8 RNNs and LSTMs

语言本质上是一种时间现象。口语是一系列随时间变化的声学事件，我们理解和产生口头语言和书面语言时，都将其视为一个连续的输入流。语言的时间特性也体现在我们使用的比喻中；我们谈论“对话的流动”、“新闻流”和“推特流”，这些都强调了语言是随着时间展开的序列。

这种时间特性也反映在一些语言处理算法中。例如，我们之前在HMM词性标注中介绍的维特比算法就是按时间顺序逐个词地处理输入，同时携带之前获取的信息向前推进。然而，其他机器学习方法，如我们用于情感分析或其他文本分类任务的方法，并没有这种时间特性——它们假设可以同时访问输入的所有方面。

第七章中的前馈神经网络也假设可以同时访问输入，尽管它们对时间有一个简单的模型。回想一下，我们将前馈网络应用于语言建模时，只关注固定大小的词窗口，然后将这个窗口在输入上滑动，沿途独立进行预测。这种滑动窗口方法也用于我们将在第九章介绍的Transformer架构中。

本章介绍了一种深度学习架构，提供了另一种表示时间的方式：循环神经网络（RNNs）及其变体，如长短期记忆网络（LSTMs）。RNN具有一种直接处理语言序列特性的机制，允许它们处理语言的时间特性，而无需使用任意的固定大小的窗口。循环网络通过其循环连接提供了一种新的方式来表示先前的上下文，使模型的决策可以依赖于数百个词之前的信息。我们将看到如何将该模型应用于语言建模任务、诸如词性标注等序列建模任务，以及诸如情感分析等文本分类任务。

8.1 循环神经网络（RNNs）

循环神经网络（RNN）是指任何包含循环连接的网络，即某个单元的值直接或间接依赖于其自身的早期输出作为输入。虽然这种网络非常强大，但它们很难进行推理和训练。然而，在循环网络的总体类别中，有一些受限的架构在应用于语言时证明是非常有效的。本节中，我们将讨论一种称为**Elman网络**（Elman, 1990）或**简单循环网络的循环网络**类别。这些网络本身就很有用，并且作为更复杂方法的基础，比如本章稍后讨论的长短期记忆网络（LSTMs）。在本章中，当我们使用“RNN”这个术语时，我们将指代这些更简单、更受限的网络（尽管你经常会看到“RNN”这个术语用来指任何具有循环特性的网络，包括LSTM）。

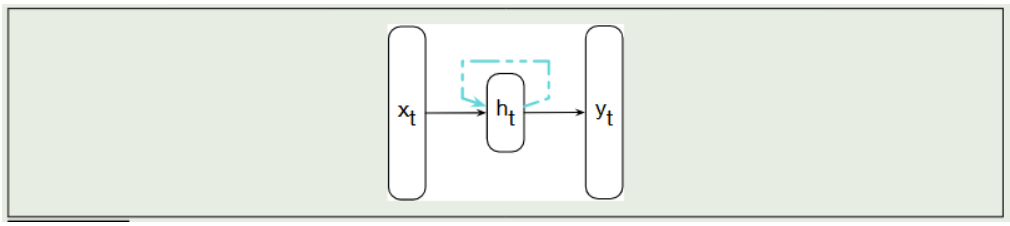

图 8.1 Elman (1990)之后的简单递归神经网络。隐藏层包括一个循环连接作为其输入的一部分。也就是说，隐含层的激活值取决于当前输入以及来自上一个时间步的隐含层激活值。

图8.1展示了RNN的结构。与普通的前馈网络一样，表示当前输入的输入向量 xt 会与权重矩阵相乘，然后通过一个非线性激活函数来计算隐藏层的值。然后，这个隐藏层用于计算相应的输出 yt。与

之前基于窗口的方法不同，RNN处理序列时是一次向网络输入一个 item。我们将使用下标来表示时间，因此 xt 将表示时间 t 时刻的输入向量 x。与前馈网络的关键区别在于图中用虚线表示的循环链接。这个链接将前一个时刻隐藏层的值加入到当前时刻隐藏层的计算中，增强了输入信息。

前一时间步的隐藏层提供了一种记忆或上下文的形式，它编码了早期（前面）的处理结果，并影响稍后时间点的决策。关键在于，这种方法不会对之前的上下文施加固定长度的限制；前一隐藏层所体现的上下文可以包含序列开头以来的延续信息。

增加了这个时间维度后，使得RNN看起来比非循环架构复杂得多。但实际上，它们并没有那么不同。给定一个输入向量和前一时间步的隐藏层值，我们仍然执行的是第七章中介绍的标准前馈计算。为了更清楚地看到这一点，请参考图8.2，它阐明了循环的本质以及它如何融入隐藏层的计算。最显著的变化在于一组新的权重 U，它将前一时间步的隐藏层连接到当前时间步的隐藏层。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。与网络中的其他权重一样，这些连接通过反向传播进行训练。

8.1.1 RNN中的推理

在RNN中，前向推理（将输入序列映射到输出序列）几乎与我们之前在前馈网络中看到的相同。为了计算输入 xt 的输出 yt，我们需要计算隐藏层的激活值 ht。要计算这个值，我们将输入 xt 与权重矩阵 W 相乘，并将前一个时间步的隐藏层 ht−1 与权重矩阵 U 相乘。我们将这些值相加，并通过一个合适的激活函数 g，得到当前隐藏层的激活值 ht。一旦得到隐藏层的值，我们就可以进行常规计算来生成输出向量：

$$\mathbf{h}_t = g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t)$$
$$\mathbf{y}_t = f(\mathbf{V}\mathbf{h}_t)$$

假设输入层、隐藏层和输出层的维度分别为 d_{in} 、 d_h 和 d_{out} 。根据这个假设，我们有三个参数矩阵： $W \in \mathbb{R}^{d_h \times d_{in}}$ ， $U \in \mathbb{R}^{d_h \times d_h}$ ，以及 $V \in \mathbb{R}^{d_{out} \times d_h}$ 。

