自然语言处理期末项目

基于循环神经网络的机器翻译的调研报告

中山大学计算机学院　计算机科学与技术

19335174　施天予

一、以序列到序列模型（seq2seq）为框架的神经机器翻译的原理.

什么是 seq2seq 呢? 简单的说, 就是根据一个输入序列 x, 来生成另一个输出序列 y。seq2seq 是一种常见的框架，不光用于神经机器翻译，在文本总结、对话系统、句法分析、代码生成中都有使用。神经机器翻译中 seq2seq 是一个条件语言模型：预测基于源语言句子，解码器由一个词预测下一个词。

对于 seq2seq 问题，有人提出了 encoder-decoder 模型。编码就是将输入序列转化成一个固定长度的向量；解码就是将之前生成的固定向量再转化成输出序列。具体实现的时候，编码器和解码器都不是固定的，输入语句和输出语句都包含多个词并且数量不一定相同。最简单的做法是先将整个输入语句编码成固定长度的向量表示，然后再逐步进行解码输出对应的翻译语句，Encoder 和 Decoder 都可以使用 RNN 来实现。当然也可以使用 Attention 的 Encoder 和 Decoder，比如 Transformer。

二、解释编码器和解码器各自的作用.

1. 编码器

从源语言中读取单词的输入序列，并将该信息编码为实值向量，也称为隐状态。该向量将输入序列的“意义”编码为单个向量。编码器输出被抛弃（除非使用 Attention），并把最后一个隐状态作为初始输入传给解码器。

2. 解码器

将来自编码器的隐状态作为输入,并将 <START> 标记作为初始输入,最后解码到 <END> 结束，生成输出序列得到翻译结果。解码器在不同阶段的工作方式互不相同，训练阶段有 free-running 模式、teacher-forcing 模式等, 解码策略有贪婪搜索、集束搜索等, 也可以配合 Attention 机制，具体过程见下文。

三、如何训练神经机器翻译？

1. 文本预处理

1. 清除双语语料的脏数据并进行分词
2. 构造词表，并将 <START> 和 <END> 也分别对应一个索引
3. 用所有源语言向量和目标语言向量构成一个 Dataloader

2. 单轮迭代训练过程

1. 从训练集的 Dataloader 取出一个 batch，得到输入向量 x 和目标向量 y
2. 将优化器梯度归零 zero_grad
3. 对输入向量 x 进行 pack_padded_sequence，统一句子长度
4. 将 x 输入编码器得到编码器输出和编码器隐状态
5. 将编码器隐态传递给解码器，解码器第一个输入为 <START>，如果设置了 attention，则还需将编码器的所有隐态输出传递给解码器
6. 解码器需要一步一步进行迭代（1 step 1 word）
   - 如果采用 Free-running，则将当前步的输出作为下一步的输入
   - 如果采用 Teacher-forcing，则每个时间步上的输入为正确目标句子中的词语
7. 当到最大序列长或输出为 <END> 时停止当前的输出
8. Loss 计算并回传
9. 用优化器对模型进行优化（梯度下降法）

3. 相关问题

1. 损失函数：CrossEntropyLoss
2. 可使用学习率衰减，迭代一定轮数后对学习率进行衰减，避免学习停滞
3. 为避免梯度回传时经过 RNN 的隐态，故需要将其 detach，否则会引起梯度爆炸、重复梯度计算等问题
4. 为避免梯度爆炸，还可以使用梯度裁剪

