

Seq2seq模型

概要

- 根本问题: 序列到序列的转换挑战
- 核心思想: 编码器-解码器 (Encoder-Decoder) 范式
- 具体实现: 序列到序列 (Seq2seq) 模型
- 解码策略: 从贪婪搜索到束搜索 (Beam Search)
- 质量评估: BLEU 分数

根本问题：序列到序列的转换

➤根本问题

- 传统的神经网络（如CNN, MLP）要求输入和输出的维度固定
- 无法处理自然语言中普遍存在的不等长序列转换任务（如机器翻译、文本摘要）

➤核心目标

- 设计通用的、端到端的神经网络架构，能够学习任意两个长度可变的序列之间的映射关系

编码器-解码器类比

➤ Seq2seq模型就像一位“使用速记的单线程思维译员”

- 理解与速记 (Encoding): 译员逐字听完一段话。边听边做笔记，不断更新和覆盖已有理解，最终形成一个高度浓缩的最终摘要 - 上下文向量 c
- 生成与输出 (Decoding): 随后，译员仅凭最终摘要开始翻译。逐词生成译文，每个词，都会参考刚刚说的内容和最终摘要，来决定下一个词

➤ 核心缺陷暴露

- 信息瓶颈 (Information Bottleneck): 如果原文很长，一个固定维度的向量 c 必然会丢失大量信息，尤其是开头的细节
- 与原文的隔离: 译员在翻译后期需要精确细节时，无法从模糊的摘要中获取

编码器-解码器 (Encoder-Decoder) 范式

➤ 编码器-解码器是一种通用的模型架构。其中

➤ 编码器 (Encoder): 理解。将变长输入序列 $X = (x_1, x_2, \dots, x_T)$ 映射到固定维度上下文向量 c

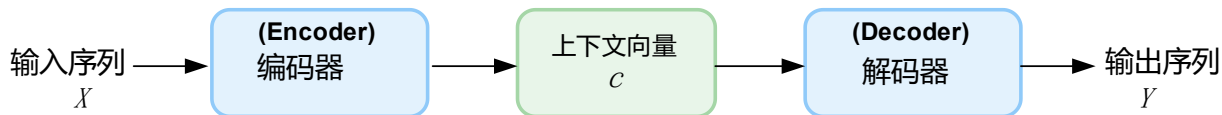
➤ 解码器 (Decoder): 生成。以 c 为条件，逐个生成变长目标序列 $Y = (y_1, y_2, \dots, y_{T'})$

➤ 上下文向量 c : 连接两个世界的唯一桥梁

➤ 核心属性:

➤ 解耦: 将输入表示与输出生成分离，两个模块可以采用不同的结构

➤ 信息桥梁: 上下文向量 c 连接编码器和解码器的唯一信息通道。必须封装源序列全部必要语义



具体实现：序列到序列 (Seq2seq) 模型

- 如何为序列数据（如文本）找到一个强大的编码器和解码器实现？
- 利用循环神经网络 (RNN) 及其变体 (LSTM, GRU) 的强大序列建模能力，来实例化 Encoder-Decoder 范式
 - 编码器: 一个 RNN。它按时间步逐个读取输入序列的元素，并将其信息逐步压缩进其隐状态 (hidden state)
 - 解码器: 另一个 RNN。它接收编码器的最终摘要信息，并逐个生成输出序列的元素
- Seq2seq 模型
 - 专门为解决序列到序列转换任务而设计的具体模型
 - Encoder-Decoder 范式在 NLP 等领域的经典实现

核心思想与工作机制

➤ 编码阶段：递归的信息压缩

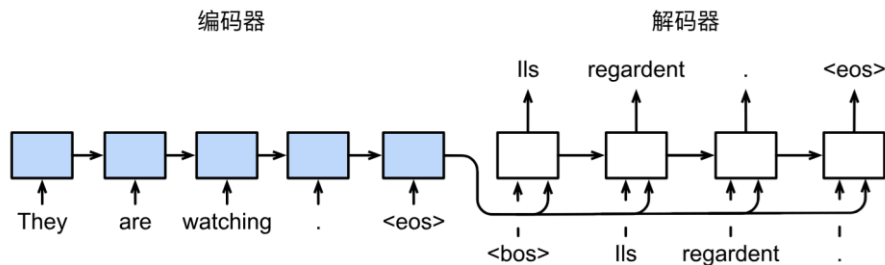
- 每个时间步 t 读取输入词元 x_t ，更新隐状态 $h_t = f(h_{t-1}, x_t), c = h_T$
- 最后一个时间步的隐状态 h_T ，视作上下文向量 $c = h_T$ ，为输入序列的语义摘要（充分统计量）