我们通过softmax计算来得到 yt，它生成了可能输出类别的概率分布。

$$\mathbf{y}_t = \text{softmax}(\mathbf{V}\mathbf{h}_t)$$

在时间 t 的计算需要前一时间步 t−1 的隐藏层值，这要求使用一种增量推理算法，从序列的开始逐步计算到结束，如图8.3所示。简单循环网络的序列特性也可以通过将网络在时间上展开来观察，如图8.4所示。在这个图中，不同时间步的单元层被复制出来，以说明它们在不同时间会有不同的值。然而，各个时间步的权重矩阵是共享的。

```
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 8.3 简单循环网络中的前向推断。矩阵U，V和W在时间上是共享的，而h和y的新值是在每个时间步计算的。

8.1.2 训练

与前馈网络类似，我们将使用训练集、损失函数和反向传播来获取调整这些循环网络中的权重所需的梯度。如图8.2所示，我们现在有三组权重需要更新：W（从输入层到隐藏层的权重）、U（从前一隐藏层到当前隐藏层的权重）以及 V（从隐藏层到输出层的权重）。

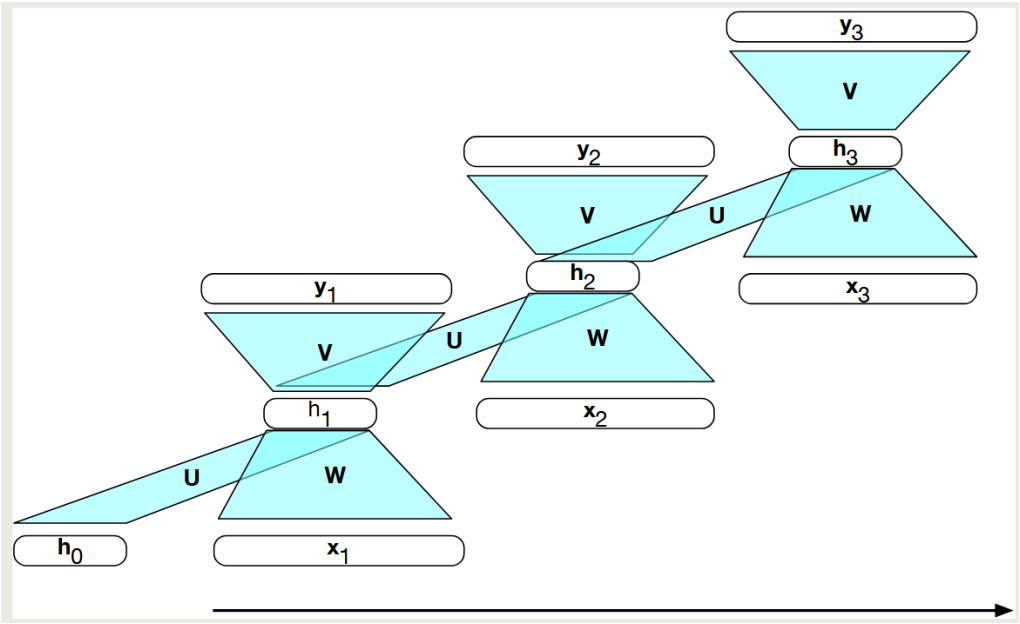

图 8.4 一个简单的递归神经网络在时间上是展开的。网络层在每个时间步重新计算，而权重U，V和W在所有时间步共享

图8.4强调了两个我们在前馈网络的反向传播中不需要担心的因素。首先，要计算时间 t 的输出损失函数，我们需要时间 t－1 的隐藏层。其次，时间 t 的隐藏层不仅影响时间 t 的输出，还会影响时间 t+1 的隐藏层（因此也影响 t+1 的输出和损失）。由此可见，为了评估累积到 ht 的误差，我们需要了解它对当前输出以及后续输出的影响。

根据这种情况调整反向传播算法，得到了一个两步训练RNN权重的算法。第一步，我们进行前向推理，计算 ht 和 yt，并在每个时间步累积损失，同时保存每个时间步的隐藏层值以便用于下一个时间步。第二步，我们反向处理序列，计算所需的梯度，同时在每个时间步保存隐藏层的误差项，以便在时间反向传播时使用。这个通用的方法通常被称为时间上的反向传播（Backpropagation Through Time, BPTT）（Werbos 1974，Rumelhart 等 1986，Werbos 1990）。

幸运的是，随着现代计算框架和充足计算资源的进步，训练RNN并不需要专门的方法。如图8.4所示，显式地将循环网络展开为一个前馈的计算图可以消除任何显式的循环，从而允许直接训练网络的权重。在这种方法中，我们提供一个模板来指定网络的基本结构，包括输入层、输出层和隐藏层的所有必要参数、权重矩阵、以及使用的激活函数和输出函数。然后，当给定一个特定的输入序列时，我们可以生成一个针对该输入的展开前馈网络，并通过普通的反向传播来进行前向推理或训练。

对于涉及更长输入序列的应用，如语音识别、字符级处理或连续输入流的应用，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片段，并将每个片段视为一个独立的训练项。

8.2 RNN 作为语言模型

让我们看看如何将 RNN 应用于语言建模任务。回顾第 3 章中的内容，语言模型根据给定的前文预测序列中的下一个词。例如，如果前文是 "Thanks for all the"（谢谢你给了我所有的），我们想知道下一个词 "fish"（鱼）出现的可能性，我们会计算：

$$P(fish|Thanksforallthe)$$

语言模型赋予每个可能的下一个词一个条件概率，从而生成一个覆盖整个词汇表的分布。通过结合这些条件概率和链式法则，我们还可以为整个序列分配概率：

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

第 3 章中的 n-gram 语言模型根据某个词与之前 n－1 个词一起出现的次数来计算该词的概率。因此，n-gram 模型的上下文大小为 n－1。对于第 7 章中的前馈语言模型，上下文是窗口的大小。

RNN 语言模型（Mikolov 等, 2010）一次处理一个输入词，尝试根据当前词和前一个隐藏状态来预测下一个词。因此，RNN 不会像 n-gram 模型那样遇到有限上下文的问题，也不会像前馈语言模型那样有固定的上下文，因为在原理上，RNN 的隐藏状态可以表示从序列开头开始的所有前文信息。图 8.5 描绘了前馈神经网络（FFN）语言模型和 RNN 语言模型之间的区别，显示了 RNN 语言模型使用 ht－1（即前一个时间步的隐藏状态）作为前文上下文的表示。

8.2.1 RNN语言模型中的前向推理

在循环语言模型中的前向推理过程与8.1.1节中描述的一样。输入序列 $\mathbf{X} = [\mathbf{x}_1; ...; \mathbf{x}_t; ...; \mathbf{x}_N]$ 由一系列词组成，每个词用大小为 $|V| \times 1$ 的one-hot向量表示，输出预测 y 是一个表示词汇表上概率分布的向量。在每个时间步，模型使用词嵌入矩阵 E 来检索当前词的嵌入，将其与权重矩阵 W 相乘，然后将其与前一时间步的隐藏层（通过权重矩阵 U 加权）相加，计算出新的隐藏层。这个隐藏层随后用于生成输出层，输出层通过 softmax 层生成整个词汇表的概率分布。在时间步 t：

$$\begin{aligned}
\mathbf{e}_t &= \mathbf{E}\mathbf{x}_t \\
\mathbf{h}_t &= g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{e}_t) \\
\hat{\mathbf{y}}_t &= \text{softmax}(\mathbf{V}\mathbf{h}_t)
\end{aligned}$$

为了方便介绍，在使用 RNN 进行语言建模时（我们在第9章介绍 Transformer 时也会看到这一点），假设词嵌入维度 d_e 和隐藏层维度 d_h 是相同的。因此，我们将这两个都称为模型维度 d。因此，词嵌入矩阵 E 的形状为 $[d \times |V|]$，而 xt 是形状为 $|V| \times 1$ 的one-hot向量。这样，嵌入 et 的形状为 $[d \times 1]$。W 和 U 的形状都是 $[d \times d]$，所以 ht 的形状也是 $[d \times 1]$。V 的形状为 $[|V| \times d]$，因此 Vh 的结果是一个形状为 $[|V| \times 1]$ 的向量。这个向量可以被看作是基于隐藏层 h 生成的词汇表上的分数。通过 softmax 函数将这些分数归一化为概率分布。词汇表中某个特定词 k 作为下一个词的概率由 $\hat{\mathbf{y}}_t[k]$ 表示，$\hat{\mathbf{y}}$ 的第 k 个分量：

