循环神经网络

目录

1.词嵌入

1.词的表示

循环神经网络逐个接受单词，在循环神经网络中我们有几种方式来表示每个单词：

- one-hot向量：缺点：大词汇量将导致维数灾难；所有单词都是独立的，没有体现单词之间的关系。
- 词袋模型：使用单词频率表示句子，缺点：大词汇量会导致维数灾难；丢失了单词的位置信息。
- 词嵌入：用一组浮点数向量来表示一个单词，相似的词会被分在一起

2.词嵌入的学习 Word2Vec

现有的算法通过阅读大量文本文档来学习词嵌入表，这是一种自监督的学习方法。

- 连续词袋模型CBOW：通过上下文预测中间的单词
- Skip-gram(SG)：通过中间的单词预测上下文

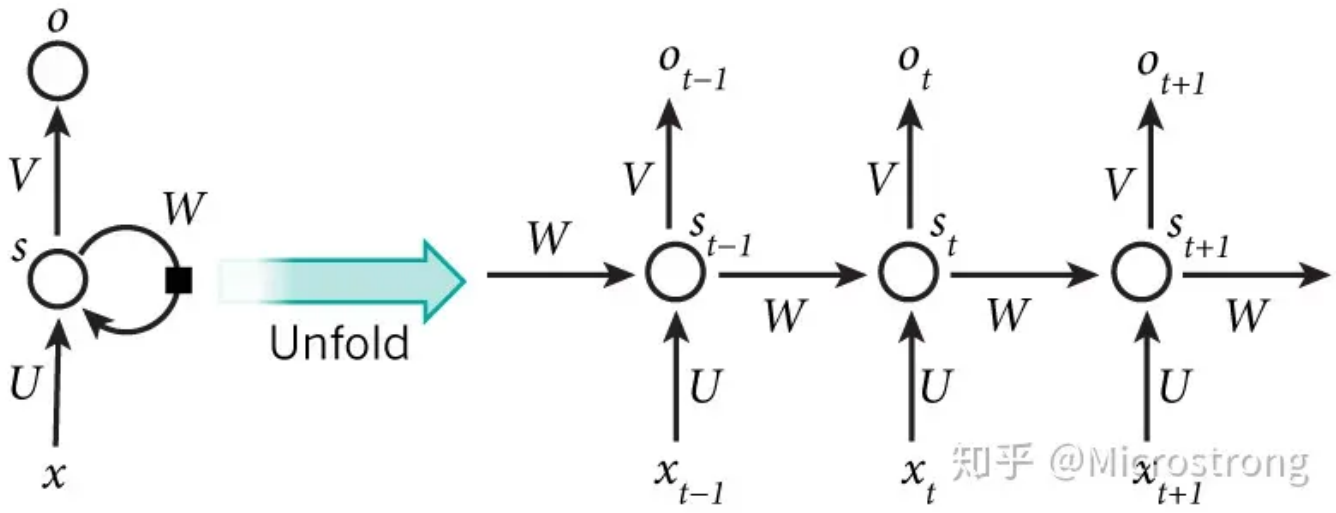
噪声对比估计 NCE

- SG在训练过程中，每个单词都被分为正样本和负样本，其中正样本是真正的单词，负样本是人工构造的，它们并未在实际文本中作为上下文词对出现。负样本的选择通常是基于词频的，但会进行一些调整。这些负样本被用来训练模型以识别不太可能出现的词对。
- Skip-gram有多个目标输出，因此使用sigmoid而不是softmax，我们独立的对每个单词进行分类。
- 大词汇表会导致巨大的计算成本，我们可以使用负采样来加速损失函数的计算，从词汇表中随机采样N个负样本，这种方法被称为噪声对比估计（NCE）。

