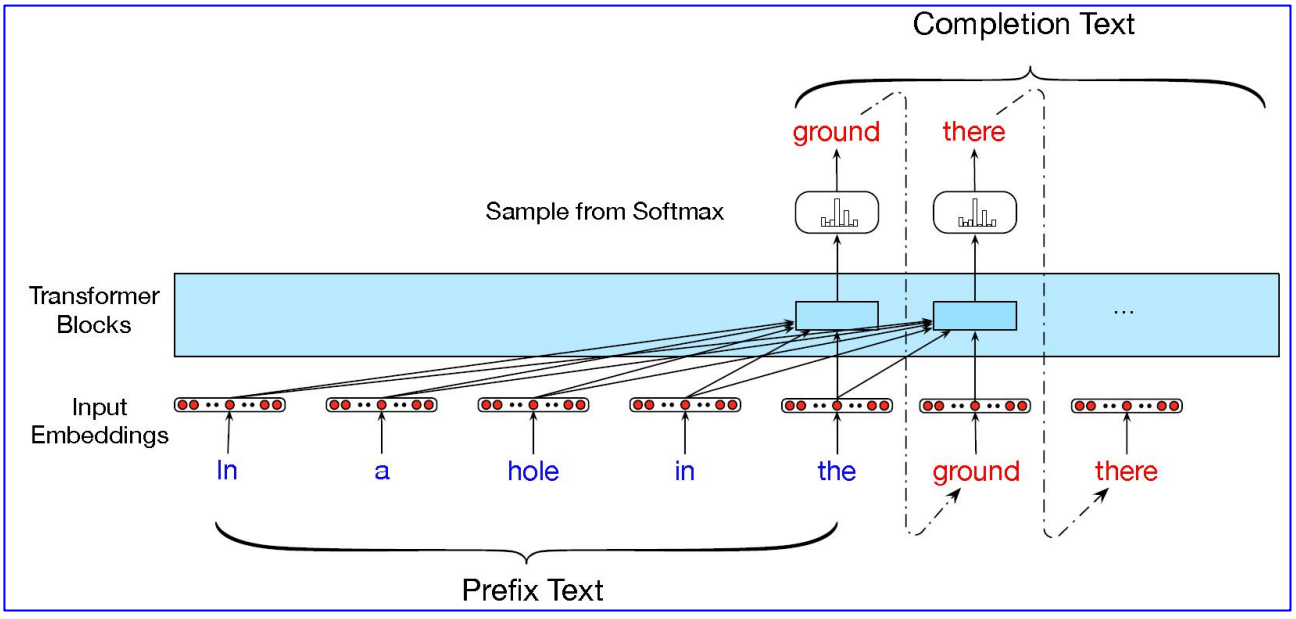

图 9-20：Transformers 的自回归文本补全

Original Article

The only thing crazier than a guy in snowbound Massachusetts boxing up the powdery white stuff and offering it for sale online? People are actually buying it. For $89, self-styled entrepreneur KyleWaring will ship you 6 pounds of Boston-area snow in an insulated Styrofoam box – enough for 10 to 15 snowballs, he says.

But not if you live in New England or surrounding states. “We will not ship snow to any states in the northeast!” saysWaring’s website, ShipSnowYo.com. “We’re in the business of expunging snow!”

His website and social media accounts claim to have filled more than 133 orders for snow – more than 30 on Tuesday alone, his busiest day yet. With more than 45 total inches, Boston has set a record this winter for the snowiest month in its history. Most residents see the huge piles of snow choking their yards and sidewalks as a nuisance, but Waring saw an opportunity.

According to Boston.com, it all started a few weeks ago, when Waring and his wife were shoveling

deep snow from their yard in Manchester-by-the-Sea, a coastal suburb north of Boston.

He joked about shipping the stuff to friends and family in warmer states, and an idea was born. His business slogan: “Our nightmare is your dream!” At first, ShipSnowYo sold snow packed into empty 16.9-ounce water bottles for $19.99, but the snow usually melted before it reached its destination…

Summary

KyleWaring will ship you 6 pounds of Boston-area snow in an insulated Styrofoam box – enough for 10 to 15 snowballs, he says. But not if you live in New England or surrounding states.