$$P(w_{t+1} = k|w_1, \ldots, w_t) = \hat{\mathbf{y}}_t[k]$$

整个序列的概率只是序列中每个词的概率的乘积，我们用 $\hat{\mathbf{y}}_i[w_i]$ 来表示时间步 i处真实词 wi的概率：

$$P(w_{1:n}) = \prod_{i=1}^{n} P(w_i|w_{1:i-1})$$
$$= \prod_{i=1}^{n} \hat{\mathbf{y}}_i[w_i]$$

8.2.2 训练RNN语言模型

为了将RNN训练为语言模型，我们使用与第??节中介绍的**自监督**（或自训练）算法相同的方法：我们将一段文本作为训练材料，在每个时间步 t 让模型预测下一个词。我们称这样的模型为自监督模型，因为我们不需要为数据添加任何特殊的标注；词的自然序列本身就是监督信号！我们只是训练模型，以最小化预测训练序列中真实下一个词时的误差，使用交叉熵作为损失函数。回想一下，交叉熵损失衡量的是预测的概率分布与正确分布之间的差异：

$$L_{CE} = -\sum_{w \in V} \mathbf{y}_t[w] \log \hat{\mathbf{y}}_t[w]$$

在语言建模的情况下，正确的分布 yt 来自对下一个词的已知信息。这表示为一个one-hot向量，词汇表中实际下一个词对应的条目为1，其他所有条目为0。因此，语言建模中的交叉熵损失取决于模型赋予正确下一个词的概率。在时间步 t 时，交叉熵损失是模型赋予训练序列中下一个词的负对数概率：

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

因此，在输入的每个词位置 t 上，模型将正确词 wt 与先前的隐藏状态 ht−1 一起作为输入，ht−1 编码了之前词 w1:t−1$w_{1:t-1}$ 的信息，利用它们来计算可能的下一个词的概率分布，从而计算模型对下一个词 wt+1的损失。然后我们移到下一个词，我们忽略模型对下一个词的预测，取而代之的是使用正确的词 wt+1 和之前的历史信息来估计词 wt+2的概率。这种我们始终将正确的历史序列提供给模型以预测下一个词（而不是让模型使用上一个时间步的最佳预测结果）的策略称为“教师强制”（teacher forcing）。

通过梯度下降，网络中的权重会被调整，以最小化整个训练序列的平均交叉熵损失。图8.6展示了这一训练过程。

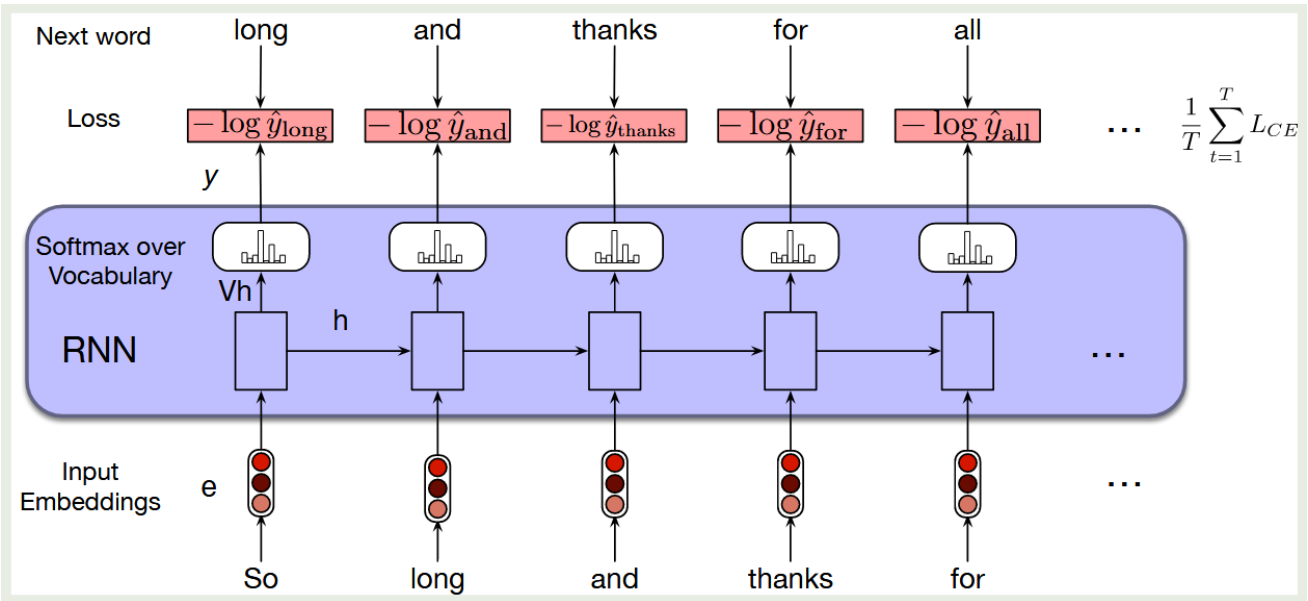

图 8.6 训练RNN作为语言模型。

8.2.3 权重共享

细心的读者可能已经注意到，输入嵌入矩阵 E 和用于输出 softmax 的最终层矩阵 V非常相似。

矩阵 E 的列表示词汇表中每个词的词嵌入，这些词嵌入是在训练过程中学到的，目标是让具有相似意义和功能的词具有相似的嵌入。而且，由于我们在使用 RNN 进行语言建模时假设嵌入维度和隐藏层维度是相同的（即模型维度 d），因此嵌入矩阵 E 的形状为 $[d \times |V|]$。最终层矩阵 V 提供了一种方式，通过计算 Vh，根据网络最后隐藏层中的信息来对词汇表中每个词的可能性进行评分。矩阵 V 的形状为 $[|V| \times d]$。也就是说，V 的行形状类似于 E*E* 的转置，这意味着 V 提供了第二组学习到的词嵌入。

不是使用两组嵌入矩阵，语言模型使用单一的嵌入矩阵，既出现在输入层也出现在 softmax 层。也就是说，我们舍弃 V，在计算的开头使用 E，并在计算的结尾使用 ET（因为 V 的形状是 E 的转置）。在两个地方使用同一个矩阵（转置形式）称为权重共享（weight tying）。RNN 语言模型的权重共享方程：

$$\begin{aligned}\mathbf{e}_t &= \mathbf{E}\mathbf{x}_t \\ \mathbf{h}_t &= g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{e}_t) \\ \hat{\mathbf{y}}_t &= \text{softmax}(\mathbf{E}^\intercal \mathbf{h}_t)\end{aligned}$$

这种方法不仅可以改善模型的困惑度（perplexity），还显著减少了模型所需的参数数量。

8.3 RNN在其他NLP任务中的应用

现在我们已经了解了基本的RNN架构，让我们看看如何将其应用于三种类型的自然语言处理（NLP）任务：序列分类任务（如情感分析和主题分类）、序列标注任务（如词性标注）和文本生成任务，包括使用一种新的架构——编码器-解码器（encoder-decoder）。

8.3.1 序列标注

在序列标注任务中，网络的任务是为序列中的每个元素分配一个来自固定标签集的标签，比如第17章中的词性标注和命名实体识别任务。在RNN的序列标注方法中，输入是词嵌入，输出是通过softmax层生成的标签集上的标签概率，如图8.7所示。

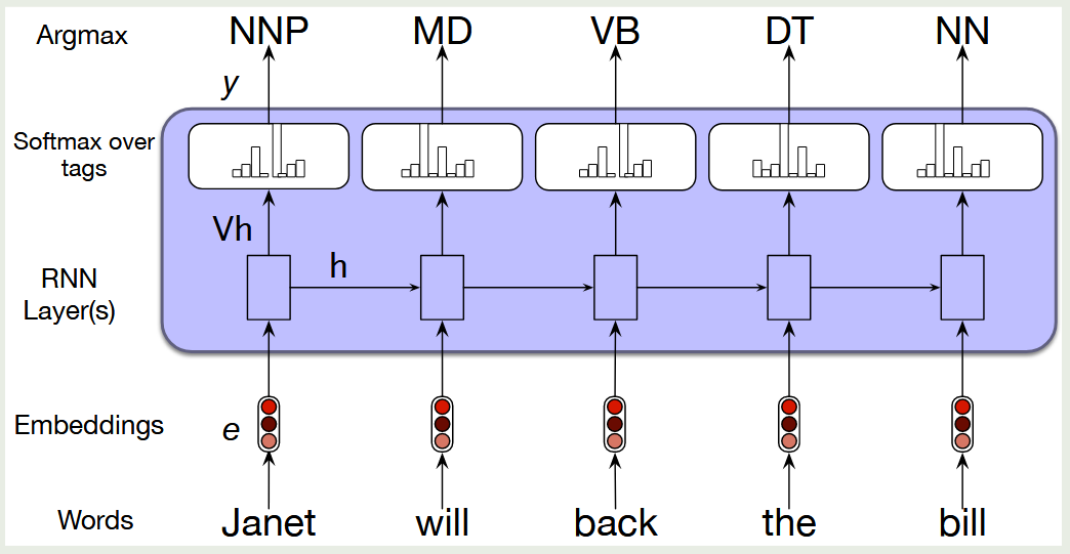

图 8.7 词性标注是用一个简单的RNN进行序列标注。预训练的词嵌入作为输入，softmax层在每个时间步提供词性标签的概率分布作为输出。

在这个图中，每个时间步的输入是预训练的、与输入标记对应的词嵌入。RNN模块是一个抽象表示，它代表一个展开的简单循环网络，在每个时间步都有一个输入层、隐藏层和输出层，以及组成网络的共享权重矩阵 U、V 和 W。网络在每个时间步的输出表示通过softmax层生成的词性标签集上的分布。

为了给定输入生成标签序列，我们在输入序列上进行前向推理，并在每个时间步从softmax中选择最可能的标签。由于我们使用softmax层在每个时间步生成输出标签集上的概率分布，因此在训练过程中我们将再次使用交叉熵损失。

8.3.2 RNN用于序列分类

