8 综述 —— RNNs 和 LSTMs

本章概述

从改进N元语言模型的研究发展历程的角度来看，**神经网络语言模型**或者说神经概率语言模型的出现具有一定的**趋势性**。

N元语言模型基于统计连续的N个词出现的频率来估计概率，进而实现对下一个词的预测。由于**词的长尾分布**，N元语言模型不可避免的存在**数据稀疏**问题，统计语料中那些罕见或未出现的 N 元语法会导致概率为接近或等于零，从而使得**最大似然估计**的结果不可靠。**收集更多数据**是一种简单的解决方案，但成本高昂，并且受 **Zipf 律**的约束，无论数据集大小如何，罕见的单词或组合都很难捕获。因此人们考虑对N与语言模型的N语言语法频率统计矩阵进行**平滑**操作，从最初简单的全部一刀切的频率+1的**Laplace平滑**，再到频率相同的相同处理的**Good Turning平滑**，再到频次相同下、具有不同活性的语法分组处理的**KN平滑**等。实验表明平滑技术在低于5阶时尚有一定效果，但随着阶数的提升也无法阻止数据稀疏造成的模型性能下降。

从Laplace平滑到KN平滑，平滑技术做得越来越复杂，可以发现其思想就在于利用更细粒度的**词间相似**得到越来越小的**分组**，处于同一分组的进行相同的平滑处理。 一般情况下，相似的单词应该具有相似的条件概率。这就需要单词的**连续表示**。如果的话 x 和 y 相似，条件概率 $P(y|\text{context})$ 应该接近 $P(x|\text{context})$ 。实现该想法的第一步是将词映射到**连续的向量空间** $C(.)$ 。这种向量表示对语义相似性进行编码，并允许模型通过将相似的上下文与相似的单词相关联来更好地概括。基于第一步，构建**连续函数** $P(.)$ ，此时若 $C(y)$ 接近 $C(x)$ ，则 $P(C(y))$ 接近 $P(C(x))$ 。这样的方案可以认为是实现了更细粒度的平滑。而根据**通用近似定理**，我们知道神经网络在理论上具有很强的拟合任意函数的能力，因此出现了神经网络语言模型，神经网络用基于连续嵌入的近似概率的函数取代了 N-gram 模型中基于离散计数的概率表。

基于**前馈神经网络**的语言模型在性能上超越了传统的N元语法模型，但其无法建模序列的**次序**，且无法应对**变长**的语言串，因此研究提出了**循环神经网络**。循环神经网络顺序接收输入，和语言的顺序特点吻合。而最朴素的RNN模型因其**梯度消失**和梯度爆炸问题需要进行改进，从而发展出**LSTM**、GRU等更成熟的循环神经网络。

RNN

RNN（Recurrent Neural Network，循环神经网络）是一种用于处理序列数据的神经网络，广泛应用于自然语言处理、时间序列预测等领域。在语言建模任务中，RNN的目标是根据给定的前序词预测下一个词，因此其核心目的是捕捉序列数据中的依赖关系。传统的前馈神经网络对于输入序列中的每个时间步独立处理，无法有效地利用序列中的时间依赖性。而RNN通过在隐藏层中保留状态，将序列中的信息逐步传递，使得模型能够记忆前面步骤中的信息，从而具备对时间序列的建模能力。

模型结构

RNN 的主要结构包括输入层、隐藏层和输出层。隐藏层的状态通过时间传递，形成了序列数据中的依赖关系。

1. 输入和输出的表示

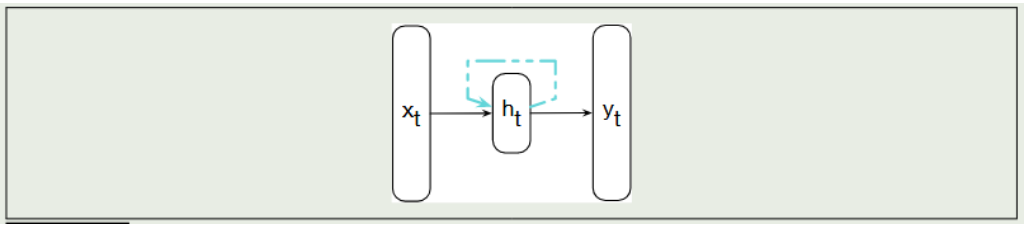
假设给定输入序列为 $\mathbf{X} = (x_1, x_2, \ldots, x_T)$，其中 x_t 表示在时间步 t 的输入。每个时间步上的输入会与当前的隐藏状态一起影响输出。

