

Seq2Seq Attention

目录

1. 问题背景：Seq2Seq 在解决什么？	2
2. 没有 Attention 的 Seq2Seq	2
2.1. Encoder-Decoder 基本结构	2
2.2. 固定长度瓶颈	2
3. Attention 的核心想法	3
4. Bahdanau Attention	3
4.1. Bahdanau 的直觉	4
5. Luong Attention	4
5.1. Dot	4
5.2. General	4
5.3. Concat	4
6. Seq2Seq Attention 的完整生成过程	5
7. Attention 矩阵和对齐	5
8. 训练时 Attention 怎么学出来？	5
9. 推理时怎么生成？	6
10. 和 Transformer Attention 的关系	6
11. 为什么 Attention 重要？	7
12. 常见误区	7
13. 总结	7

1. 问题背景：Seq2Seq 在解决什么？

Seq2Seq 是 sequence-to-sequence 的缩写，意思是把一个序列映射成另一个序列。典型任务包括：

1. 机器翻译：英文句子到中文句子
2. 摘要生成：长文本到短摘要
3. 语音识别：音频特征序列到文字序列
4. 对话生成：用户输入到模型回复

最早的神经网络 seq2seq 通常由两个 RNN 组成：

1. **Encoder**：从左到右读完整个源序列，得到隐藏状态
2. **Decoder**：根据 Encoder 给出的信息，从左到右生成目标序列

例如源序列是：

I love machine learning

目标序列是：

我 喜欢 机器 学习

Encoder 先读入英文，Decoder 再逐步生成中文。

2. 没有 Attention 的 Seq2Seq

2.1. Encoder-Decoder 基本结构

设源序列为：

$$x_1, x_2, \dots, x_n \quad (1)$$

Encoder RNN 逐步读取输入：

$$h_i = f_{\text{enc}}(h_{i-1}, x_i) \quad (2)$$

读完整个源序列后，通常取最后一个隐藏状态 h_n 作为整句话的压缩表示：

$$c = h_n \quad (3)$$

Decoder 用这个上下文向量 c 和已经生成的目标 token 来预测下一个 token：

$$s_t = f_{\text{dec}}(s_{t-1}, y_{t-1}, c) \quad (4)$$

$$P(y_t \mid y_1, \dots, y_{t-1}, x_1, \dots, x_n) = \text{softmax}(W_o s_t + b_o) \quad (5)$$

其中：

1. h_i 是 Encoder 第 i 个位置的隐藏状态
2. s_t 是 Decoder 第 t 个位置的隐藏状态
3. c 是 Encoder 传给 Decoder 的上下文向量
4. y_t 是目标端第 t 个 token

2.2. 固定长度瓶颈

没有 attention 的 seq2seq 有一个明显问题：无论源句子多长，Encoder 都要把所有信息压缩进一个固定长度向量 c 。

如果源句子很短，这还勉强可行：

I agree

但如果源句子很长：

The book that the professor recommended to the students yesterday is very useful

最后一个隐藏状态 h_n 很难完整保留所有细节。Decoder 生成目标句子后半部分时，可能需要回看源句子开头的信息，但它只能依赖已经被压缩进 c 的内容。

例 2.2.1: 翻译长句时，Decoder 在生成“这本书”时需要关注源句子的“The book”，在生成“教授推荐”时需要关注“the professor recommended”。固定向量 c 要同时保存所有这些信息，压力很大。

这个问题可以叫做固定长度瓶颈：源序列长度可变，但传给 Decoder 的信息通道长度固定。

3. Attention 的核心想法

Attention 的关键改动是：不要只把最后一个 Encoder 状态 h_n 交给 Decoder，而是把所有 Encoder 状态都保留下来：

$$h_1, h_2, \dots, h_n \quad (6)$$

Decoder 在生成每个目标 token 时，动态决定自己要看源序列的哪些位置。

生成第 t 个目标 token 时，Decoder 有当前状态 s_{t-1} 。它会拿这个状态去和每个 Encoder 状态 h_i 做匹配，得到一组分数：

$$e_{t,i} = \text{score}(s_{t-1}, h_i) \quad (7)$$

然后对这些分数做 softmax：

$$\alpha_{t,i} = \frac{\exp(e_{t,i})}{\sum_{j=1}^n \exp(e_{t,j})} \quad (8)$$

其中 $\alpha_{t,i}$ 表示：

$$\text{Decoder 在生成第 } t \text{ 个目标 token 时，应该从源序列第 } i \text{ 个位置拿多少信息} \quad (9)$$