RNN的另一个用途是对整个序列进行分类，而不是对序列中的标记进行分类。这类任务通常称为文本分类，如情感分析或垃圾邮件检测，我们将文本分类为两个或三个类别（如正面或负面），还有一些带有大量类别的分类任务，如文档级别的主题分类或用于客户服务应用的消息路由。

为了在这种设置中应用RNN，我们将要分类的文本逐词传递给RNN，在每个时间步生成一个新的隐藏层表示。然后我们可以取文本最后一个标记的隐藏层 hn，作为整个序列的压缩表示。我们可以将这个表示 hn 传递给前馈网络，通过softmax选择一个可能的类别，如图8.8所示。

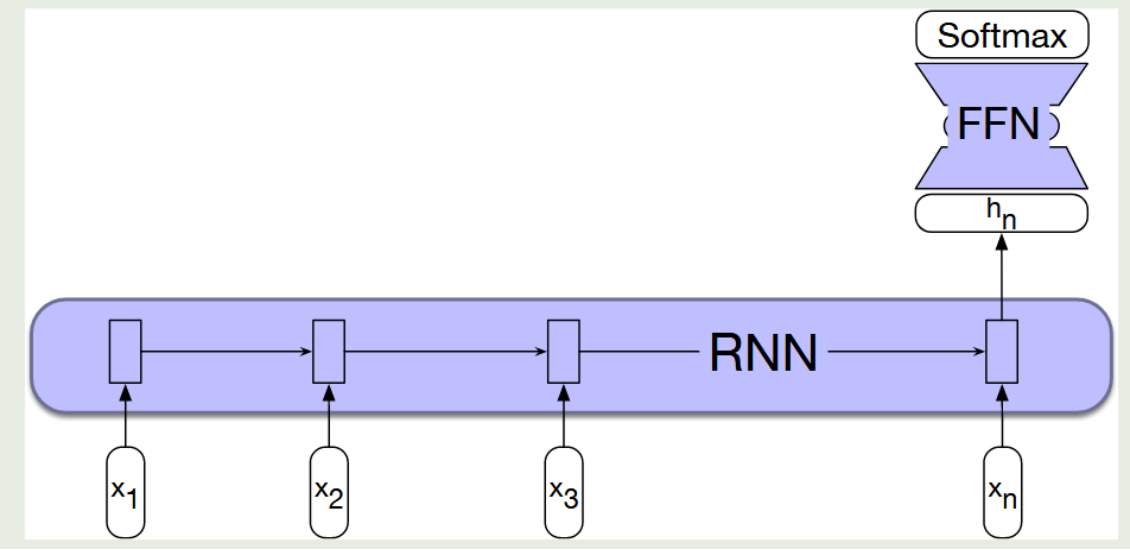

图 8.8 使用简单的RNN结合前馈网络进行序列分类。来自RNN的最终隐藏状态被用作执行分类的前馈网络的输入。

请注意，在这种方法中，我们不需要前序单词的中间输出。因此，与这些元素相关的没有损失项。相反，用于训练网络权重的损失函数完全基于最终的文本分类任务。前馈分类器的softmax输出与交叉熵损失一起驱动训练。分类产生的误差信号将通过前馈分类器的权重反向传播到其输入，然后再传递到RNN中的三组权重，如8.1.2节中所述。使用下游应用的损失来调整整个网络权重的训练方法称为**端到端训练**。

另一种选择是，不仅仅使用最后一个标记的隐藏状态 hn 来表示整个序列，而是使用对序列中每个词 i的所有隐藏状态 hi 进行某种**池化**函数。例如，我们可以通过对所有 n 个隐藏状态取元素均值来创建一个表示：

$$\mathbf{h}_{mean} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{h}_i$$

或者我们可以取元素的最大值；一组 n 向量的元素最大值是一个新向量，其第 k 个元素是所有 n 个向量中第 k 个元素的最大值。

由于RNN的长上下文特性，成功将误差反向传播到整个输入的难度很大；我们将在8.5节讨论这个问题及其一些常见的解决方案。

8.3.3 基于RNN的语言模型生成

基于RNN的语言模型也可以用于生成文本。文本生成在实际应用中有着巨大的重要性，涉及诸如问答系统、机器翻译、文本摘要、语法纠正、故事生成和对话系统等任务；任何需要根据其他文本生成新文本的任务。这种使用语言模型生成文本的方式是神经语言模型对NLP影响最为深远的领域之一。文本生成与图像生成和代码生成一起，构成了一个新的人工智能领域，通常称为**生成式AI**。

回顾第3章，我们曾看到如何通过n-gram语言模型生成文本，采用了Claude Shannon（Shannon, 1951）和心理学家George Miller及Jennifer Selfridge（Miller和Selfridge, 1950）同时期提出的一种**采样**技术。我们首先根据某个词作为序列开头的适用性随机采样一个词。然后，继续基于之前的选择采样词，直到达到预定长度，或生成结束标记。

如今，使用语言模型通过反复采样下一个词，**基于之前的选择逐步生成单词的方法被称为自回归生成（autoregressive generation）或因果语言模型生成（causal LM generation）**。这个过程基本上与第??页描述的一样，在此处调整为适用于神经网络环境：

- 首先，使用句子开头标记 <s> 作为第一个输入，从softmax分布中采样一个词作为输出。
- 然后，在下一个时间步中，使用该词的词嵌入作为输入，并以相同的方式采样下一个词。
- 持续生成，直到采样到句子结束标记 </s> 或达到固定长度限制。

技术上来说，**自回归模型**是在时间 t 基于时间 t－1、t－2 等的先前值的线性函数预测一个值。虽然语言模型不是线性的（因为它们包含许多非线性层），我们仍然广义地将这种生成技术称为自回归生成，因为在每个时间步生成的词都依赖于网络从上一个时间步选择的词。图8.9展示了这种方法。在图中，RNN隐藏层和循环连接的细节都隐藏在蓝色模块中。

这种简单的架构是诸如机器翻译、文本摘要和问答系统等应用的最新方法的基础。关键在于使用适当的上下文来初始化生成组件。也就是说，不仅仅使用 <s> 作为起始标记，我们可以提供更丰富的任务相关上下文；对于翻译任务，上下文是源语言的句子；对于摘要任务，上下文是我们想要总结的长文本。

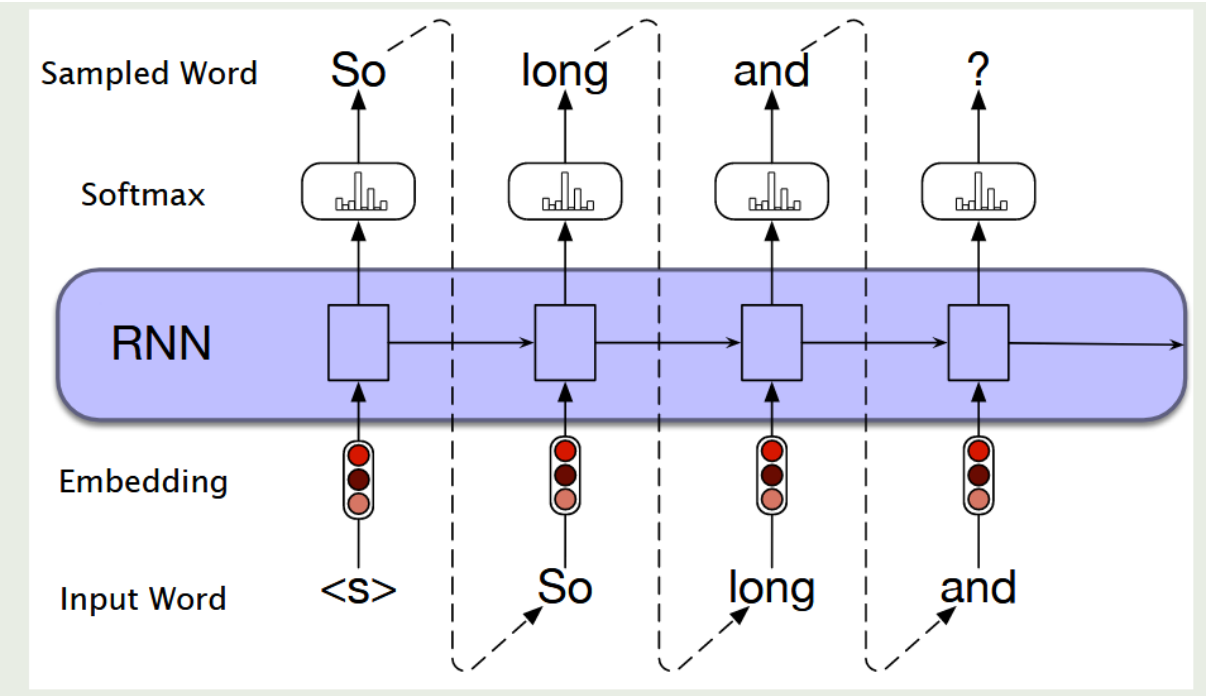

图 8.9 使用基于RNN的神经语言模型进行自回归生成。

8.4 堆叠和双向RNN架构

循环网络非常灵活。通过将展开的计算图的前馈性质与向量作为常见的输入和输出相结合，复杂的网络可以作为模块以创新的方式组合使用。本节介绍了RNN在语言处理中的两种常见网络架构。

8.4.1 堆叠RNN

在我们目前的示例中，RNN的输入由词或字符嵌入（向量）序列组成，输出则是对预测词、标签或序列标签有用的向量。然而，没有任何限制阻止我们将一个RNN的整个输出序列作为另一个RNN的输入序列。堆叠RNN由多个网络组成，其中一层的输出作为下一层的输入，如图8.10所示。

堆叠RNN通常比单层网络表现更好。这种成功的一个原因似乎在于网络在不同的层次上诱导了不同抽象水平的表示。就像人类视觉系统的早期阶段检测边缘，然后用这些边缘来发现更大的区域和形状一样，堆叠网络的初始层可以诱导出用于进一步层次的抽象表示——这些表示在单个RNN中可能难以诱导。堆叠RNN的最佳层数取决于每个应用和训练集。然而，随着堆叠层数的增加，训练成本会迅速上升。

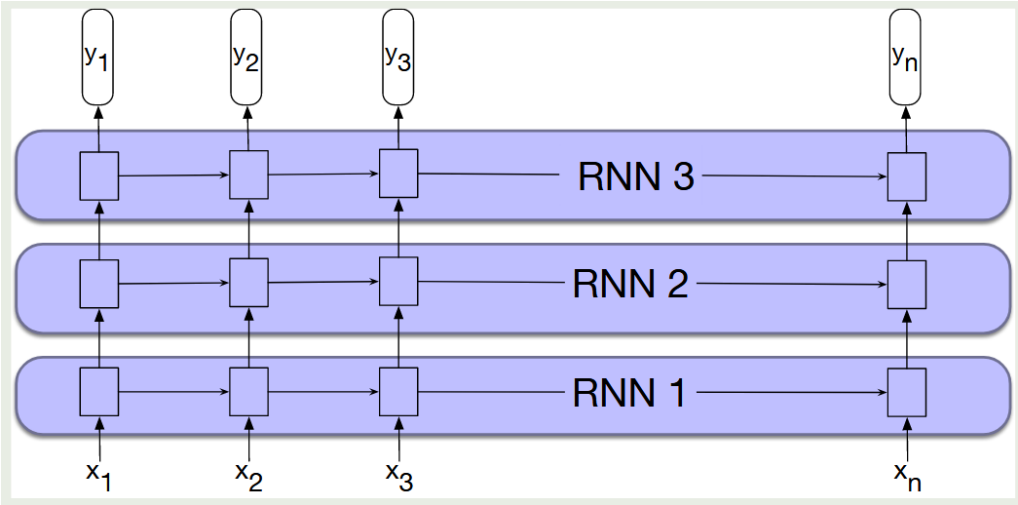

图 8.10 堆叠循环网络。低层网络的输出作为高层网络的输入，最后一层网络的输出作为最终的输出。

8.4.2 双向RNN

RNN使用来自左侧（先前）上下文的信息来进行时间步t的预测。但是，在许多应用中，我们可以访问整个输入序列；在这些情况下，我们希望使用来自时间步t右侧上下文的词。实现这一目标的一种方法是运行两个独立的RNN，一个从左到右，另一个从右到左，然后将它们的表示拼接在一起。

在我们讨论的左到右的RNN中，给定时间步t的隐藏状态表示网络到那时为止关于序列所知道的一切。该状态是输入x1,...,xt 的函数，表示网络在当前时间步左侧的上下文。

$$\mathbf{h}_t^f = \text{RNN}_{\text{forward}}(\mathbf{x}_1, \ldots, \mathbf{x}_t)$$

新符号 $\mathbf{h}_t^f$ 对应于时间步t的正常隐藏状态，代表网络到目前为止从序列中获取的所有信息。