2. 隐藏层的递归结构

在RNN中，隐藏状态 $\mathbf{h}_t$ 是用于表示时间步 t 的上下文信息。隐藏状态的更新通过递归公式来实现：

$$\mathbf{h}_t = f(\mathbf{W}_h x_t + \mathbf{U}_h \mathbf{h}_{t-1} + \mathbf{b}_h)$$

其中 $\mathbf{W}_h$ 是输入到隐藏层的权重矩阵，$\mathbf{U}_h$ 是上一时间步隐藏状态到当前隐藏状态的权重矩阵，$\mathbf{b}_h$ 是偏置向量，$f(\cdot)$ 是激活函数，通常使用 $\tanh$。



3. 输出层

RNN的输出通常是一个概率分布，用于预测下一个词。给定隐藏状态 $\mathbf{h}_t$，输出 $\mathbf{y}_t$ 通常通过下式得到：

$$\mathbf{y}_t = g(\mathbf{W}_o \mathbf{h}_t + \mathbf{b}_o)$$

其中 $\mathbf{W}_o$ 是隐藏层到输出层的权重矩阵，$\mathbf{b}_o$ 是偏置向量，$g(\cdot)$ 是一个softmax函数，用于将输出转换为概率分布。因此，输出 $\mathbf{y}_t$ 表示在时间步 t 时，每个可能词汇出现的概率。

模型推理

模型推理是指给定序列中的一部分，预测下一个词的过程。

在推理过程中，RNN会根据输入的每一个时间步更新隐藏状态 $\mathbf{h}_t$，并输出当前时间步的概率分布 $\mathbf{y}_t$。这种逐步递归的推理使得RNN能够生成一个序列中的每个元素，直到满足特定的终止条件。如果给定序列为 $(x_1, x_2, \ldots, x_{t-1})$，推理的步骤如下：

a. 初始化隐藏状态 $\mathbf{h}_0$（通常为零向量）。

b. 对于每一个时间步 t，输入 x_t：

- 使用 $\mathbf{h}_{t-1}$ 和 x_t 计算当前的隐藏状态 $\mathbf{h}_t$。
- 通过输出层计算概率分布 $\mathbf{y}_t$。

c. 从 $\mathbf{y}_t$ 中采样，获得下一个可能的词 x_{t+1}。

模型训练

RNN语言模型的训练目标是最小化预测的下一个词和真实词之间的差异。具体来说，通常通过最大化条件概率 $P(x_t|x_1, x_2, \ldots, x_{t-1})$ 来训练模型。

在训练过程中，RNN 使用交叉熵损失函数来衡量模型输出与真实标签之间的误差。假设训练数据包含一个长度为 T 的序列 $\mathbf{X} = (x_1, x_2, \ldots, x_T)$，交叉熵损失函数定义为：