2.朴素循环神经网络与门控单元

1.朴素循环神经网络

循环神经网络与前馈神经网络之间唯一的区别在于，它会将前一个时间步的信息传递给下一个时间步：

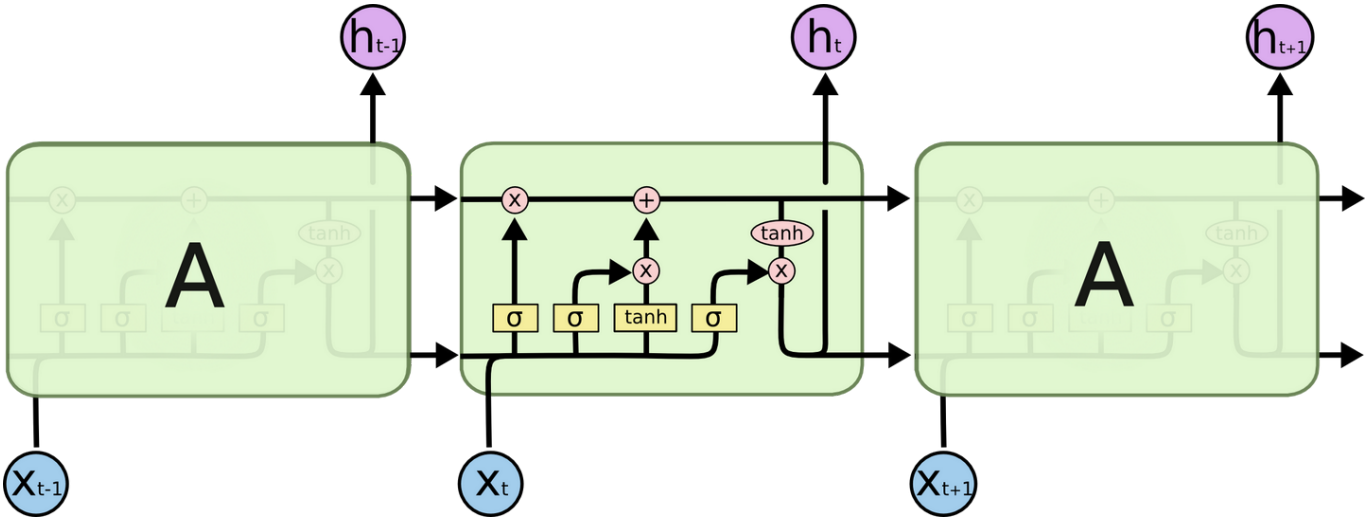


假设第t个时间步的输出为o_t，激活函数为$g(x)$，每一个时间步的输入为x_t，每个时间步的矩阵为W_t，输入到隐藏层和隐藏到输出层的矩阵分别为$U、V$，隐藏层的输出为s_t，则有： $$ s_t = f(Ux_t + Ws_{t-1}) \ o_t = g(Vs_t) = g(Vf(Ux_t + Ws_{t-1}))$$ 这样就实现了前一个时间步保留信息的传递。

朴素神经网络的一个局限性是：长期依赖问题。朴素RNN很难维持较为长期的信息，可能存在梯度消失的问题。

2.长短期记忆网络 LSTM

LSTM通过门控单元来控制信息的流动，从而解决了长期依赖问题。LSTM的结构如下：



朴素RNN在时间维度上只传递一个隐藏状态，而LSTM在时间维度上传递两个隐藏状态，一个是长期状态c_t，一个是短期状态h_t：

- **隐藏状态向量**：这是LSTM的输出，通常记为h。对于每一个时间步，隐藏状态向量包含了过去信息的某种程度的总结，同时也用于预测当前时间步的输出。这个向量也会被传递到下一个时间步，与新的输入一起决定下一步的输出和新的隐藏状态。
- **细胞状态向量**：这是LSTM的"记忆"部分，通常记为c。细胞状态是一条在时间维度上"流动"的信息通道，它能够跨越很长的时间序列进行信息传递。通过门控机制（包括遗忘门、输入门和输出门），LSTM能够

学会如何添加新的信息到细胞状态中，或者从细胞状态中删除不再需要的信息。这种设计使得LSTM能够有效地捕捉和利用长期依赖关系。

LSTM是通过三个门控单元完成细胞状态的存储的：

- **遗忘门**：决定了细胞状态中哪些信息需要被删除。它通过一个sigmoid层计算h和x来输出，并且传递给c。
- **输入门**：决定了细胞状态中哪些新的信息需要被添加。它通过一个sigmoid层和一个tanh层计算h和x来输出，并且叉乘两个输出结果，最后将结果添加到细胞状态中。
- **输出门**：决定了细胞状态中哪些信息需要被输出。它通过一个sigmoid层返回新的隐藏状态h，再使用tanh层将结果添加到细胞状态中。

3.序列到序列模型 Seq2Seq

序列到序列模型是一种常见的循环神经网络结构，它由两个循环神经网络组成，一个编码器（Encoder）和一个解码器（Decoder）。编码器将输入序列转换为一个向量，解码器将这个向量转换为一个新的序列。