为了利用当前输入右侧的上下文，我们可以在反向输入序列上训练一个RNN。在这种方法中，时间步t的隐藏状态表示关于当前输入右侧序列的信息：

$$\mathbf{h}_t^b = \text{RNN}_{\text{backward}}(\mathbf{x}_t, \ldots \mathbf{x}_n)$$

此时，隐藏状态 $\mathbf{h}_t^b$ 表示我们从时间步t到序列结束所获取的所有信息。

双向RNN（Schuster和Paliwal, 1997）结合了两个独立的RNN，一个从序列的开始处理到结束，另一个从结束处理到开始。然后，我们将这两个网络计算的表示拼接成一个向量，该向量捕捉了输入的左侧和右侧上下文。这里我们使用“;”或者直和符号 $\oplus$ 来表示向量拼接：

$$\begin{aligned}\mathbf{h}_t &= [\mathbf{h}_t^f; \mathbf{h}_t^b] \\ &= \mathbf{h}_t^f \oplus \mathbf{h}_t^b\end{aligned}$$

图8.11展示了这样一个双向网络，它将前向和后向传递的输出拼接在一起。其他简单的组合前后上下文的方式包括元素级的加法或乘法。因此，每个时间步的输出既捕捉了当前输入左侧的信息，也捕捉了右侧的信息。在序列标注应用中，这些拼接的输出可以作为局部标注决策的基础。

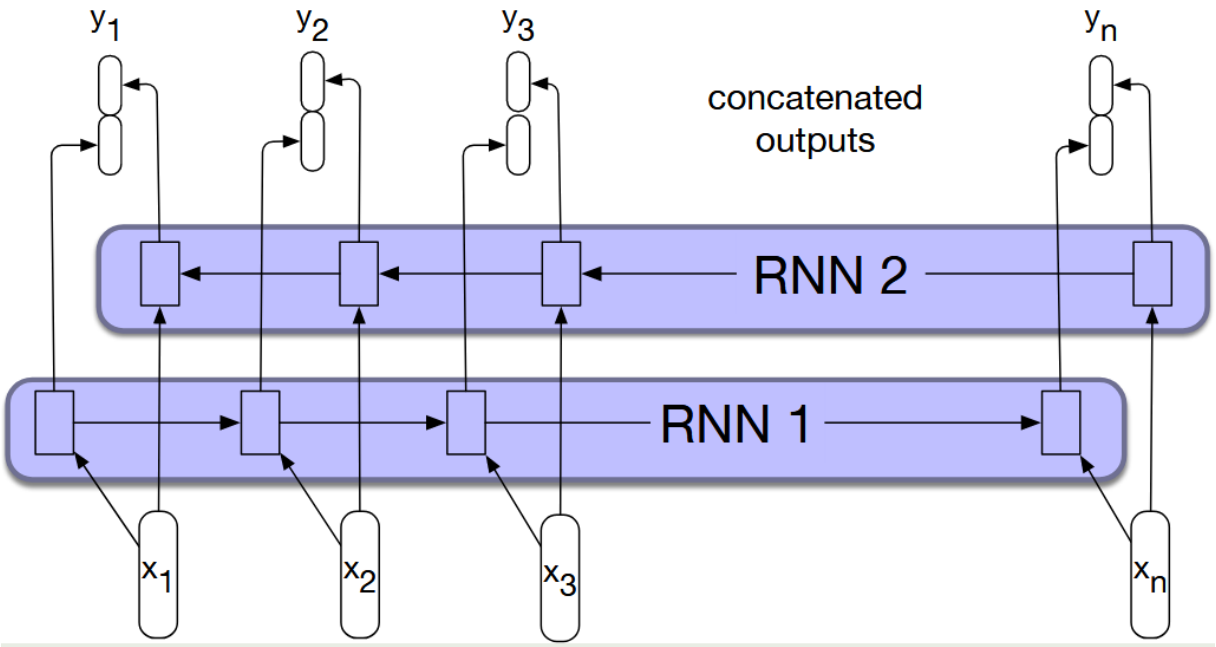

图 8.11 双向RNN。在前向和后向两个方向上分别训练单独的模型，将每个模型在每个时间点的输出串联起来，以表示该时间点的双向状态。

双向RNN在序列分类中也被证明非常有效。回想一下图8.8，对于序列分类，我们使用RNN的最终隐藏状态作为随后的前馈分类器的输入。这种方法的一个难点在于，最终状态自然会反映更多关于句子末尾的信息，而不是开头。双向RNN为此问题提供了一个简单的解决方案；如图8.12所示，我们只需将前向和后向传递的最终隐藏状态（例如通过拼接）组合起来，并将其用作后续处理的输入。

8.5 长短期记忆网络（LSTM）

在实际操作中，训练RNN来处理需要利用距离当前处理点较远信息的任务是相当困难的。尽管RNN可以访问整个前面的序列，但隐藏状态中编码的信息往往是局部的，更多地与输入序列最近的部分和最近的决策相关。然而，许多语言应用中远距离的信息是至关重要的。用下面的语言建模为例：

The flights the airline was canceling were full.

预测“was”出现在“airline”之后的概率很容易，因为“airline”为单数动词提供了强有力的局部上下文。然而，给“were”分配一个合适的概率则很困难，不仅因为复数名词“flights”距离较远，还因为单数名词“airline”在中间更接近。理想情况下，网络应该能够保留远距离的复数信息“flights”，直到需要时使用，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在于，隐藏层（以及决定隐藏层值的权重）同时承担了两个任务：提供对当前决策有用的信息，并更新和传递对未来决策所需的信息。

训练RNN的另一个难点来自需要将误差信号按时间顺序反向传播。回想一下第8.1.2节中提到的，时间步 t*t*的隐藏层对下一个时间步的损失有贡献，因为它参与了该计算。这导致在训练的反向传播过

程中，隐藏层需要进行多次相乘，这取决于序列的长度。这一过程的常见结果是梯度最终被缩小到接近零的情况，称为**梯度消失**问题。

为了解决这些问题，设计了更复杂的网络架构，明确管理随时间变化的上下文任务，允许网络学习忘记不再需要的信息，并记住为未来决策所需的信息。

最常用的这种RNN扩展是**长短期记忆网络**（LSTM）（Hochreiter和Schmidhuber, 1997）。LSTM将上下文管理问题分为两个子问题：移除不再需要的上下文信息，以及添加可能在之后决策中需要的信息。解决这两个问题的关键在于学习如何管理上下文，而不是将某种策略硬编码到架构中。LSTM通过首先在架构中添加一个显式的上下文层（除了通常的循环隐藏层），以及通过使用特殊的神经单元（这些单元使用门控机制来控制信息流入和流出网络层中的单元）来实现。通过附加的权重，这些门控依次操作输入、前一隐藏层和前一上下文层。

LSTM中的门控共享一个通用的设计模式；每个门控由一个前馈层组成，后跟一个sigmoid激活函数，然后与被门控的层进行逐元素乘法。选择sigmoid作为激活函数是因为它倾向于将输出推向0或1。将其与逐元素乘法结合的效果类似于二进制掩码。与掩码中接近1的值对齐的层中的值几乎不变地通过；对应于较低值的部分则基本被抹去。

我们将首先考虑遗忘门（forget gate）。这个门控的目的是删除上下文中不再需要的信息。遗忘门计算前一状态的隐藏层和当前输入的加权和，并通过sigmoid函数。这一掩码然后与上下文向量逐元素相乘，以移除上下文中不再需要的信息。两向量的逐元素乘法（表示为 $\odot$，有时称为Hadamard积）是与输入向量相同维度的向量，其中每个元素 i 是两个输入向量中第 i个元素的乘积:

$$\mathbf{f}_t = \sigma(\mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{W}_f \mathbf{x}_t)$$
$$\mathbf{k}_t = \mathbf{c}_{t-1} \odot \mathbf{f}_t$$

接下来要执行的任务是计算我们需要从前一隐藏状态和当前输入中提取的实际信息——这是我们在所有循环网络中使用的基本计算：

$$\mathbf{g}_t = \tanh(\mathbf{U}_g \mathbf{h}_{t-1} + \mathbf{W}_g \mathbf{x}_t)$$

然后，我们生成添加门（add gate）的掩码，以选择需要添加到当前上下文中的信息。

$$\mathbf{i}_t = \sigma(\mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{W}_i \mathbf{x}_t)$$
$$\mathbf{j}_t = \mathbf{g}_t \odot \mathbf{i}_t$$

接下来，我们将其添加到已修改的上下文向量中，得到新的上下文向量。

$$\mathbf{c}_t = \mathbf{j}_t + \mathbf{k}_t$$