$$L = -\sum_{t=1}^{T} \log P(x_t|x_1, x_2, \ldots, x_{t-1})$$

其中，$P(x_t|x_1, x_2, \ldots, x_{t-1})$ 是通过softmax层计算得到的。

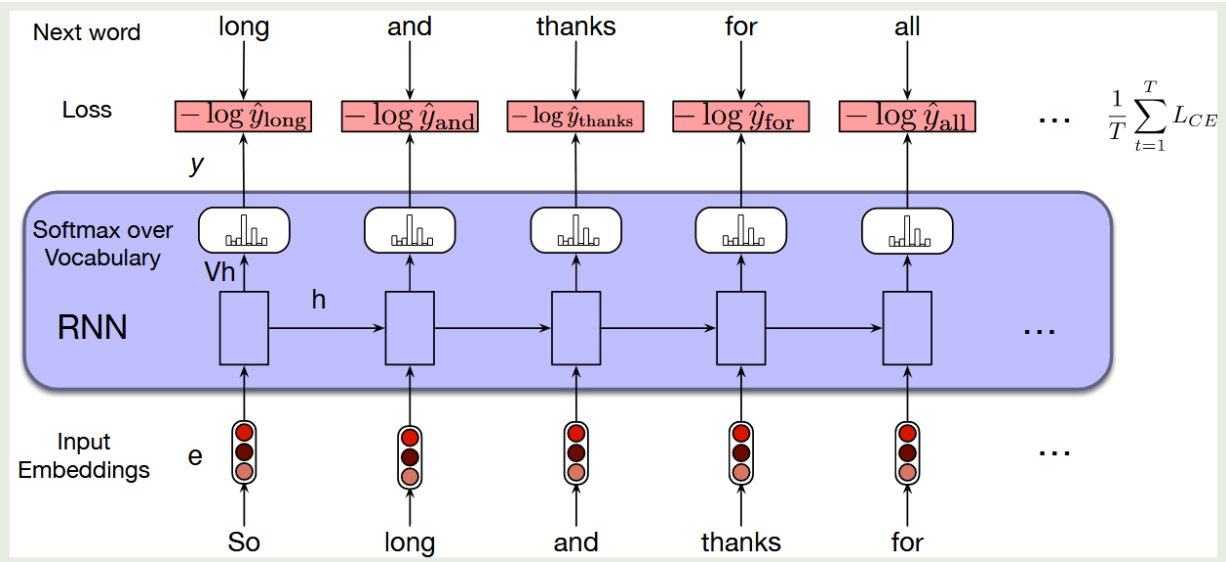
在前馈神经网络的训练中我们使用到了反向传播算法。为了将其扩展到具有时序关系的RNN模型上，我们使用其扩展版本“基于时间步的反向传播”算法（Backpropagation Through Time，BPTT）。在 BPTT 中，损失函数的梯度通过时间方向进行反向传播。

- 前向传播：对于每一个时间步 t，计算隐藏状态 $\mathbf{h}_t$ 和输出 $\mathbf{y}_t$。
- 反向传播：通过时间反向传播误差，从时间步 T 传播到 $t = 1$，更新所有的模型参数（如权重矩阵 $\mathbf{W}_h$、$\mathbf{U}_h$ 和 $\mathbf{W}_o$）。
- 梯度更新仍然遵循链式法则。对于隐藏层的梯度，BPTT 通过反向传播跨越多个时间步，将误差积累起来进行梯度计算。隐藏状态的梯度表示为：

$$\frac{\partial L}{\partial \mathbf{h}_t} = \frac{\partial L}{\partial \mathbf{h}_{t+1}} \cdot \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t} + \frac{\partial L_t}{\partial \mathbf{h}_t}$$

