**Chapters 9. RNNs and LSTMs**

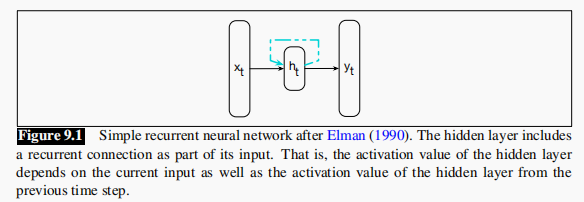
语言是一种随时间展开的序列，无论是口语还是书面语，我们都将其视为连续的输入流。这种时间性在我们使用的比喻中得到反映，比如我们会提到对话的流动、新闻源和推特流。一些语言处理算法，如隐马尔可夫模型(HMM)的部分词性标记中使用的Viterbi算法，就体现了这种时间性，它通过逐词处理输入，沿途携带信息前进。然而，其他机器学习方法，如情感分析或文本分类任务的方法，并不考虑时间性，它们假设可以同时访问输入的所有方面。

循环神经网络(RNN)及其变种，如长短期记忆网络(LSTMs)。RNNs通过其循环连接直接处理语言的序列性，允许模型在不使用任意固定大小窗口的情况下处理语言的时间性。这种网络提供了一种新的方式来表示之前的上下文，使模型的决策能够依赖于过去数百个词的信息。文中还讨论了如何将这种模型应用于语言建模、序列建模任务如词性标记，以及文本分类任务如情感分析。

**9.1 Recurrent Neural Networks**

循环神经网络（RNN）定义：RNN是包含循环连接的网络，这意味着某些单元的值直接或间接地依赖于其先前输出作为输入。尽管RNN强大，但理解和训练它们较为困难。Elman网络（或简单的循环网络）是RNN中的一类，因其在语言处理中的有效性而被广泛研究和应用。

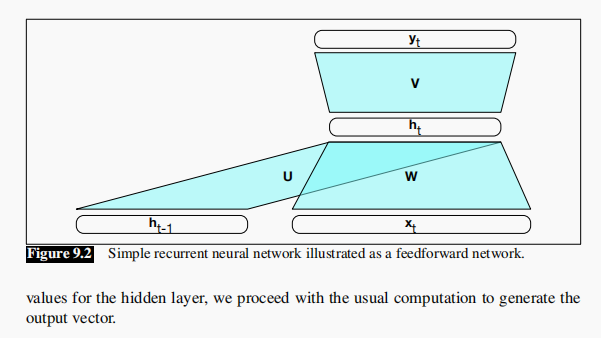
Elman网络结构：Elman网络通过在隐藏层中加入循环连接来处理序列数据，使得隐藏层的激活值不仅依赖于当前输入，还依赖于上一时间步的隐藏层激活值。这种结构使网络能够记忆和利用过去的信息。



RNN的推理（Inference）过程：

RNN的推理过程与前馈网络类似，但关键区别在于RNN引入了循环链接，将上一时间步的隐藏层值作为当前计算的一部分。

对于输入向量xt和上一时间步的隐藏层值ht-1，RNN通过结合这两个信息并通过激活函数计算当前隐藏层的激活值ht，然后基于ht计算输出yt。



训练RNN：

训练RNN涉及到使用损失函数和反向传播算法来更新权重。RNN具有三组权重：W（输入层到隐藏层）、U（上一隐藏层到当前隐藏层）、V（隐藏层到输出层）。

为了计算输出在时间t的损失，需要从时间t-1的隐藏层值开始，逐步向前计算，直到序列结束。这个过程称为时间上的反向传播（BPTT）。

在训练过程中，首先通过前向传播计算各时间步的隐藏层和输出，累积损失；然后通过反向传播计算梯度，并更新权重。

应对长序列问题：

对于长序列数据，如语音识别或字符级处理，直接展开整个序列进行训练可能不可行。解决方案是将输入序列分解成可管理的固定长度段，每个段作为独立的训练项处理。

**9.2 RNNs as Language Models**

RNN作为语言模型

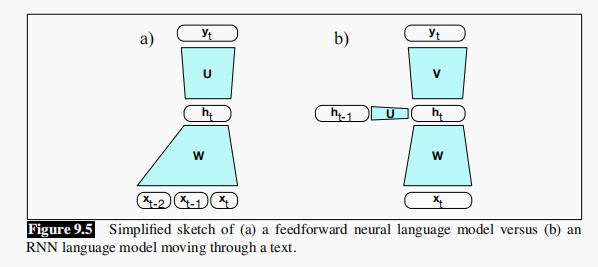
基本概念：语言模型的任务是在给定一些前置上下文的情况下，预测序列中的下一个词。RNN语言模型通过逐字处理输入序列，使用当前单词和先前的隐藏状态来预测下一个单词，从而不受n-gram模型的限制上下文问题或前馈语言模型的固定上下文问题的影响。