➤ 解码阶段：条件语言建模

- 核心任务：学习一个条件概率分布 $P(Y|X) = P(y_1, \dots, y_{T'}|c)$ 。
- 链式法则展开： $P(Y|c) = \prod_{t=1}^{T'} P(y_t|y_{<t}, c)$
- 初始隐状态： $s_0 = c$
- 从起始符 <bos> (begin-of-sequence) 开始，在每个时间步 t 生成一个输出 y_t
 - 隐状态更新： $s_t = g(s_{t-1}, y_{t-1})$
 - 输出 y_t ： $P(y_t|y_{<t}, c) = \text{Softmax}(W_o s_t)$
 - 生成过程持续进行，直到结束符 <eos> (end-of-sequence)

Seq2seq 的工作机制：一个实例

- 通过一个具体的翻译任务（英语 → 法语）来可视化Seq2seq的工作流程
- 编码阶段（Encoder），读取输入序列 " They are watching. <eos> "
 - 每个词元更新RNN的内部隐状态。信息在时间步之间向右传递
 - 读到<eos>（序列结束符）时，编码器最终的隐状态被捕获，形成上下文向量 c
 - c 是整个输入句子的浓缩表示
- 解码阶段（Decoder），直到解码器生成<eos>标记，表示输出序列已完成
 - 上下文向量 c 被用作解码器RNN的初始隐状态
 - 从一个特殊的起始符<bos>开始生成
 - 自回归：在每个时间步，上一步的输出作为下一步的输入，引导模型生成序列的下一个部分



训练与推理的根本差异

- Seq2seq模型在训练和推理（生成）时，解码器的工作方式完全不同
- 训练：Teacher Forcing (强制教学)
 - 解码每一步，无论模型上一步预测什么，都将真实的目标词元作为下一步输入
 - 优点
 - 允许并行计算，训练速度快
 - 避免了误差累积（如果模型早期预测错误，不会影响后续步骤的学习）
 - 缺点
 - 模式暴露 (Exposure Bias)。造成训练与推理之间的差异，模型在训练时从未见过自己生成的错误
- 推理：Autoregressive (自回归)
 - 在解码的第 t' 步，将模型自己在第 $t' - 1$ 步预测出的词元作为输入
 - 特点
 - 生成过程是串行的，速度较慢
 - 一旦出现预测错误，这个错误可能会被带入后续的计算，导致误差累积，生成质量下降

核心矛盾：信息瓶颈C

关键假设与前提：信息瓶颈

- 一个固定大小的上下文向量 c 能够充分地捕捉任意长度输入序列的全部语义信息
- 这个假设在什么时候会不成立？
 - 当输入序列非常长时，例如在长文档翻译或摘要任务中
- 后果
 - 信息丢失：RNN在处理长序列时，倾向于忘记早期的信息（类似“遗忘症”）。强迫所有信息挤入一个固定大小的向量 c 会加剧这个问题，导致它成为一个信息瓶颈
 - 性能退化：模型翻译长句子的能力会显著下降
 - 这一根本性弱点，催生了后续的注意力机制 (Attention Mechanism)，后者允许解码器在生成每个词时，“回顾”并动态地关注输入序列的不同部分

解码策略与评估

解码的困境：局部最优 vs. 全局最优

- 推理时，解码器在每个时间步都会输出一个词汇表大小的概率分布 $P(y_{t'}|y_{<t'}, \mathbf{c})$ 。我们的目标是找到一个整体概率最大的输出序列 Y

$$P(Y|X) = \prod_{t'=1}^{T'} P(y_{t'}|y_1, \dots, y_{t'-1}, \mathbf{c})$$

策略1：贪婪搜索 (Greedy Search)