- **编码器**：编码器是一个循环神经网络（RNN）或其变体（如LSTM或GRU），用于处理输入序列。输入序列可以是一句话、一段文字或者时间序列等。编码器会将这个输入序列转化为一个固定大小的上下文向量（也称为编码向量）。上下文向量试图捕捉输入序列的主要信息。
- **解码器**：解码器是另一个RNN，它接收编码器产生的上下文向量，并以此为基础生成输出序列。输出序列可以是翻译后的句子、图像描述、下一个时间步的预测等。

束搜索 Beam Search

在Seq2Seq模型中，输出序列的生成可以看作是一个搜索问题，我们需要在所有可能的输出序列中寻找最好的序列。贪心搜索和束搜索都是为了解决这个问题的方法，但是贪心所做的第一选择不一定是最优的，所以我们使用束搜索。 **束搜索的Beam越大，结果越好，越慢，越耗算力；Beam越小，结果越差，越快，越省算力。**

3.注意力机制

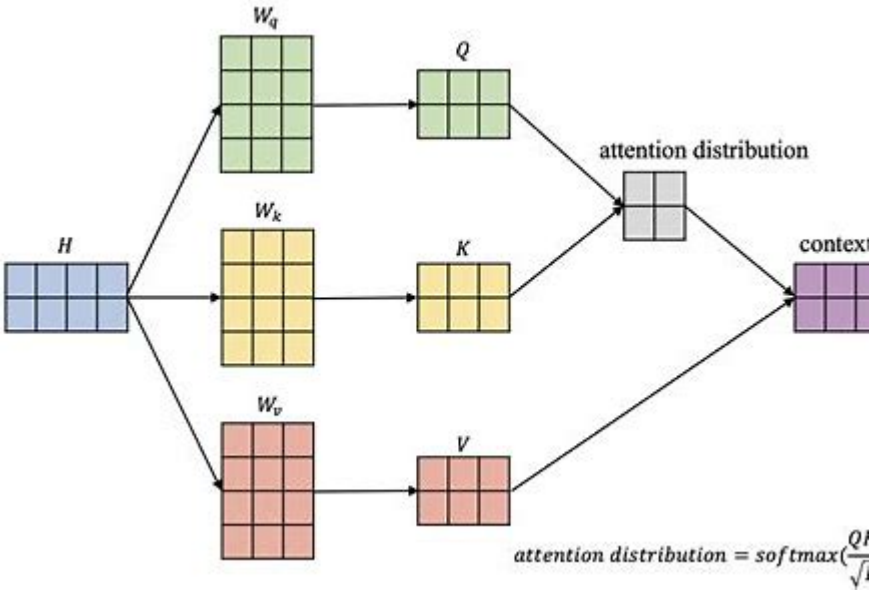
早期的序列模型有一个共同的问题：长度依赖问题。人在阅读长篇内容时，通常会将注意力集中在一些词语上以便理解。