前向推理：

输入序列由一系列词组成，每个词通过词嵌入矩阵转换成一维向量。

每一步中，模型使用词嵌入矩阵来检索当前词的嵌入，然后与上一步的隐藏层结合来计算新的隐藏层。

新的隐藏层用于生成输出层，通过softmax层生成整个词汇表上的概率分布。

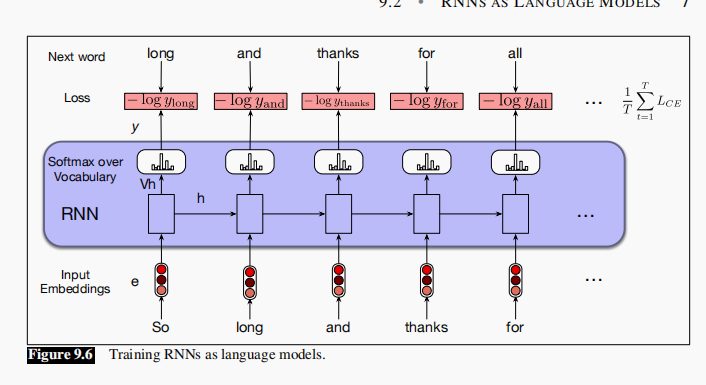


训练RNN语言模型：

使用自监督（自训练）算法：利用文本语料库作为训练材料，在每个时间步骤t要求模型预测下一个词。

训练模型以最小化训练序列中真实下一个词预测的误差，使用交叉熵作为损失函数。

实施"教师强制"策略，即总是提供正确的历史序列给模型来预测下一个词，而不是使用模型上一时间步的预测结果。



权重绑定（Weight Tying）

概念：注意到输入嵌入矩阵E和最终层矩阵V（提供softmax输出）在功能上非常相似。两者都提供了词汇的嵌入表示，只是在计算流程的不同阶段使用。

实施方式：权重绑定技术通过在输入层和softmax层使用同一组嵌入来消除这种冗余，即放弃使用V而只使用E完成计算。

好处：这种方法不仅提高了模型的复杂性，还显著减少了模型所需的参数数量。

**9.3 RNNs for other NLP tasks**

序列标注

定义：序列标注是指为序列中的每个元素（如单词或字符）分配一个固定标签集中的标签的任务。

RNN应用：通过使用预训练的词嵌入作为输入，RNN模型输出每个时间步的标签概率分布，利用softmax层进行计算。在训练过程中，使用交叉熵损失函数。

实例：词性标注和命名实体识别是序列标注的典型例子。

序列分类

定义：序列分类涉及将整个文本序列分类到一个或多个类别中，如情感分析或垃圾邮件检测。

RNN应用：文本通过RNN逐字处理，使用最后一个单词的隐藏状态作为整个序列的压缩表示。这个表示然后通过一个前馈网络传递，使用softmax进行分类。

变体：除了使用最后一个隐藏状态，还可以通过对所有隐藏状态应用池化（如取均值或最大值）来获取序列的表示。

文本生成

重要性：文本生成在诸如机器翻译、文本摘要、语法纠正、故事生成和对话生成等任务中具有巨大的实用价值，是生成性AI的重要组成部分。

方法：使用RNN基于语言模型逐步生成文本，通过重复采样下一个单词的方式，基于之前的选择条件，这被称为自回归生成或因果LM生成。

自回归生成：一个自回归模型基于时间t之前的值来预测时间t的值。尽管语言模型不是线性的，但生成技术因为每一步生成的单词都是基于网络在前一步选择的单词而被条件化，故仍然被称为自回归生成。