- 在每个时间步，简单地选择当前概率最高的那个词。这样非常快，实现简单
- 但是局部最优不等于全局最优。一个当前看起来最好的选择，可能会引入一条后续概率很低的路径
 - 路径A: $0.5 \times 0.4 \times 0.4 \times 0.6 = 0.048$
 - 路径B: $0.5 \times 0.3 \times 0.6 \times 0.6 = 0.054$ (更优)
 - 贪婪搜索会选择路径A的第一步 (A: 0.4 > B: 0.3)，从而错失了全局最优的路径B

策略2：穷举搜索 (Exhaustive Search)

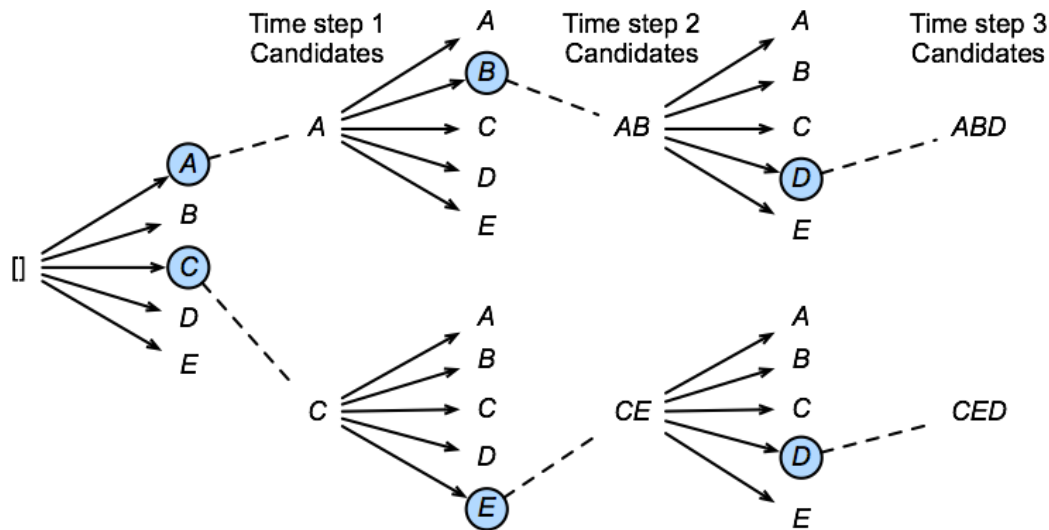
- 计算所有可能的输出序列的概率，然后选择最好的一个
- 如果词汇表大小为 $|V|$ ，最大长度为 T' ，计算复杂度为 $O(|V|^{T'})$ ，这在计算上是绝对不可行的
- 我们需要一个介于两者之间的折中方案

务实的妥协：束搜索 (Beam Search)

- 束搜索是在计算成本和生成质量之间取得平衡的解码算法
- 核心思想: 在解码的每一步，不再只保留一个最佳选择，而是保留 k 个最有可能的候选序列（这 k 个序列被称为“束”或“beam”）
 - 参数 k 称为束宽 (Beam Width)
 - 当 $k=1$ 时，束搜索退化为贪婪搜索
 - 当 k 接近词汇表大小时，束搜索接近穷举搜索
 - k 是一个超参数，需要在生成质量和计算成本之间进行权衡。常用值为 5, 10
- 束搜索通过在每一步探索有限数量的可能的路径，来近似寻找全局最优的输出序列

束搜索：工作机制图示 (k=2)