最后一个门控是输出门（output gate），它用于决定当前隐藏状态所需的信息（与需要为未来决策保留的信息相对）。

$$\mathbf{o}_t = \sigma(\mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{W}_o \mathbf{x}_t)$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

通过为各种门控分配适当的权重，LSTM接受上下文层、前一时间步的隐藏层和当前输入向量作为输入。然后，它生成更新后的上下文和隐藏向量作为输出。

LSTM在每个时间步的输出是隐藏状态 ht，这一输出可以作为堆叠RNN中后续层的输入，或者在网络的最终层使用 ht 作为LSTM的最终输出。

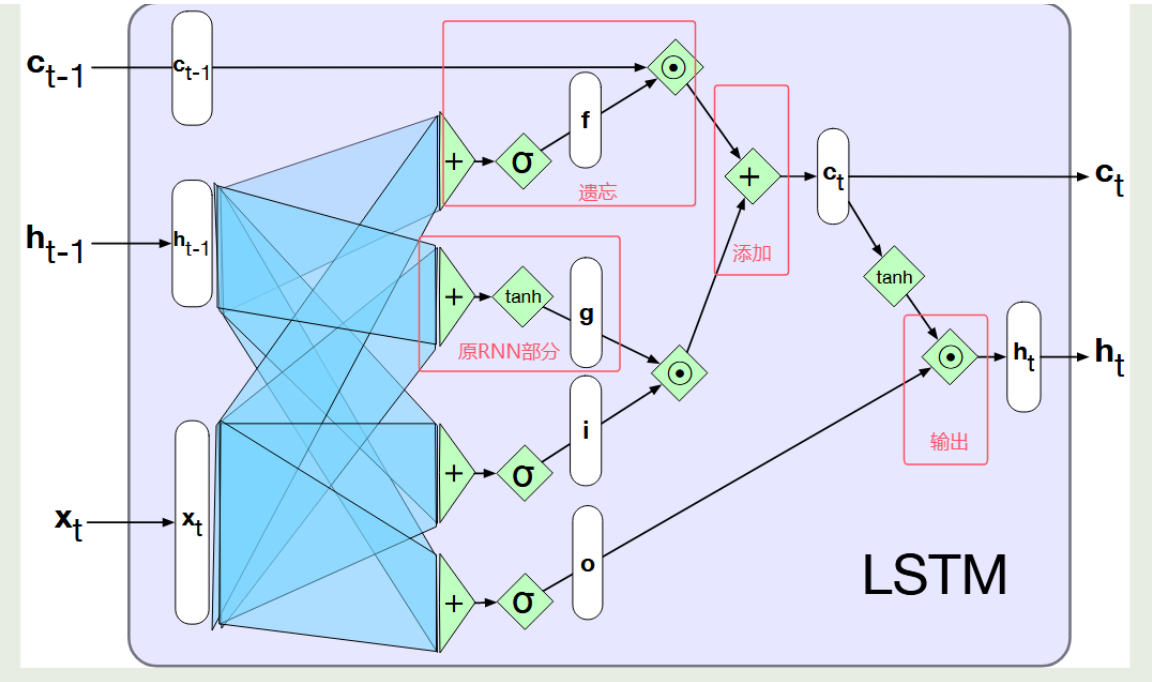

图 8.13 单个LSTM单元显示为计算图。每个单元的输入由当前输入，x，前一个隐藏状态，ht - 1和前一个上下文，ct - 1组成。输出是一个新的隐藏状态ht和一个更新的上下文ct。

8.5.1 门控单元、层和网络

LSTM中使用的神经单元显然比基本的前馈网络中使用的单元复杂得多。幸运的是，这种复杂性被封装在基本的处理单元中，使我们能够保持模块化，并轻松尝试不同的架构。为了理解这一点，请参考图 8.14，它展示了每种单元的输入和输出。

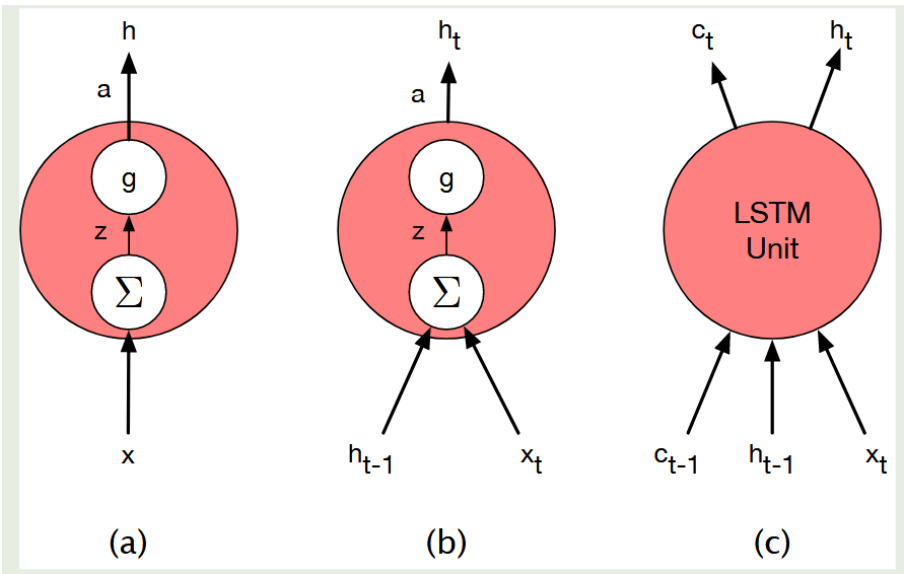

图 8.14 前馈、简单循环网络(SRN)和长短期记忆网络(LSTM)中使用的基本神经单元。

在最左边，（a）是基本的前馈单元，它使用一组权重和一个激活函数来确定输出，当这些单元排列在一层时，层中的单元之间没有连接。接下来，（b）代表简单循环网络中的单元。现在有了两个输入和一组额外的权重。然而，仍然只有一个激活函数和输出。

LSTM单元的复杂性被封装在单元本身。相对于基本的循环单元（b），LSTM的唯一额外的外部复杂性是存在额外的上下文向量作为输入和输出。

这种模块化是LSTM单元强大功能和广泛适用性的关键。LSTM单元（或其他变体，如GRU）可以替换到第8.4节描述的任何网络架构中。而且，像简单RNN一样，使用门控单元的多层网络可以展开为深度前馈网络，并以常规的方式通过反向传播进行训练。因此，在实践中，LSTM已成为任何使用循环网络的现代系统的标准单元，而不是RNN。

8.6 总结：常见的RNN NLP架构

我们已经介绍了RNN，了解了如堆叠多层和使用LSTM版本等高级组件，并且看到了RNN如何应用于各种任务。让我们花点时间总结这些应用的架构。

图8.15展示了我们迄今讨论的三种架构：序列标注、序列分类和语言建模。在序列标注中（例如词性标注），我们训练模型为每个输入词或标记生成一个标签。在序列分类中，例如情感分析，我们忽略每个标记的输出，只取序列末尾的值（类似地，模型的训练信号来自最后一个标记的反向传播）。在语言建模中，我们训练模型在每个标记步骤预测下一个词。在下一节中，我们将介绍第四种架构：编码器-解码器。

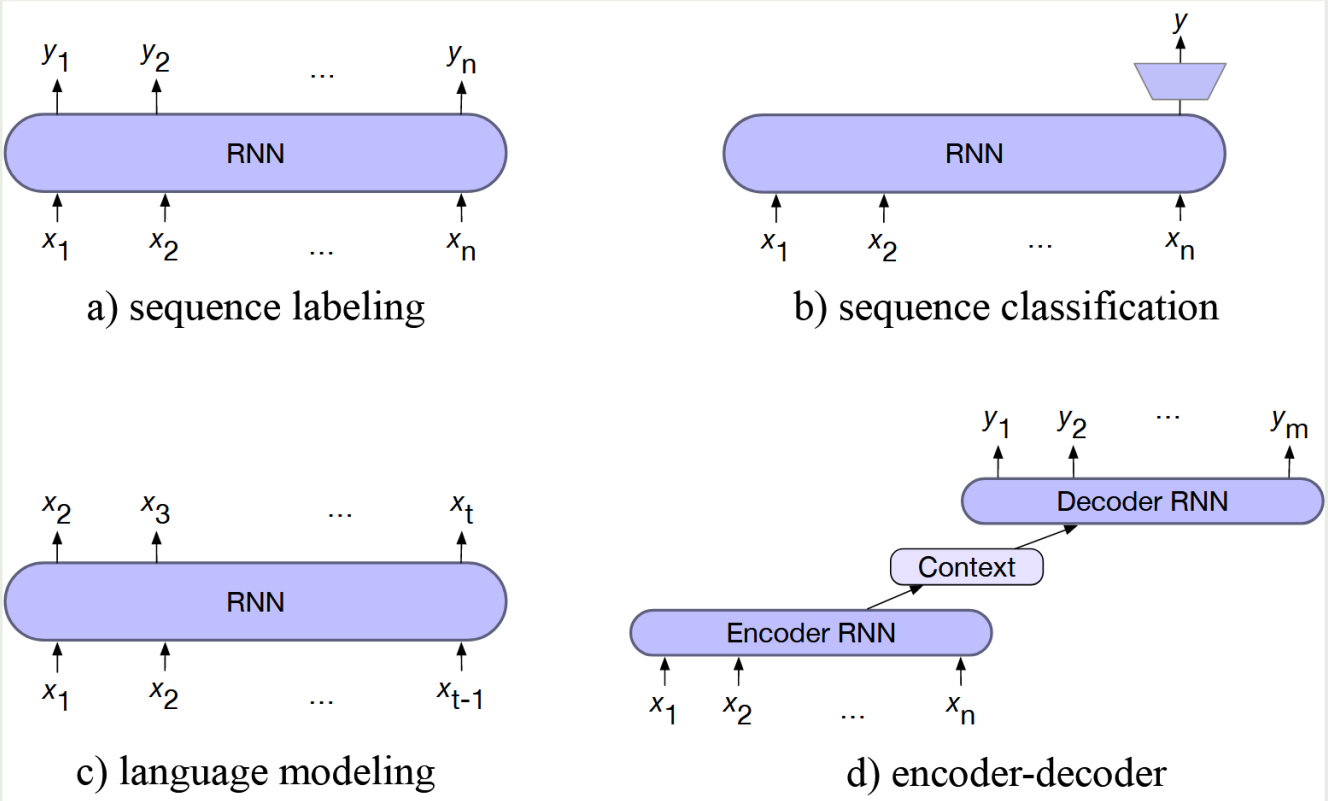

图 8.15 面向NLP任务的四种架构。在顺序标记(词性或命名实体标注)时，我们将每个输入令牌xi映射到一个输出令牌yi。在序列分类中，我们将整个输入序列映射到单个类别。在语言建模中，我们根据前面的标记输出下一个标记。在编码器模型中，我们有两个独立的RNN模型，其中一个从输入序列x映射到我们称之为上下文的中间表示，另一个从上下文映射到输出序列y。

8.7 RNN的编码器-解码器模型

在本节中，我们介绍一种新模型，即编码器-解码器模型，当我们将一个输入序列翻译成一个长度不同且不与输入一一对应的输出序列时使用。回顾在序列标注任务中，我们有两个序列，但它们长度相同

（例如在词性标注中，每个标记都有一个相关的标签），每个输入与一个特定输出相关联，且输出的标注主要依赖于局部信息。因此，决定一个词是动词还是名词时，主要查看该词及其邻近词。

相比之下，编码器-解码器模型主要用于像机器翻译这样的任务，在这些任务中，输入序列和输出序列可能具有不同的长度，并且输入中的一个标记与输出中的一个标记的映射可能非常间接（例如，在某些语言中，动词出现在句子的开头，而在另一些语言中则出现在末尾）。我们将在第13章详细介绍机器翻译，但目前只需指出，从英文句子翻译为他加禄语或约鲁巴语的句子可能会有非常不同的词数，并且词的顺序可能也非常不同。

编码器-解码器网络，有时称为序列到序列网络，是一类能够根据输入序列生成上下文合适的、任意长度的输出序列的模型。编码器-解码器网络已被应用于非常广泛的应用领域，包括摘要生成、问答系统和对话，但它们在机器翻译中尤为流行。

这些网络的关键思想是使用一个编码器网络，该网络接受一个输入序列并创建一个上下文化的表示，通常称为上下文（context）。然后将该表示传递给解码器，解码器生成特定任务的输出序列。图8.16展示了这种架构。

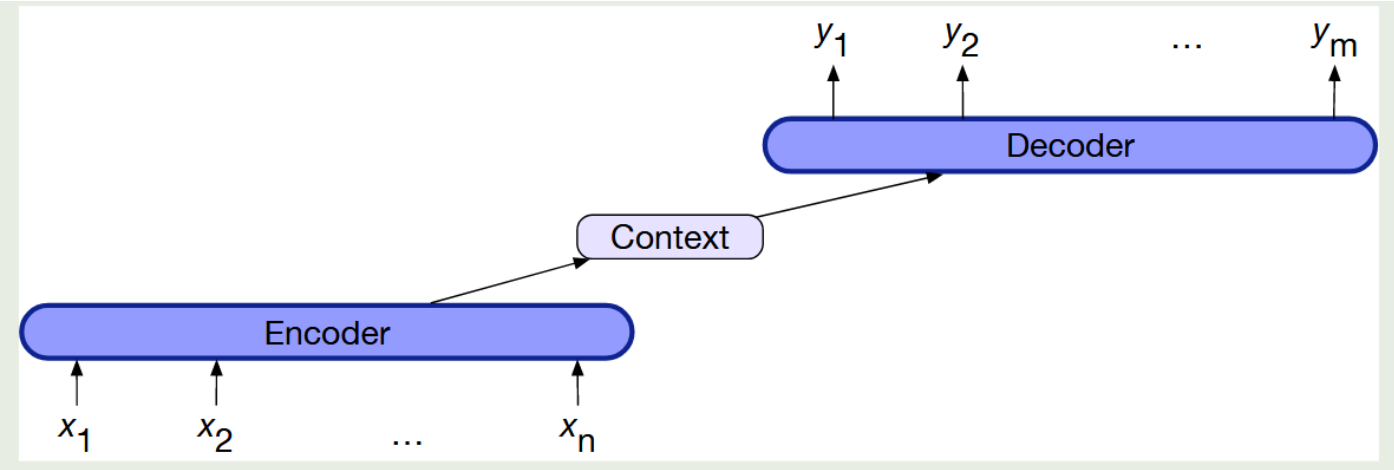

图 8.16 Encoder - Decoder架构。上下文是输入的隐藏表示的函数，并且可以通过多种方式被解码器使用。

编码器-解码器网络由三个组件组成：

1. 编码器：接受输入序列 x1:n，并生成相应的上下文化表示序列 h1:n。LSTM、卷积网络和Transformer都可以作为编码器。
2. 上下文向量（context vector）：它是 h1:n的函数，向解码器传达输入的精髓。
3. 解码器：接受上下文向量 c 作为输入，并生成一个任意长度的隐藏状态序列 h1:m*h*，从中可以获得相应的输出状态序列 y1:m。解码器也可以由任何序列架构实现。

在本节中，我们将描述基于一对RNN的编码器-解码器网络，但在第13章中我们将看到如何将其应用于Transformer。我们将从条件RNN语言模型 p(y)*p*（即序列 *y* 的概率）开始构建编码器-解码器模型的方程。

回想一下，在任何语言模型中，我们都按如下可以分解概率：

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

在RNN语言建模中，在特定时间 t，我们通过语言模型传递时间步 t－1 的前缀标记，使用前向推理生成一系列隐藏状态，最后以与前缀最后一个词对应的隐藏状态结束。然后，我们使用该前缀的最终隐藏状态作为起点，生成下一个标记。