综合应用

RNN模型的这些应用展示了其在处理序列数据时的灵活性和效率，无论是为序列中的每个元素分配标签、分类整个序列，还是生成新的文本内容。

通过精确调整和训练，RNN可以适应多种NLP任务，使其成为处理语言数据的强大工具。

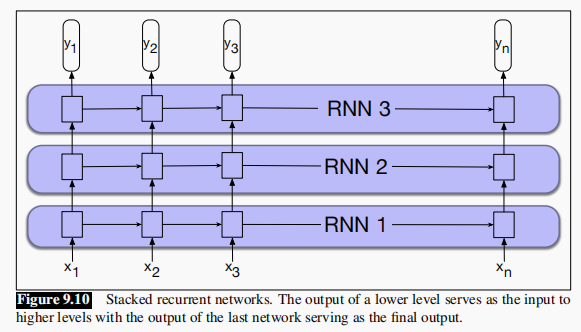
**9.4 Stacked and Bidirectional RNN architectures**

堆叠RNN（Stacked RNNs）

基本概念：堆叠RNN由多层RNN构成，其中一层的输出作为下一层的输入。

性能提升：相比单层网络，堆叠RNN通常表现得更好，原因之一是网络能在不同层次上引入不同级别的抽象表示，这些表示可能在单个RNN中难以学习到。

应用灵活性：堆叠的层数具体取决于应用和训练集，但层数增加会显著提高训练成本。



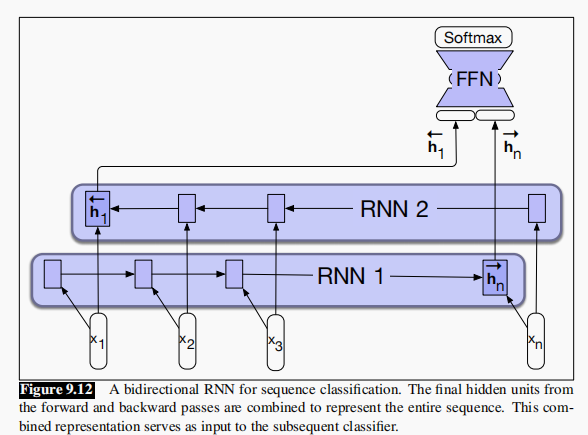
双向RNN（Bidirectional RNNs）

基本概念：双向RNN通过同时处理正向（从左到右）和反向（从右到左）的序列信息，使得每个时间点的表示既包含了左侧（之前）的上下文信息，也包含了右侧（之后）的上下文信息。

实现方式：运行两个独立的RNN，一个按正常顺序处理输入序列，另一个按反向顺序处理，然后将它们的表示合并成一个向量。

上下文利用：合并的表示捕获了当前输入点左右两侧的上下文信息，这对于序列标注和序列分类等任务特别有用。

特点：双向RNN特别适合于需要考虑全局上下文信息的应用，例如文本分类，其中通过合并最后一个时间点的前向和后向隐藏状态，可以获得反映整个序列信息的表示。



综合比较

堆叠RNN：通过增加网络深度来增强模型的表示能力，允许模型捕获更复杂的特征和层次结构。适用于需要高级抽象能力的场景。

双向RNN：通过考虑输入序列的前后上下文来增强模型的信息捕获能力，特别适合于上下文信息对预测结果影响显著的任务。

选择依据：选择哪种架构取决于具体任务的需求。对于需要理解整个序列上下文的任务（如文本分类），双向RNN更为合适；而对于需要深层次理解和抽象的任务（如复杂的语言理解或生成任务），堆叠RNN可能更优。

**9.5 The LSTM**

LSTM基础

背景问题：在实践中，训练RNN处理需要利用当前处理点远处信息的任务是相当困难的。虽然RNN能够访问前面的整个序列，但隐藏状态中编码的信息往往与输入序列的最近部分和近期决策更相关，而远处的信息对于许多语言应用至关重要。