图 9-21：CNN/每日邮报语料库的文章和摘要示例

图注：来自(Hermann 等人, 2015b), (Nallapati 等人, 2016)。

将 Transformers 应用于摘要的一种出乎意料的有效方法是将摘要添加到语料库中的每篇全长文章中，并用独特的标记将两者分开。更正式地说，训练语料库中的每个文章摘要配对$(x_1, \ldots, x_m)$，$(y_1, \ldots, y_n)$都转换为单个训练实例$(x_1, \ldots, x_m, \delta, y_1, \ldots y_n)$，总长度为 n + m +1。将这些训练实例视为长句子，然后像我们之前所做的那样，用教师强制方法来训练自回归语言模型。

经过培训后，以特殊标记结尾的完整文章将用作上下文，引导生成过程并且产生摘要，如图 9.22 所示。请注意，与 RNN 相比，该模型在整个过程中都可以访问原始文章以及新生成的文本。

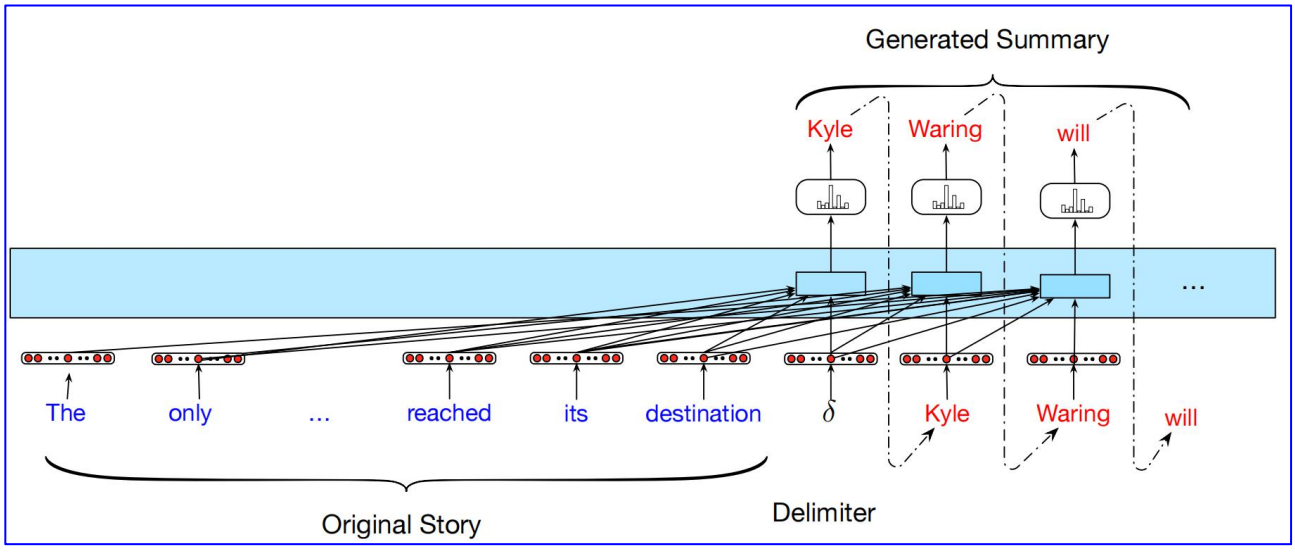

图 9-22：Transformers 用于摘要

正如我们将在后面的章节中看到的那样，此简单方案的各种变体是成功进行文本到文本应用程序(包括机器翻译、摘要和问答)的基础。

9.5. 语言模型的潜在危害

大型神经语言模型表现出许多在第 4 章和第 6 章中讨论的潜在危害。每当将语言模型用于文本生成时(如在 Web 搜索查询完成或电子邮件的预测键入的辅助技术中)，都可能会出现问题(Olteanu 等人，2020)。

例如，语言模型可以生成有毒的语言。Gehman 等人(2020)表明，许多种类的完全无毒的提示仍然可以导致大型语言模型输出仇恨言论和辱骂。Brown 等人(2020)和 Sheng 等人(2019)表明，大型语言模型生成的句子显示出对少数族裔身份(如黑人或同性恋)的消极态度。

实际上，语言模型因其训练数据的分布而在许多方面存在偏见。Gehman 等人(2020 年)表明，大型语言模型训练数据集包括从禁令站点刮取的有毒文字。除了毒性问题之外，发达国家的作者还不成比例地生成了互联网数据，许多大型语言模型都基于 Reddit 的数据进行训练，其作者偏向于男性和年轻人。这种偏倚的人口样本，可能会使一代人偏离到代表性不足的观点或主题。此外，正如我们在第 6 章中看到的嵌入模型一样，语言模型可以放大训练数据中的人口统计特征和其他偏差。

语言模型也可以是生成用于错误信息、网络钓鱼、激进和其他对社会有害的活动的文本的工具(Brown 等人，2020)。(McGuffie 和 Newhouse，2020)显示了大型语言模型如何生成模仿在线极端分子的文本，并有加剧极端主义运动及其激进和招募的风险。