更正式地说，如果 g 是像tanh或ReLU这样的激活函数，是时间步 t 的输入和时间步 t－1的隐藏状态的函数，softmax覆盖的是可能的词汇项集合，那么在时间步 t 的输出 yt 和隐藏状态 ht如下：

$$\mathbf{h}_t = g(\mathbf{h}_{t-1}, \mathbf{x}_t)$$
$$\hat{\mathbf{y}}_t = \text{softmax}(\mathbf{h}_t)$$

只需进行一项小改动，就可以将这个带有自回归生成的语言模型转换为编码器-解码器模型，使其成为一种翻译模型，可以从一种语言的源文本翻译成另一种语言的目标文本：在源文本末尾添加一个句子分隔符标记，然后简单地将目标文本连接起来。

我们使用 <s> 作为句子分隔符标记，并以将英语源文本（“the green witch arrived”）翻译成西班牙语句子（“llegó la bruja verde”）为例（逐词翻译为 ‘arrived the witch green’）。我们还可以用问答对或文本摘要对来展示编码器-解码器模型。

我们使用 x 表示源文本（此处为英文）加上分隔符标记 <s>，使用 y 表示目标文本（此处为西班牙文）。然后，编码器-解码器模型计算 p(y ∣ x) 的概率：

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

图8.17展示了编码器-解码器模型的简化版本的设置（我们将在下一节中看到完整模型，它需要一个新概念——注意力机制）。图8.17展示了英文源文本（“the green witch arrived”），一个句子分隔符标记（<s>），以及西班牙语目标文本（“llegó la bruja verde”）。要翻译源文本，我们通过网络进行前向推理生成隐藏状态，直到到达源输入的末尾。然后我们开始自回归生成，基于源输入末尾的隐藏层以及句子结束标记请求第一个词。后续词则依赖于之前的隐藏状态和最后生成词的嵌入。

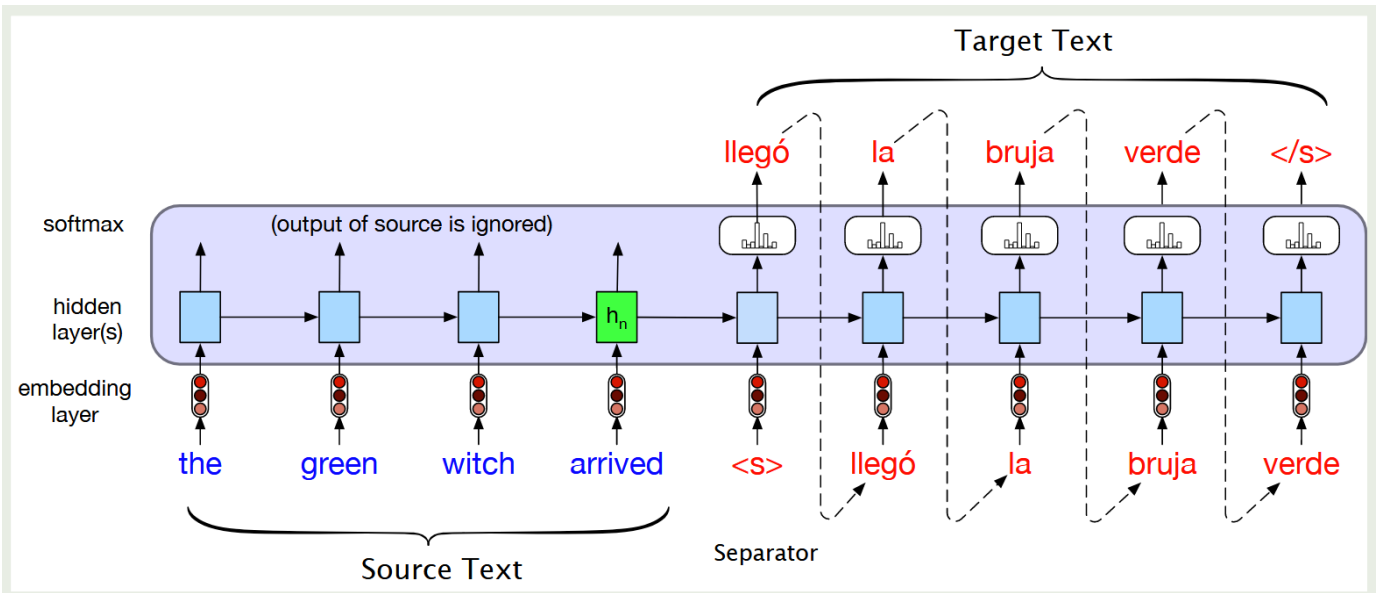

图 8.17 在编码器-解码器方法的基本RNN版本中翻译单个句子(推理时间)到机器翻译。源句和目标句之间用分隔符连接，解码器使用来自编码器最后一个隐藏状态的上下文信息。

让我们在图8.18中正式化并推广这个模型。（为了帮助理解，我们将在需要时使用上标 e 和 d 来区分编码器和解码器的隐藏状态。）左侧的网络元素处理输入序列 x 并构成编码器。虽然我们的简化图仅显示了编码器的一层网络，但堆叠架构是常态，**堆叠顶部层**的输出状态被视为最终表示，编码器

由堆叠的双向LSTM（biLSTM）组成，其中前向和后向传递的顶层隐藏状态被拼接以提供每个时间步的上下文化表示。

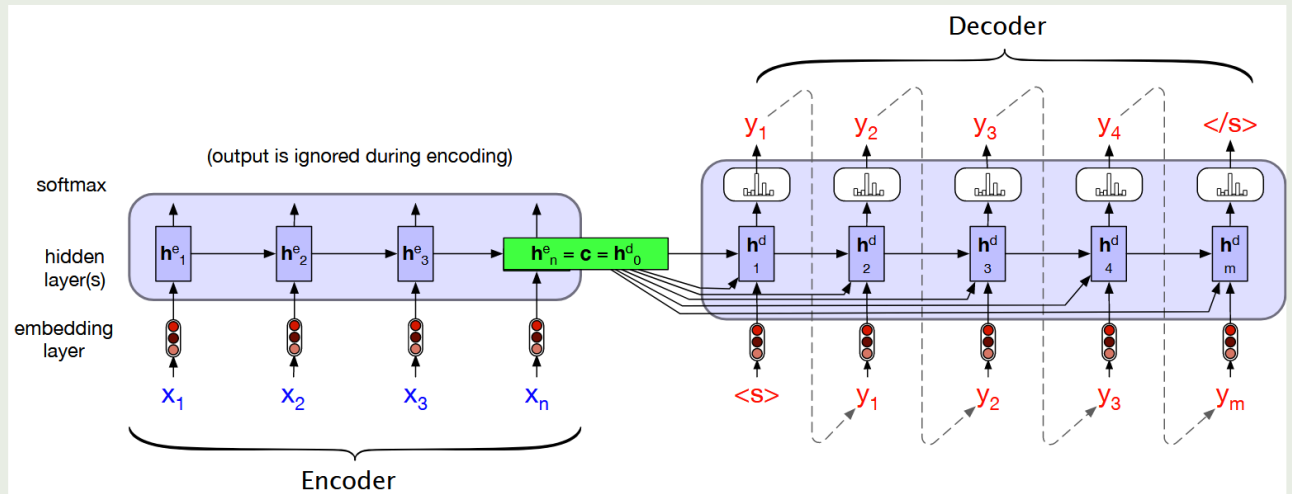

图 8.18 在基本的基于RNN的编码器-解码器架构中，在推理时翻译一个句子的更正式的版本。编码器RNN的最终隐藏状态hen作为解码器RNN中hd 0的角色作为解码器的上下文，也提供给每个解码器隐藏状态。

编码器的目的是生成输入的上下文化表示。这一表示体现在编码器的最终隐藏状态 $\mathbf{h}_n^e$ 中。这一表示，也称为上下文 c，然后传递给解码器。

最简单版本的解码器网络将使用这个状态来初始化解码器的第一个隐藏状态；第一个解码器RNN单元将使用上下文 c 作为其先前隐藏状态 $\mathbf{h}_0^d$ 。然后解码器以自回归方式生成输出序列，直到生成序列结束标记为止。每个隐藏状态依赖于之前的隐藏状态和在前一状态生成的输出。

如图8.18所示，我们采取了更复杂的方法：我们将上下文向量 c= 不仅提供给第一个解码器隐藏状态，还确保在生成输出序列时上下文向量 c 的影响不会减弱。我们通过将 c 添加为当前隐藏状态计算的一个参数来实现这一点。

现在，我们已经准备好看到该版本的解码器的完整方程，在基本编码器-解码器模型中，上下文在每个解码时间步都可用。回想一下，g 是RNN的代表， $\hat{y}_{t-1}$ 是在前一步从softmax采样的输出嵌入：

$$
\begin{aligned}
\mathbf{c} &= \mathbf{h}_n^e \\
\mathbf{h}_0^d &= \mathbf{c} \\
\mathbf{h}_t^d &= g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c}) \\
\hat{\mathbf{y}}_t &= \text{softmax}(\mathbf{h}_t^d)
\end{aligned}
$$

因此， $\hat{y}_t$ 是一个词汇表上各个词出现概率的向量，表示在时间步 t 发生每个词的概率。为了生成文本，我们从分布 $\hat{y}_t$ 中采样。例如，贪婪的选择是每个时间步都选择最可能生成的词。我们将在第??节介绍更复杂的采样方法.

8.7.1 训练编码器-解码器模型

编码器-解码器架构是端到端训练的。每个训练示例是一个成对的字符串元组，一个源字符串和一个目标字符串。将它们与分隔符标记连接后，这些源-目标对就可以作为训练数据。

对于机器翻译（MT），训练数据通常由一组句子及其翻译组成。这些数据可以从标准的对齐句子对数据集中获取，我们将在第??节中讨论。一旦有了训练集，训练过程就像任何基于RNN的语言模型一样进行。网络被输入源文本，然后从分隔符标记开始，进行自回归训练以预测下一个词，如图8.19所示。

请注意训练（图8.19）和推理（图8.17）在每个时间步的输出方面的区别。在推理时，解码器使用其自身估计的输出 $\hat{y}_t$ 作为下一个时间步 x_{t+1} 的输入。因此，解码器在生成更多标记时，往往会越来越偏离目标句子。在训练中，因此更常使用教师强制（teacher forcing）来训练解码器。教师强制意味着我们强制系统使用训练中的目标标记作为下一个输入 x_{t+1} ，而不是让它依赖于（可能是错误的）解码器输出 $\hat{y}_t$ 。这可以加快训练速度。

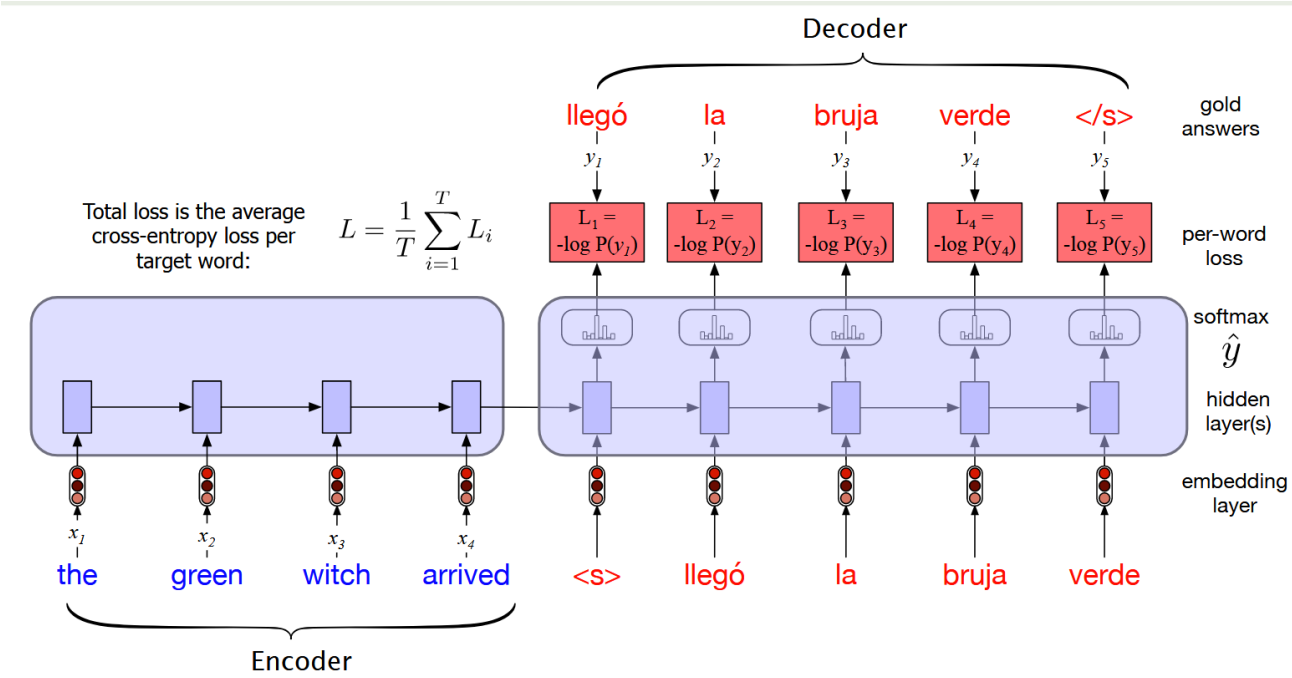