最后把所有 Encoder hidden state 加权求和，得到当前步的上下文向量：

$$c_t = \sum_{i=1}^n \alpha_{t,i} h_i \quad (10)$$

注意， c_t 现在依赖 t 。也就是说，Decoder 每生成一个目标 token，都可以得到一个不同的上下文向量。

4. Bahdanau Attention

Bahdanau attention 也叫 additive attention。它的匹配函数通常写成：

$$e_{t,i} = v_a^T \tanh(W_s s_{t-1} + W_h h_i) \quad (11)$$

这里的参数 W_s, W_h, v_a 都是训练出来的。

计算流程是：

1. 用 W_s 变换 Decoder 上一步状态 s_{t-1}
2. 用 W_h 变换 Encoder 第 i 个状态 h_i
3. 两者相加后过 $\tanh$
4. 再和向量 v_a 做点积，得到标量分数 $e_{t,i}$
5. 对所有 i 的分数做 softmax，得到 attention 权重 $\alpha_{t,i}$
6. 用 $\alpha_{t,i}$ 对所有 h_i 加权求和，得到 c_t

可以把它理解成一个可学习的打分器：

$$\text{score}(s_{t-1}, h_i) \rightarrow \text{当前 decoder 状态和源位置 } i \text{ 有多相关} \quad (12)$$

Bahdanau attention 常被称为 additive，是因为它先把两个向量分别线性变换到同一空间，再相加后打分。

4.1. Bahdanau 的直觉

Decoder 状态 s_{t-1} 表示“我现在已经生成了什么，下一步大概需要什么”。

Encoder 状态 h_i 表示“源句子第 i 个位置及其上下文包含什么信息”。

Attention 分数 $e_{t,i}$ 衡量的是：

$$\text{我当前生成到这里时，源句子第 } i \text{ 个位置对我是否有用} \quad (13)$$

例如翻译：

I love machine learning

当 Decoder 要生成“机器”时，attention 权重可能主要落在“machine”上；当 Decoder 要生成“学习”时，权重可能主要落在“learning”上。

5. Luong Attention

Luong attention 也叫 multiplicative attention。它把匹配函数写得更接近点积形式。

常见形式有三种。

5.1. Dot

最简单的是直接点积：

$$e_{t,i} = s_t^T h_i \quad (14)$$

如果 s_t 和 h_i 方向越接近，分数越高。

5.2. General

加入一个可学习矩阵：

$$e_{t,i} = s_t^T W_a h_i \quad (15)$$

这比普通点积更灵活，因为 W_a 可以学习 Decoder 状态和 Encoder 状态之间的匹配空间。

5.3. Concat

也可以把两个向量拼接后打分：

$$e_{t,i} = v_a^T \tanh(W_a [s_t; h_i]) \quad (16)$$

这个形式和 Bahdanau attention 很接近。

6. Seq2Seq Attention 的完整生成过程

把 attention 加进 Decoder 后，第 t 步大致如下：

1. 根据上一步目标 token y_{t-1} 和上一步 Decoder 状态 s_{t-1} ，得到当前 Decoder 状态 s_t
2. 用 s_t 或 s_{t-1} 和每个 Encoder 状态 h_i 计算匹配分数 $e_{t,i}$
3. 对分数做 softmax，得到 attention 权重 $\alpha_{t,i}$
4. 用权重对 Encoder 状态加权求和，得到上下文向量 c_t
5. 把 s_t 和 c_t 拼接或融合
6. 通过输出层得到词表概率
7. 选择下一个目标 token y_t

可以写成：

$$s_t = f_{\text{dec}}(s_{t-1}, y_{t-1}) \quad (17)$$

$$\alpha_{t,i} = \text{softmax}_i(\text{score}(s_t, h_i)) \quad (18)$$

$$c_t = \sum_{i=1}^n \alpha_{t,i} h_i \quad (19)$$

$$o_t = \tanh(W_c[c_t; s_t]) \quad (20)$$

$$P(y_t \mid y_{<t}, x_1, \dots, x_n) = \text{softmax}(W_o o_t + b_o) \quad (21)$$

这里 $y_{<t}$ 表示目标端已经生成的前缀。

7. Attention 矩阵和对齐

如果源句子长度是 n ，目标句子长度是 m ，那么所有 attention 权重可以组成一个矩阵：

$$A \in \mathbb{R}^{m \times n} \quad (22)$$

其中：

$$A_{t,i} = \alpha_{t,i} \quad (23)$$

第 t 行表示生成第 t 个目标 token 时，Decoder 对所有源 token 的关注分布。

这个矩阵很像传统机器翻译里的词对齐。比如：

source: I love machine learning
target: 我 喜欢 机器 学习

理想情况下，attention 矩阵可能接近：

	I	love	machine	learning
我	high	low	low	low
喜欢	low	high	low	low
机器	low	low	high	low
学习	low	low	low	high

但 attention 不等于严格的一对一词对齐。它是软权重，一个目标 token 可以同时关注多个源 token；多个目标 token 也可以关注同一个源 token。