最后，还有重要的隐私问题。和其他机器学习模型一样，语言模型也会泄露其训练数据的信息。

因此，对手可以从语言模型中提取个人训练数据短语，如个人的姓名、电话号码和地址(Carlini 等人 2020 年，使用 Henderson 等人 2017 年介绍的技术)。如果在私人数据集上训练大型语言模型，比如电子健康记录(EHRs)，这就是一个问题。

如何减轻这些危害是自然语言处理中一个重要但尚未解决的研究问题。额外的对无毒子语料的预训练(Gururangan 等人，2020)似乎在一定程度上减少了语言模型产生有毒语言的倾向(Gehman 等人，2020)。分析用于预训练大型语言模型的数据对于理解生成过程中的毒性和偏见以及隐私都很重要，因此，语言模型包括**数据表**或**模型卡**(第 4.10 节)非常重要，数据表提供了用于训练它们的语料库上的完全可复制的信息。

9.6. 总结

本章介绍了**循环神经网络(RNN)**的概念，以及如何将它们应用于语言问题。以下是我们总结的要点:

•在**简单的循环神经网络(SRN)**中，序列每次都作为一个元素自然地处理。

•神经单元在特定时间点的输出既基于当前输入，也基于来自上一时间步的隐藏层的值。

•RNN 可以通过**反向传播算法(**称为时间反向传播(BPTT))的直接扩展来训练。

•RNN 的基于通用语言的应用程序包括：

–概率语言建模，其中模型将概率分配给序列，或序列的下一个元素(在给定前面单词的情况下)。

–使用经过训练的语言模型**自动回归生成**。

–序列标签，为序列的每个元素分配一个标签，如词类标记一样。

–序列分类，将整个文本分配给一个类别，例如垃圾邮件检测、情感分析或主题分类。

•简单的循环网络常会失败，因为要成功地训练它们来解决随时间推移保持有用梯度的问题非常困难。

•更复杂的门控架构，如 **LSTM** 和 **GRU**，被设计来克服这些问题，通过显式管理任务，决定在其隐藏和上下文层中记住和忘记什么。

9.7. 文献和历史说明

在 1980 年代，在圣地亚哥加州大学的并行分布式处理(PDP)小组的背景下，对这里讨论的简单 RNN 的类型进行了有影响的研究。这项工作大部分是针对人类认知建模的，而不是针对实际的 NLP 应用的(Rumelhart 和 McClelland 1986 ，McClelland 和 Rumelhart1986)。

Elman(1990)引入了在前馈网络(Elman 网络)的隐藏层使用循环的模型。Jordan(1986)用输出层的循环研究了类似的架构，Mathis 和 Mozer(1995)在隐藏层之前添加了一个循环上下文层。在(Rumelhart 和 McClelland，1986)中讨论了将循环网络展开为等效前馈网络的可能性。

在认知建模工作的同时，RNN 在信号处理和语音社区的连续领域得到了广泛的研究(Giles 等人，1994)。Schuster 和 aliwal(1997)引入了双向 RNN，并描述了 TIMIT 音子转录任务的结果。

尽管从理论上讲很有趣，但是训练 RNN 和在长序列上管理上下文的困难阻碍了实际应用的进展。随着 Hochreiter 和 Schmidhuber(1997)引入 LSTM，这种情况发生了变化。在信号处理和语言处理的边界任务上表现出令人印象深刻的性能提升，包括音子识别(Graves 和 Schmidhuber，2005)，手写识别(Graves 等人，2007)和最重要的语音识别(Graves 等人，2013b)。

由于 Collobert 和 Weston(2008)以及 Collobert 等人(2011)的工作，人们对将神经网络应用于实际的自然语言处理问题的兴趣大增。这些努力利用了词汇嵌入、卷积网络和端到端的训练。他们在许多标准的共享任务中，包括词类标记、分块、命名实体识别和语义角色标记，在没有使用人工工程特征的情况下，展示了接近最先进的性能。

将 LSTM 与基于 word2vec(Mikolov 等人，2013)和 GLOVE(Pennington 等人，2014)的词嵌入的预训练集合相结合的方法迅速主导了许多常见任务：词类标记(Ling 等人，2015)，句法分块(Søgaard 和 Goldberg，2016)，并通过 IOB 标记(Chiu 和 Nichols 2016，Ma 和 Hovy 2016)，意见挖掘(Irsoy 和 Cardie，2014)，语义角色标记(Zhou 和 Xu，2015a)，AMR 解析(Foland 和 Martin，2016)。与早期统计机器学习的快速发展一样，这些进展是由于 CONLL、SemEval 提供的训练数据和其他共享任务，以及 Ontonotes (Pradhan 等人，2007b)和 PropBank (Palmer 等人，2005)等共享资源的可用性而实现的。

9.8. 练习

(无)