图 8.19 训练基本的RNN编码器-解码器方法进行机器翻译。注意到在解码器中我们通常不传播模型的softmax输出y (t，而是使用教师强制将每个输入强制到正确的gold值进行训练。我们在解码器中计算y (上的softmax输出分布，以便计算每个token处的损失，然后将其平均以计算句子的损失。然后这种损失通过解码器参数和编码器参数进行传播。

8.8 注意力机制

编码器-解码器模型的简洁性在于它将编码器和解码器分离开来——编码器构建源文本的表示，而解码器使用这个上下文生成目标文本。我们目前描述的模型中的上下文向量是源文本最后一个时间步（第n步）的隐藏状态 hn。因此，这个最终隐藏状态充当了一个瓶颈：它必须表示源文本的所有含义，因为解码器对源文本的了解仅限于这个上下文向量（如图8.20）。尤其对于长句子，句子开头的信息可能无法很好地被表示在上下文向量中。

注意力机制是解决这一瓶颈问题的一种方法，它允许解码器获取编码器所有隐藏状态的信息，而不仅仅是最后一个隐藏状态。

在注意力机制中，与普通的编码器-解码器模型一样，上下文向量 c 是编码器隐藏状态的函数，即 $\mathbf{c} = f(\mathbf{h}_1^e \ldots \mathbf{h}_n^e)$。由于隐藏状态的数量随输入大小变化，我们不能直接使用编码器隐藏状态的整个集合作为解码器的上下文。

注意力的想法是通过对所有编码器隐藏状态进行加权求和，创建单个固定长度的向量 c。这些权重集中在与解码器当前正在生成的标记相关的源文本部分。注意力机制因此用一个动态生成的上下文向量替代了静态上下文向量，每个解码步骤的上下文都不相同。

这个上下文向量 $\mathbf{c}_i$ 在每个解码步骤 i 都重新生成，并在推导过程中考虑了所有编码器隐藏状态。然后，通过将当前解码器隐藏状态的计算条件化在这个上下文向量上（以及前一个隐藏状态和解码器之前生成的输出），我们使上下文在解码过程中可用：

$$\mathbf{h}_i^d = g(\hat{y}_{i-1}, \mathbf{h}_{i-1}^d, \mathbf{c}_i)$$

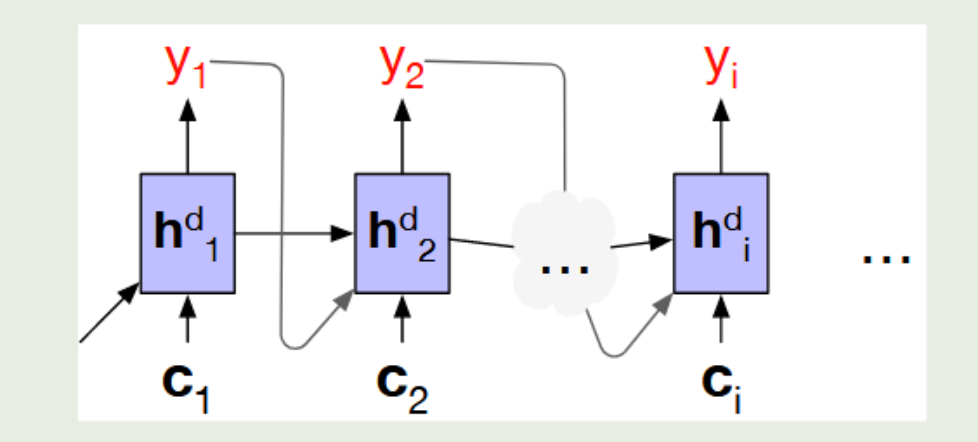

图 8.21注意力机制允许解码器的每个隐藏状态看到一个不同的、动态的、上下文的，它是所有编码器隐藏状态的函数。

计算 $\mathbf{c}_i$ 的第一步是确定要对每个编码器状态关注多少，即每个编码器状态与捕获在解码器状态 $\mathbf{h}_{i-1}^d$ 中的相关性。在每个解码步骤 i，我们通过计算每个编码器状态 j 的得分 $score(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e)$ 来衡量其相关性。

最简单的得分方式称为**点积注意力**（dot-product attention），它通过计算解码器隐藏状态和编码器隐藏状态之间的点积来衡量相关性，即相似度：

$$\text{score}(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e) = \mathbf{h}_{i-1}^d \cdot \mathbf{h}_j^e$$

点积计算的得分是一个标量，反映了两个向量的相似度。这些得分的向量反映了所有编码器隐藏状态在解码器当前步骤中的相关性。

为了使用这些得分，我们将它们通过softmax归一化，生成一个权重向量 α_{ij}，它告诉我们每个编码器隐藏状态 j 与之前解码器隐藏状态 $\mathbf{h}_{i-1}^d$ 的相关性比例。

最后，给定权重分布 α，我们可以通过对所有编码器隐藏状态进行加权平均，计算出当前解码器状态的固定长度上下文向量。

$$\mathbf{c}_i = \sum_j \alpha_{ij} \mathbf{h}_j^e$$

通过这样做，我们最终得到了一个固定长度的上下文向量，该向量考虑了编码器状态的全部信息，并根据每一步解码的需要动态更新。图8.22展示了一个带有注意力机制的编码器-解码器网络，着重于上下文向量 ci 的计算。

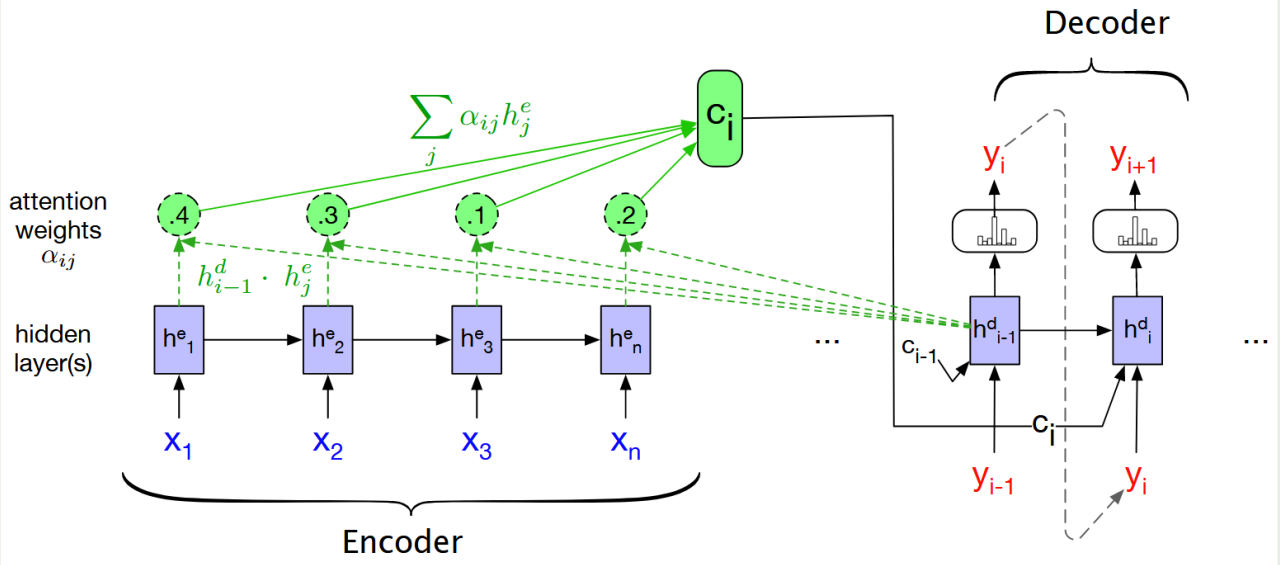

图 8.22 带注意力的编码器-解码器网络示意图，重点关注ci的计算。上下文值ci是计算hd i的输入之一。它是通过取所有编码器隐藏状态的加权和来计算的，每个隐藏状态的加权是它们与先验解码器隐藏状态hd i - 1的点积。

我们还可以为注意力模型创建更复杂的评分函数。与简单的点积注意力不同，我们可以通过用自己的权重 $\mathbf{W}_s$ 参数化得分，获得一个更强大的函数，来计算每个编码器隐藏状态与解码器隐藏状态的相关性。

$$\text{score}(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e) = \mathbf{h}_{t-1}^d \mathbf{W}_s \mathbf{h}_j^e$$

这些权重 $\mathbf{W}_s$ 在正常的端到端训练过程中被学习，它们使得网络能够学习到当前应用中哪些相似性是重要的。这种双线性模型（bilinear model）还允许编码器和解码器使用不同维度的向量，而简单的点积注意力要求编码器和解码器隐藏状态具有相同的维度。

我们将在第9章中回到注意力的概念，届时我们将定义基于注意力的Transformer架构，它基于一种稍作修改的注意力机制，称为自注意力（self-attention）。

8.9 总结

本章介绍了循环神经网络（RNN）的概念以及它们如何应用于语言问题。以下是我们涵盖的主要内容的总结：

- 在简单的循环神经网络（RNN）中，序列是按元素依次处理的，时间步 t 上每个神经单元的输出基于当前时间 t 的输入以及时间 t－1 的隐藏层状态。
- RNN可以通过反向传播算法的一个简单扩展来训练，这个扩展称为时间上的反向传播（Backpropagation Through Time，BPTT）。
- 简单的循环网络在处理长输入时会失败，原因包括梯度消失等问题；现代系统使用更复杂的门控架构，例如LSTM，这些架构明确地决定隐藏层和上下文层中哪些信息需要记住，哪些需要忘记。
- RNN常见的基于语言的应用包括：
  - 概率语言建模：为一个序列分配概率，或为给定前面单词的下一个元素分配概率。
  - 使用训练好的语言模型进行自回归生成。

- 序列标注，如词性标注，其中为序列的每个元素分配一个标签。
- 序列分类，如垃圾邮件检测、情感分析或主题分类，将整个文本分配到某个类别。
- 编码器-解码器架构，用于将输入映射到长度和对齐方式不同的输出。