- 在计算出所有参数的梯度后，可以通过梯度下降算法来更新参数。例如，对于权重矩阵 $\mathbf{W}_h$，其更新公式为：

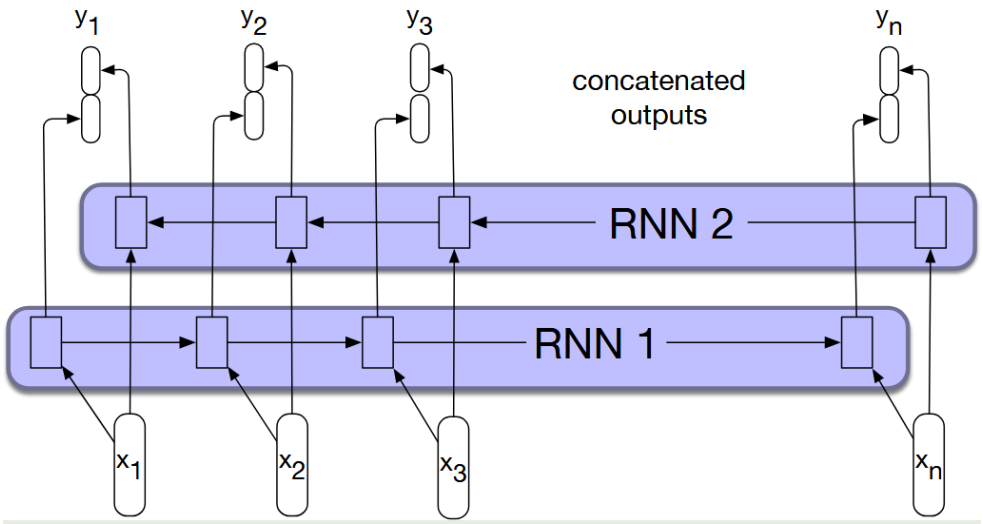
$$\mathbf{W}_h \leftarrow \mathbf{W}_h - \eta \frac{\partial L}{\partial \mathbf{W}_h}$$



根据训练过程可以看出，RNN 的梯度是通过时间反向传播的，容易出现梯度消失或梯度爆炸的问题。梯度消失会导致模型在较长的序列中无法捕捉长距离依赖，而梯度爆炸会导致权重更新过大，进而导致模型不稳定。为了缓解这些问题，可以使用梯度裁剪和更复杂的模型结构（例如 LSTM 和 GRU）来解决。

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，我们希望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。



当前时间步左侧的上下文： $\mathbf{h}_t^f = \mathrm{RNN}_{\mathrm{forward}}(\mathbf{x}_1, \ldots, \mathbf{x}_t)$

当前输入右侧序列的信息： $\mathbf{h}_t^b = \mathrm{RNN}_{\mathrm{backward}}(\mathbf{x}_t, \ldots \mathbf{x}_n)$

将前后两个网络计算的表示拼接成一个向量： $\mathbf{h}_t = [\mathbf{h}_t^f; \mathbf{h}_t^b] = \mathbf{h}_t^f \oplus \mathbf{h}_t^b$
然后再基于 $\mathbf{h}_t$ 完成后续任务。

LSTM

LSTM（Long Short-Term Memory）是一种特殊的 RNN，专为解决传统 RNN 的长期依赖性问题而设计。RNN 在捕捉长距离依赖时，常遇到梯度消失或梯度爆炸问题。LSTM 通过设计一套门控机制来控制信息的流动，能够在长时间步中保持更稳定的梯度，从而有效捕捉长距离的上下文信息。

与传统 RNN 不同的是，LSTM 通过引入 遗忘门（forget gate）、输入门（input gate） 和 输出门（output gate） 来控制信息在时间序列中的传播和状态更新，避免梯度消失或爆炸，尤其在长序列任务中表现更加出色。

模型结构

LSTM 的结构相对复杂，由多个门控单元组成，每个时间步都会通过门控单元来选择性地保留或忘记某些信息。LSTM 通过 **细胞状态（cell state）** 和 **隐藏状态（hidden state）** 的相互作用来实现对信息的控制。

在每个时间步 t ，LSTM 单元包括以下几个组件：

- 输入门：控制输入信息的更新比例。
- 遗忘门：控制上一个细胞状态中信息的遗忘比例。

- 输出门：控制当前隐藏状态的输出信息。
- 细胞状态：负责传递长期记忆。
- 隐藏状态：负责传递短期记忆。

给定输入序列 $\mathbf{X} = (x_1, x_2, \ldots, x_T)$，在每个时间步上有以下更新规则：

a. 遗忘门用于控制上一时间步的细胞状态是否被保留或丢弃：

$$\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}, x_t] + \mathbf{b}_f)$$

其中 $\mathbf{W}_f$ 和 $\mathbf{b}_f$ 是遗忘门的权重矩阵和偏置向量，$\sigma(\cdot)$ 是 sigmoid 激活函数，用于将结果压缩到区间 [0, 1]，表示信息被保留的程度。

b. 输入门用于控制新信息的写入比例:

$$\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}, x_t] + \mathbf{b}_i)$$