1.自注意力机制

注意力提示非为两类：

- 非自主性提示：键（Key）、值（Value）
- 自主性提示：查询（Query）

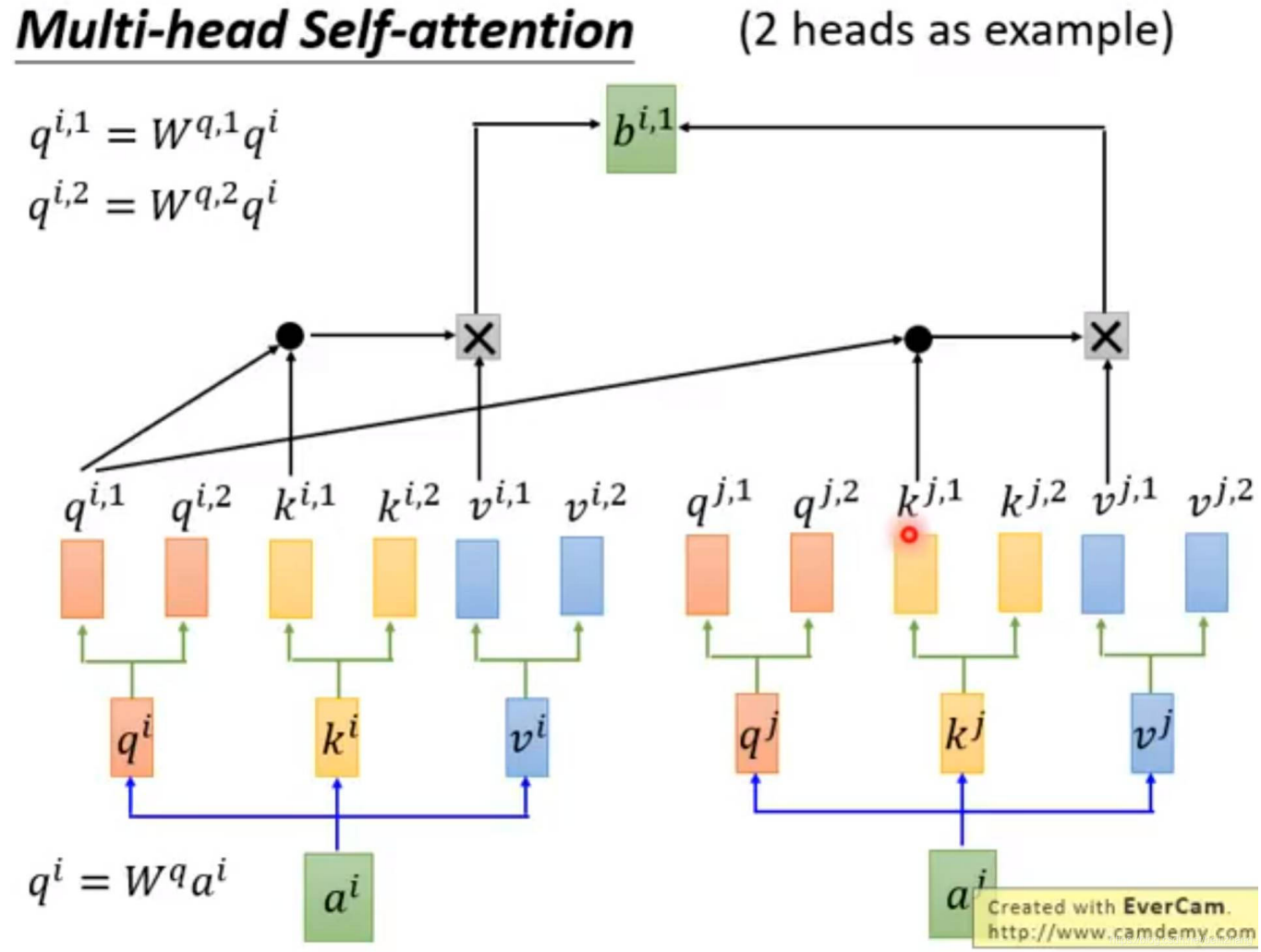
注意力汇聚则是通过计算，得到注意力汇聚在哪些单词上的过程：



我们可以使用K、V、Q三个矩阵来获得注意力，使用d_k表示键向量的维度： $$ Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V $$ 其中，缩放（除以维度的平方根）的原因是防止点积过大；softmax归一化的原因是使所有注意力权重的和为1。