双重任务的挑战：RNN的隐藏层需要同时完成两项任务——为当前决策提供信息和更新、携带对未来决策必要的信息。

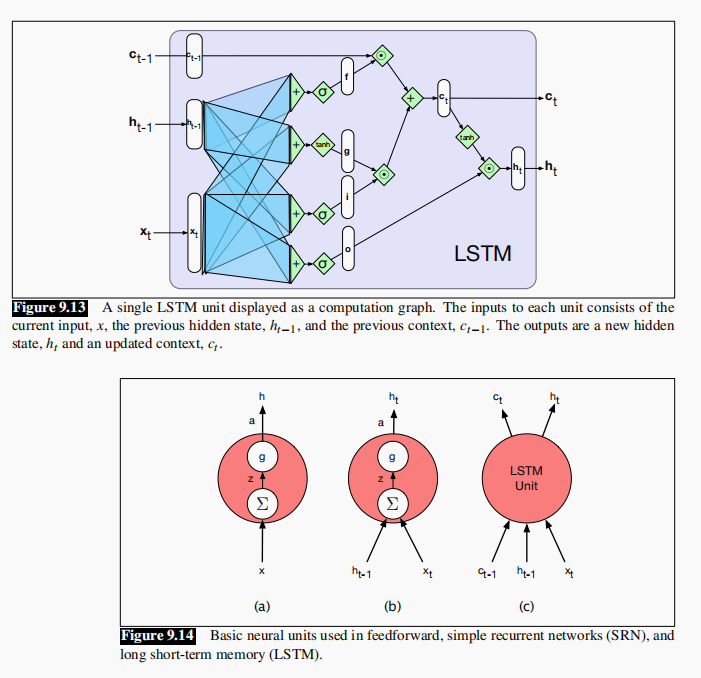
梯度消失问题：通过时间反向传播时，由于重复乘法，导致梯度最终趋向于零，这一情况被称为梯度消失问题。

LSTM架构

设计初衷：LSTM被设计用来显式地管理随时间维持相关上下文的任务，通过允许网络学会遗忘不再需要的信息和记住未来决策所需的信息。

关键特性：LSTM通过引入显式的上下文层（除了通常的循环隐藏层之外），并使用带有门控制机制的特殊神经单元来控制信息的流入和流出，来解决上下文管理问题。

门控制机制：LSTM中的门（如遗忘门、添加门和输出门）通过一系列的权重，依次作用于输入、先前的隐藏层和上下文层，使用sigmoid激活函数生成类似二进制掩码的效果，从而控制信息的保存或删除。



LSTM的计算过程

遗忘门：决定从上下文中删除哪些不再需要的信息。

添加门：计算从当前输入和先前隐藏状态中提取的实际信息。

更新上下文：将新计算的信息添加到修改后的上下文向量中，得到新的上下文向量。

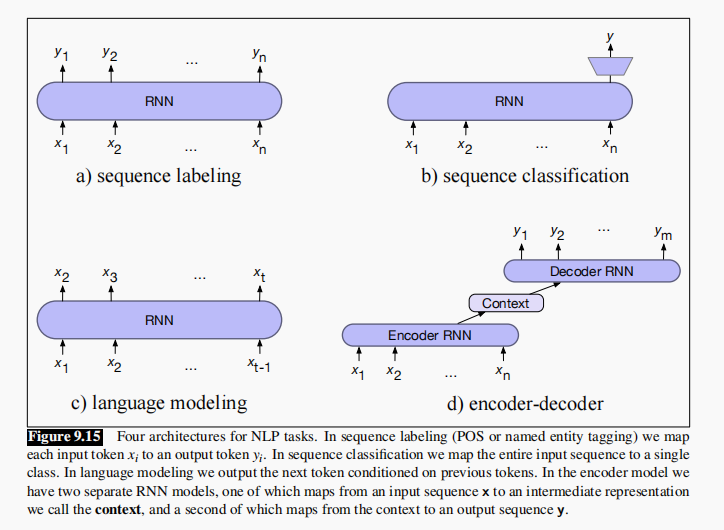
输出门：决定当前隐藏状态所需的信息（与未来决策保留的信息相对）。

LSTM的优势和应用

封装复杂性：尽管LSTM单元比基本的前馈网络单元复杂得多，但这种复杂性被封装在单元内部，使得LSTM保持了模块化，便于在不同架构中实验和应用。

模块化和通用性：LSTM单元的模块化设计是其强大和广泛应用性的关键。LSTM（或其他变体，如GRU）可以替换进任何网络架构中，包括前面提到的堆叠RNN和双向RNN中。