候选细胞状态 $(\tilde{\mathbf{C}}_t)$ 用于更新细胞状态：

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{W}_C[\mathbf{h}_{t-1}, x_t] + \mathbf{b}_C)$$

然后，输入门决定了多少新信息被加入到细胞状态中。

c. 更新细胞状态，通过结合遗忘门和输入门来实现：

$$\mathbf{C}_t = \mathbf{f}_t \odot \mathbf{C}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{C}}_t$$

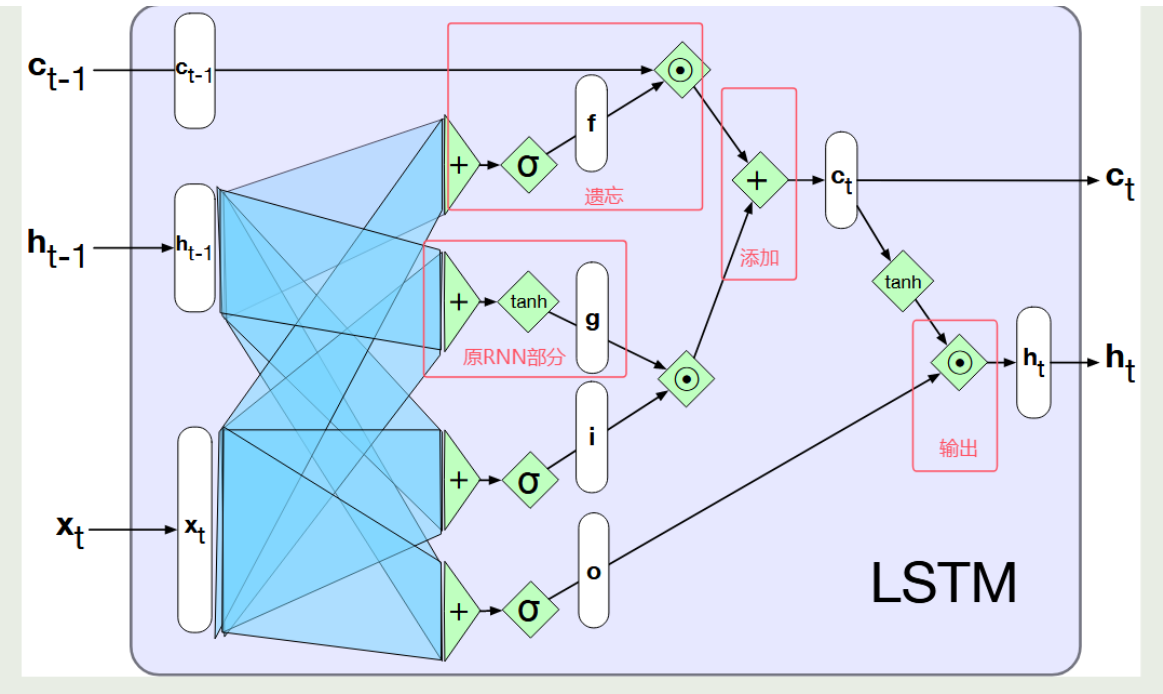
其中 $\odot$ 表示按元素相乘。

d. 输出门控制当前隐藏状态的输出：

$$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}, x_t] + \mathbf{b}_o)$$

隐藏状态则是基于当前的细胞状态和输出门的结果：

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{C}_t)$$



本章问题

编码器-解码器架构的输入序列长度一直是我们关心的一个重要指标。理论上RNN和LSTM甚至Transformer再理论上都能处理任意长或无限长的序列长度，但在实际的模型训练、推理中会因为梯度、计算复杂度等各种原因而无法实现理论长度。能否考虑给这种编码器-解码器架构构造一种显示的记忆机制（即保留过程中的状态计算），从而实现文本的分段处理，但仍然能保持连贯性？