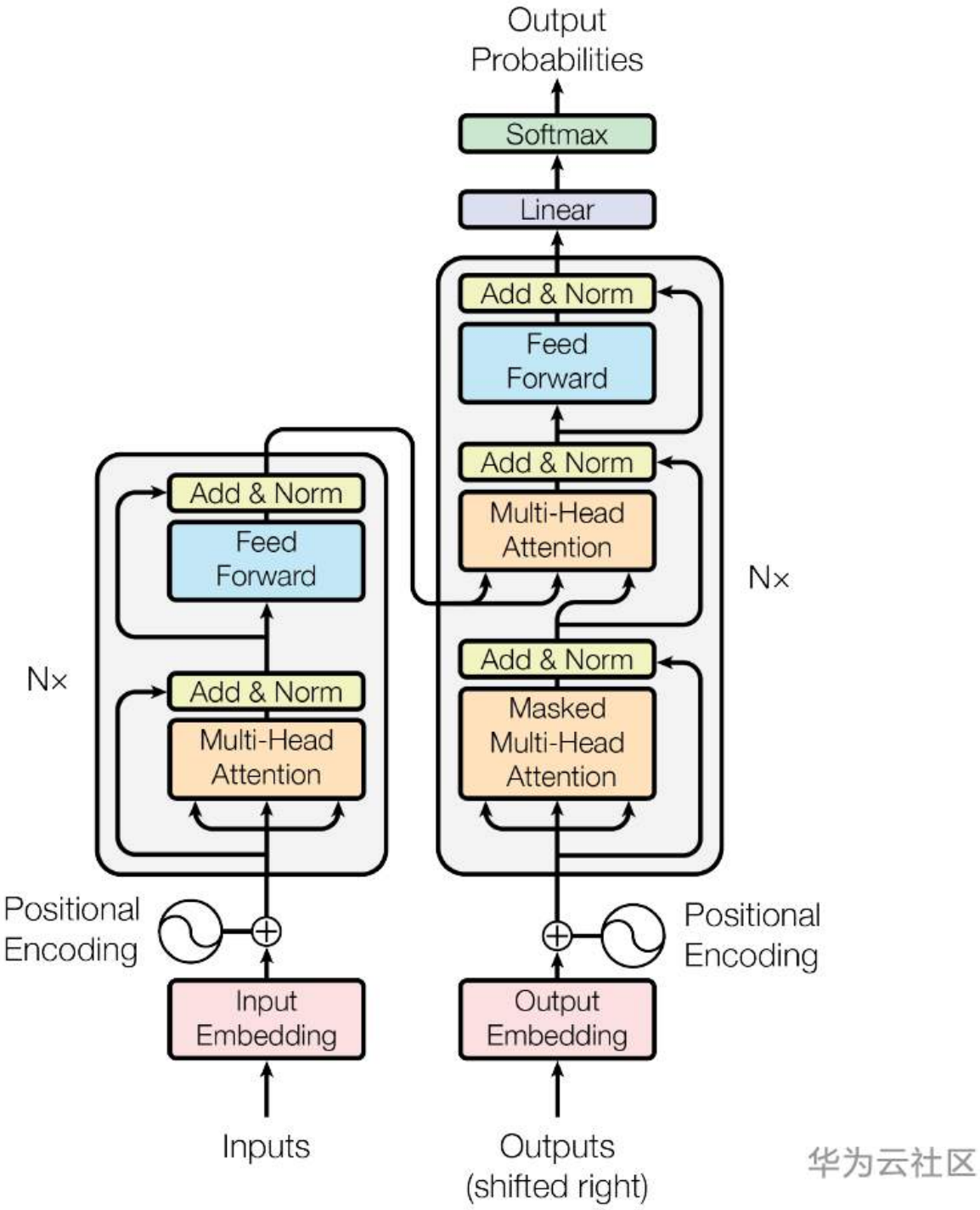
2.多头注意力机制

多头注意力包括了h个self-attention组合，映射到h个空间，关注不同的特征维度：



4.Transformer

Transformer模型与传统的Seq2Seq模型（如基于RNN的模型）的主要区别在于，Transformer完全基于自注意力机制（self-attention）来捕捉序列中的依赖关系，而不依赖于循环或卷积结构，这使得它在处理长序列时具有优越的性能：



transformer的计算过程包括如下几个步骤：

- **编码器**：编码器由多层相同的层叠加而成，每一层主要包括两部分：自注意力机制和全连接的前馈网络。自注意力机制用于捕捉输入序列中的全局依赖关系，全连接的前馈网络用于进行非线性变换。此外，每个子层（包括自注意力子层和全连接子层）都有一个残差连接和层归一化。
- **解码器**：解码器的结构与编码器类似，也是由多层相同的层叠加而成，每一层主要包括三个部分：掩蔽注意力机制、编码器-解码器注意力机制和全连接的前馈网络。解码器的自注意力机制允许解码器在生成

每个词时考虑到前面已经生成的词，编码器-解码器注意力机制则使得解码器可以关注到编码器的输出。每个子层也有一个残差连接和层归一化。

掩码

在使用自注意力机制时，有一个问题需要注意，那就是在生成序列的任务中，如机器翻译或者文本生成，**当前的词只应该依赖于前面的词，而不应该看到后面的词。**这就需要对自注意力机制进行一定的修改，即使用**掩蔽多头注意力**

假设我们正在生成一个英文句子，当前已经生成到"The cat is"，下一个词应该是"on"，在这个时候，我们希望模型只看到"The cat is"，而不能看到"on"，因为在实际使用时，"on"还没有被生成出来。掩蔽多头注意力就可以实现这一点。

在掩码矩阵中，如果该元素对应的位置是要掩盖的位置，则设置为负无穷，在softmax前，将掩码矩阵和注意力分数相加，对应的注意力权重就会变成0，这样就实现了掩蔽的效果。

编码器-解码器注意力机制

编码器-解码器注意力层中，Q来自前一个解码器层的输出，而K和V来自于编码器的输出。