**9.6 Summary: Common RNN NLP Architectures**



a) 序列标注：这种架构用于词性标注（POS）或命名实体识别（NER）等任务。在这里，RNN为每个输入词或标记（x1至xn）分配一个输出标签（y1至yn）。这是一种多对多的关系，每个输入都映射到相应的输出。

b) 序列分类：用于情感分析等任务，其中整个输入序列（x1至xn）被用来将序列分类到类别中（例如，正面或负面情感）。RNN在序列结束时产生单一输出（y），反映了整个序列的分类。

c) 语言建模：在这种设置中，RNN根据先前的标记预测序列中的下一个标记。这种模型训练的目的是预测每个时间步的下一个词。

d) 编码器-解码器：这是一个两部分的架构，用于将输入序列（x1至xn）转换为一个新的输出序列（y1至ym）。在编码器阶段，RNN将输入序列转换成一个中间表示，称为上下文（context）。在解码器阶段，另一个RNN将这个上下文转换为输出序列。这种架构常用于机器翻译等任务，其中输入序列和输出序列之间的关系可能不是一对一的。

**9.7 The Encoder-Decoder Model with RNNs**

编码器-解码器模型概念

设计用途：当输入序列需要翻译成与其长度不同、且不按逐词对应方式对齐的输出序列时，就会使用编码器-解码器模型。

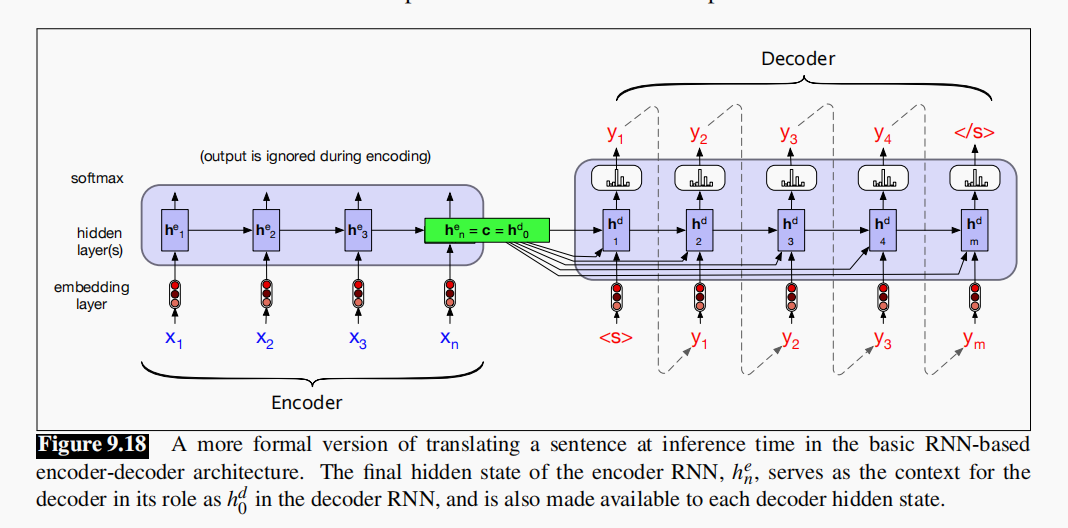
与序列标注的区别：在序列标注任务中，虽然有两个序列，但它们的长度相同，每个输入对应一个特定的输出，且标签决策主要依赖局部信息。相比之下，编码器-解码器模型适用于如机器翻译这样的任务，输入和输出序列的长度可能不同，且单词间的映射关系可能非常间接。

编码器-解码器网络构成

编码器：接受输入序列x1:n，生成一系列上下文化表示h1:n。可以使用LSTM、卷积网络或变换器作为编码器。

上下文向量：是h1:n的函数，携带输入序列的精华信息传递给解码器。

解码器：接受上下文向量作为输入，生成任意长度的隐藏状态序列h1:m，从中获取对应的输出状态序列y1:m。



编码器-解码器运作原理

上下文表示：编码器的整个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后一个隐藏状态he\_n中，也称为上下文c，随后传递给解码器。