8. 训练时 Attention 怎么学出来？

训练数据通常只提供源句子和目标句子，不会直接提供正确的 attention 权重。

例如训练样本是：

source: I love machine learning
target: 我 喜欢 机器 学习

模型训练目标仍然是最大化正确目标序列的概率：

$$P(y_1, \dots, y_m \mid x_1, \dots, x_n) = \prod_{t=1}^m P(y_t \mid y_{<t}, x_1, \dots, x_n) \quad (24)$$

损失函数是每个目标位置的交叉熵：

$$L = - \sum_{t=1}^m \log P(y_t^{\text{gold}} \mid y_{<t}^{\text{gold}}, x_1, \dots, x_n) \quad (25)$$

如果某种 attention 权重能让正确 token 概率变高，它对应的参数就会被强化；如果某种 attention 权重让模型预测错，它对应的参数就会被削弱。

所以 attention 权重不是人工标注出来的，而是通过最终翻译任务的损失间接学出来的。

9. 推理时怎么生成？

推理时目标句子还不存在，Decoder 只能从左到右生成。

常见流程是：

```
encoder_outputs = Encoder(source)
y = <bos>
state = initial_state

while y != <eos> and step < max_len:
    state = DecoderRNN(state, y)
    weights = attention(state, encoder_outputs)
    context = weighted_sum(weights, encoder_outputs)
    probs = output_layer(state, context)
    y = decode(probs)
    append y to result
```

每一步都会重新计算一组 attention 权重。源序列的 Encoder 输出 $h_1, \dots, h_n$ 不变，但 Decoder 当前状态 s_t 会随着已生成前缀变化，所以 $\alpha_{t,i}$ 也会变化。

10. 和 Transformer Attention 的关系

Seq2seq attention 和 Transformer attention 思想是一脉相承的：都是用一个当前位置的状态去查询一组候选状态，然后按相关性加权取信息。

在 RNN seq2seq attention 中：

1. Decoder 状态 s_t 类似 Query
2. Encoder 状态 h_i 同时承担 Key 和 Value
3. 打分函数 $\text{score}(s_t, h_i)$ 负责计算 Query 和 Key 的相似度
4. 上下文向量 c_t 是对 Value 的加权和

在 Transformer cross-attention 中，这件事被写成更统一的矩阵形式：

$$Q = SW^Q \quad (26)$$

$$K = HW^K \quad (27)$$

$$V = HW^V \quad (28)$$

$$O = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (29)$$

其中 S 是 Decoder 端状态矩阵， H 是 Encoder 输出矩阵。

对应关系可以这样看：

1. RNN seq2seq 每个时间步算一次 c_t
2. Transformer cross-attention 一次性对所有目标位置算出上下文表示
3. RNN attention 常用 additive 或 dot/general/concat 打分
4. Transformer attention 使用 scaled dot-product，并且通过 multi-head 学多组关系

因此，Transformer 不是凭空发明了 attention，而是把 seq2seq attention 的“动态读取源序列”思想改造成更适合并行矩阵计算的形式。

11. 为什么 Attention 重要？

Attention 对 seq2seq 的意义主要有三点。

第一，它缓解了固定长度瓶颈。Decoder 不再只依赖一个 h_n ，而是每一步都可以访问所有 Encoder hidden states。

第二，它让生成过程具备动态检索能力。Decoder 生成不同目标 token 时，可以关注源句子不同部分。

第三，它提供了一定可解释性。Attention 矩阵可以展示目标 token 大致从哪些源 token 读取信息，虽然不能把它简单等同于严格解释。

12. 常见误区

1. **Attention 不是只选择一个源 token。**它通常是对所有源 hidden states 的软加权。
2. **Attention 权重不是人工标注的对齐结果。**它由最终任务损失间接训练出来。
3. **Attention 不是 Decoder 的全部。**Decoder 仍然需要 RNN 状态、目标端历史 token 和输出层。
4. **高 attention 权重不一定等于唯一因果解释。**它可以帮助理解模型行为，但不能完全代表模型推理原因。
5. **Seq2seq attention 和 Transformer attention 不是两套完全无关的东西。**Transformer cross-attention 可以看成 seq2seq attention 的矩阵化、并行化版本。

13. 总结

没有 attention 的 seq2seq 把源句子压缩成一个固定向量 c ，长句信息容易丢失。

Seq2seq attention 保留所有 Encoder 状态 $h_1, \dots, h_n$ 。Decoder 每生成一个目标 token，就根据当前状态计算一组权重 $\alpha_{t,i}$ ，再把源端状态加权求和得到当前上下文 c_t 。

一句话总结：

“Seq2Seq Attention 让 Decoder 在每一步生成时，都能动态回看源序列中最相关的位置。”