- 每次都保留最好的 k 个候选
- 向候选项添加一项 (n 种可能)，在 kn 序列选出最好的 k 个



束搜索的优化：对数概率与长度惩罚

➤ 问题1: 数值下溢

- 概率是多个小于1的数连乘，序列越长，总概率会越接近0，可能导致计算机浮点数下溢
- 解决方案: 使用对数概率 (log-probability)。连乘变为求和，计算更稳定，且不改变最优解

$$\text{score}(Y) = \sum_{t=1}^{T'} \log P(y_t | y_{<t}, c)$$

➤ 问题2: 对短序列的偏好

- 因为对数概率是负数，序列越长，累加的负数越多，分数越低。导致束搜索天然地偏爱更短的序列
- 解决方案: 长度惩罚 (Length Penalty)。我们用序列长度的某个函数来对总分进行归一化，以平衡这种偏好
- 一个常见的归一化方式是：

$$\text{Final Score} = \frac{1}{L(Y)^\alpha} \sum_{t=1}^{T'} \log P(y_t)$$

- 其中 $L(Y)$ 是候选序列的长度， α 是一个超参数（通常在0.6到0.75之间），用于控制惩罚的强度。 $\alpha = 0$ 表示不惩罚， $\alpha = 1$ 表示完全按长度平均

束搜索的评分与长度惩罚

- 由于概率是多个小于1的数连乘，序列越长，总得分（概率）会越低。这会导致束搜索天然地偏爱更短的序列
- 为了解决这个问题，我们通常使用对数概率 (log-probability) 并进行长度归一化
- 候选序列得分：

$$Score(y_1, \dots, y_t) = \sum_{i=1}^t \log P(y_i | y_{<i}, \mathbf{c})$$

➤ 长度惩罚 (Length Penalty)

- 为了平衡对短序列的偏好，用序列长度的某个函数来对总分进行归一化。常见的归一化方式

$$FinalScore = \frac{1}{L^\alpha} \sum_{i=1}^L \log P(y_i | \dots)$$

- 其中 L 是候选序列的长度， α 是一个超参数（通常在0.6到0.75之间），用于控制惩罚的强度。 $\alpha = 0$ 表示不惩罚， $\alpha = 1$ 表示完全按长度平均
- 这个最终得分用于在束搜索的最后，从 k 个完整候选序列中选出最优的一个

评估生成序列：BLEU 分数

- 如何客观地衡量一个机器翻译的句子“好”还是“不好”？
 - BLEU (Bilingual Evaluation Understudy) 是一种广泛使用的自动化评估指标
- 好的翻译应该与人类专业译员的翻译在词组 (n-grams) 上有很高的重合度，由三部分构成：
 - 改进的n-gram精度 (p_n):
 - 计算预测序列中，长度为n的词组（如unigram, bigram...）出现在任一参考翻译中的比例
 - 为什么“改进”：使用“截断计数”。n-gram在预测中出现的次数，不能超过它在任何单个参考翻译中出现的最大次数
 - 目的: 惩罚无意义的重复。如，预测“the the the the”而参考是“the cat is on the mat”，普通精度会很高，但截断后“the”的计数只为2，精度会大幅下降
 - 短句惩罚 (Brevity Penalty, BP)
 - 如果预测序列比所有参考翻译都短，就施加一个惩罚项
 - 保证翻译的充分性。一个极短的预测可能很高的n-gram精度，但显然不是一个完整的翻译
- 最终公式:

$$\text{BLEU} = \text{BP} \cdot \exp \left(\sum_{n=1}^N w_n \log p_n \right)$$

- 通常 $N = 4$ ，权重 $w_n = 1/N$ （即几何平均）。BLEU分数越高，表示生成序列的质量越好

总结

- Encoder-Decoder 范式为一种强大的抽象架构，将复杂序列转换问题解耦为理解（编码）和生成（解码）两个阶段，其核心是通过一个固定大小的上下文向量传递信息
- Seq2seq 模型使用RNN来具体实现Encoder-Decoder范式的经典模型。它通过将编码器最终的隐状态作为解码器的初始隐状态来工作，但也因此存在信息瓶颈问题
 - 为Attention机制的奠定基础
- 束搜索为一种在计算成本和生成质量之间取得平衡的解码算法。通过在每一步保留 k 个最佳候选，近似寻找全局最优的输出序列，并通过长度惩罚来避免对短序列的偏好
- BLEU 分数: 一种衡量生成序列质量的评估指标，它通过匹配预测与参考翻译之间的 n-gram并惩罚过短的句子来工作，兼顾了准确性与充分性