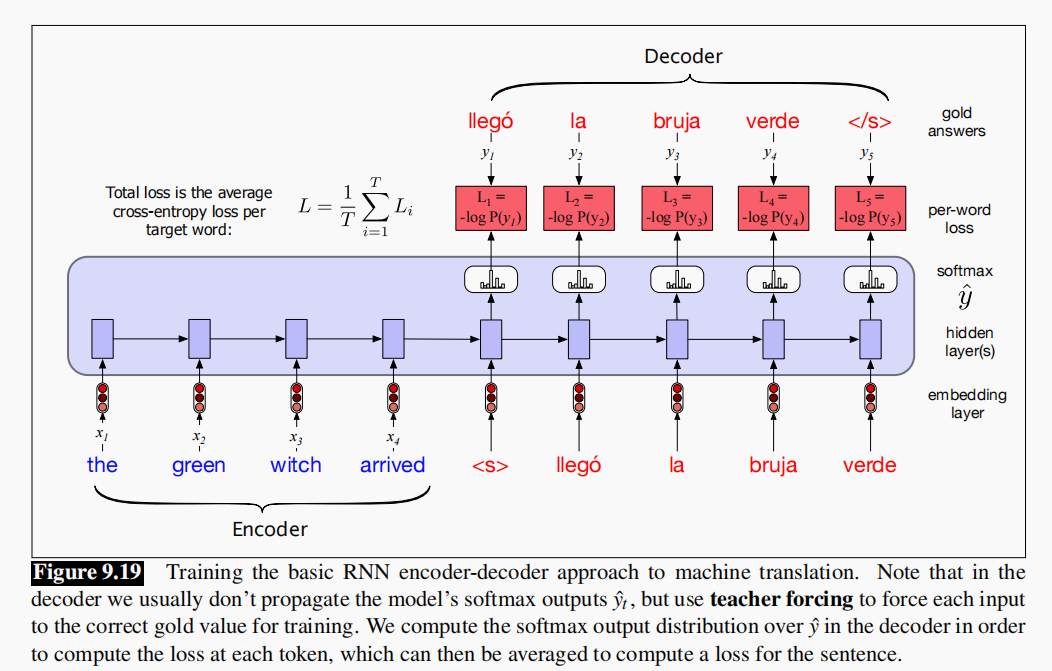
解码器初始化：解码器网络的最简单版本将使用这个状态初始化解码器的第一个隐藏状态；然后解码器自回归地生成输出序列，直到生成序列结束标记。

训练编码器-解码器模型

端到端训练：编码器-解码器架构的训练是端到端的。每个训练实例都是成对的字符串，一个源文本和一个目标文本。

机器翻译训练数据：通常由句子及其翻译组成的对应集合构成，这些可以来自标准的句子对数据集。

训练过程：网络接收源文本，然后从分隔符标记开始自回归地训练以预测下一个词。



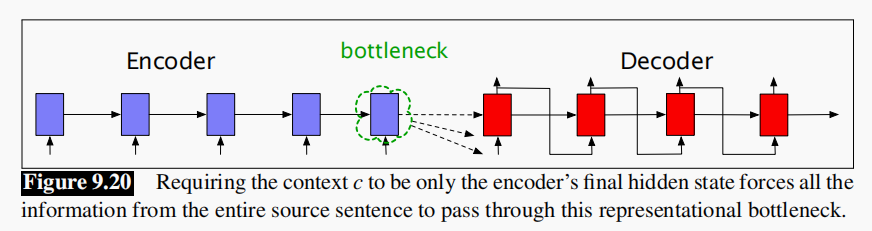
teacher forcing

训练和推理的差异：在训练期间，通常使用教师强制方法，即强制系统在解码器中使用训练时的正确目标标记作为下一个输入，而不是依赖于可能错误的解码器输出，这加速了训练过程。

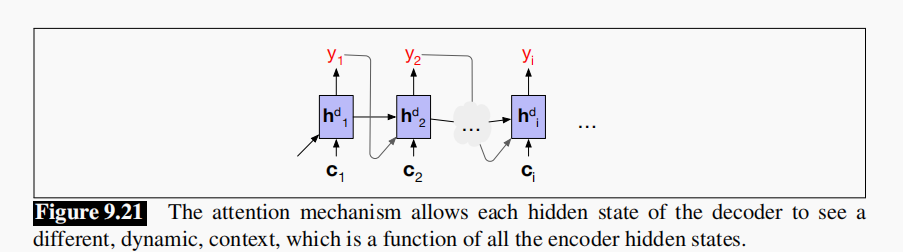
**9.8 Attention**

注意力机制基础

瓶颈问题：在简单的编码器-解码器模型中，源文本的整个含义必须通过最后一个隐藏状态（hn）来表示，这形成了一个信息瓶颈。



注意力机制的作用：解决了这个瓶颈问题，允许解码器访问编码器的所有隐藏状态，而不仅仅是最后一个。



注意力计算过程

重要性打分：在解码过程的每一步i，为每个编码器状态j计算一个分数，表示其对于解码器当前状态的相关性。

点积注意力：通过计算解码器隐藏状态与编码器隐藏状态之间的点积来实现，这个点积的分数反映了两个向量的相似度。

权重和上下文向量：利用这些分数，通过softmax归一化生成权重αij，然后使用这些权重对编码器的所有隐藏状态进行加权求和，得到固定长度的上下文向量ci。

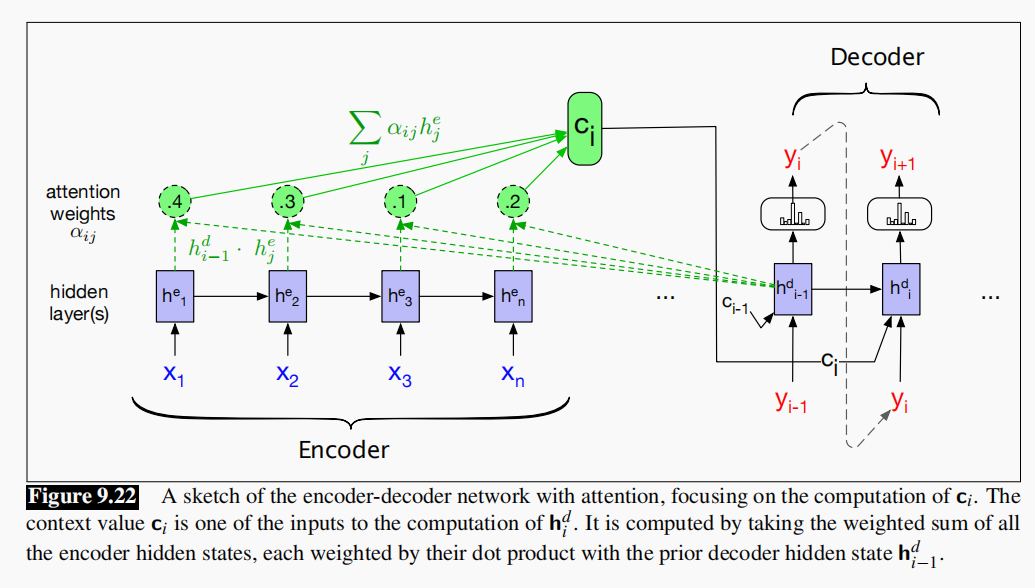
注意力在解码中的应用

动态上下文向量：在解码的每一步中，新生成的上下文向量ci考虑了所有编码器隐藏状态的信息，并根据解码的需要动态更新。

解码器的隐藏状态：通过使用先前的隐藏状态、先前的输出以及当前的上下文向量ci来计算当前的解码器隐藏状态。

训练及其优势

端到端训练：编码器-解码器架构通过正常的端到端训练来学习，在训练过程中，网络有能力学习解码器和编码器状态之间的相似性的重要方面。



参数化打分函数：除了简单的点积注意力，还可以通过引入自己的权重集Ws来创建更复杂的打分函数，使得网络可以使用不同维度的向量，并学习对当前应用重要的编码器和解码器状态之间的相似性。

注意力机制与变换器

注意力与变换器架构：变换器架构（Transformer）基于一种称为自注意力（Self-Attention）的注意力机制的小变种。在第10章中会具体介绍变换器架构。

提问：

1. 在有限的计算资源下，如何平衡注意力模型的精确度与计算复杂度？
2. 在具体的NLP任务（如情感分析）中，是否有策略来选择RNN的变种（如标准RNN、GRU、LSTM）
3. 在处理跨语言跨域应用时：

在跨语言NLP任务中，LSTM的哪些特性可以被用来捕获不同语言间的结构相似性？

如何利用LSTM模型在一个领域训练好的知识迁移到另一个领域（例如，从金融新闻到医疗报告的文本分类）？