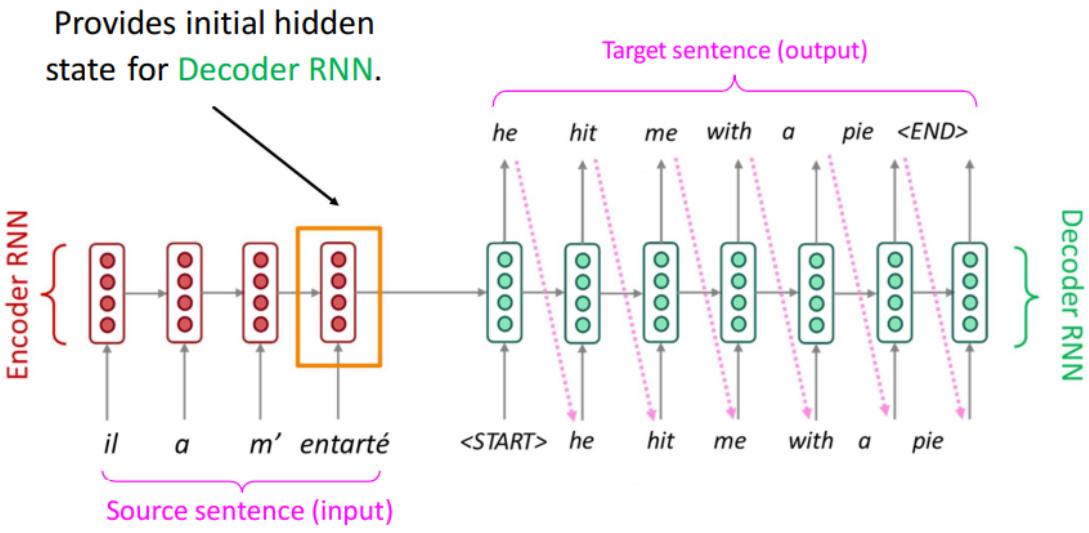

图 1: 神经机器翻译

四、怎样在训练阶段计算模型的损失 (误差)？

在解码器的每个 time step 使用交叉熵计算 loss。假设预测的概率分布是 $\hat{y}^{(t)}$，真实的单词表示是 $y^{(t)}$

$$J^{(t)}(\theta) = CE(y^{(t)}, \hat{y}^{(t)}) = -\sum_{w \in V} y_w^{(t)} log \hat{y}_w^{(t)}$$

最后计算所有 time step 的平均 loss

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J^{(t)}(\theta)$$

五、什么是 free-running 模式和 teacher-forcing 模式?

1. Free-running

RNN 在机器翻译中有 Free-running 和 Teacher-Forcing 两种模式。Free-running 就是我们常见的那种训练网络的方式：将上一个 state 的输出作为下一个 state 的输入，不论是训练阶段还是测试阶段都是如此。这种模式虽然具有识别错误的能力，也就是能根据 loss 不断从错误中学习，但是其也有许多问题：收敛速度慢，模型稳定性差，如果某个 state 的输出结果是错误的，那么输入到后面的 state 后结果会越来越糟糕。

2. Teacher-Forcing

正是由于训练迭代过程早期的 RNN 预测能力非常弱，几乎不能给出好的生成结果，因此产生了一种新的模式 Teacher-Forcing。Teacher-Forcing 在训练阶段不使用上一个 state 的输出作为下一个 state 的输入，而是直接使用训练数据的标准答案 (ground truth) 的对应上一项作为下一个 state 的输入，但其在测试阶段仍然需要上一个 state 的输出作为下一个 state 的输入。Teacher-Forcing 成功解决了 Free-running 的许多问题，它收敛快，且模型稳定性强，不光在机器翻译领域，在文本摘要、图像字幕的深度学习语言模型以及许多其他应用程序都至关重要。

然而 Teacher-Forcing 同样也存在缺点：正是因为其在训练阶段一直依赖标签数据，当它在测试阶段没有 ground truth 的支持时模型就会变得脆弱，自身不具有识别错误的能力。如果测试数据集与训练数据集来自不同的领域，模型的效果就会变差。

当然现在也有许多方法应对这一问题。比如在测试阶段使用集束搜索 (Beam Search)：在预测单词这种离散值的输出时，对词表中每一个单词的预测概率执行搜索，生成多个候选的输出序列，以优化翻译的输出序列。通过这种启发式搜索，可减小模型学习阶段与测试阶段效果的差异。另外还有一个 Teacher Forcing 的变种 Curriculum Learning 可以解决这个问题。Curriculum Learning 使用一个概率 p 去选择使用 ground truth 还是前一个 state 的输出作为下一个 state 的输入。这个概率 p 会随着时间的推移而改变，这就是所谓的计划抽样 (scheduled sampling)。训练会从 Teacher-Forcing 开始，慢慢地降低在训练阶段输入 ground truth 的频率。

六、介绍三种不同的神经机器翻译解码策略 (decoding strategies).

1. 祖先抽样 (Ancestral Sampling)

其实就是以 t 时刻前面所有的词为条件的概率分布里面，选取 t 时刻的单词，一直重复这样的过程，直到最后一个单词选取完毕。虽然此方法比穷举法节省时间，但是此方法是按照概率随机性取得单词，因此同一个句子可能会得到不同的翻译，所以这个方法基本不用。

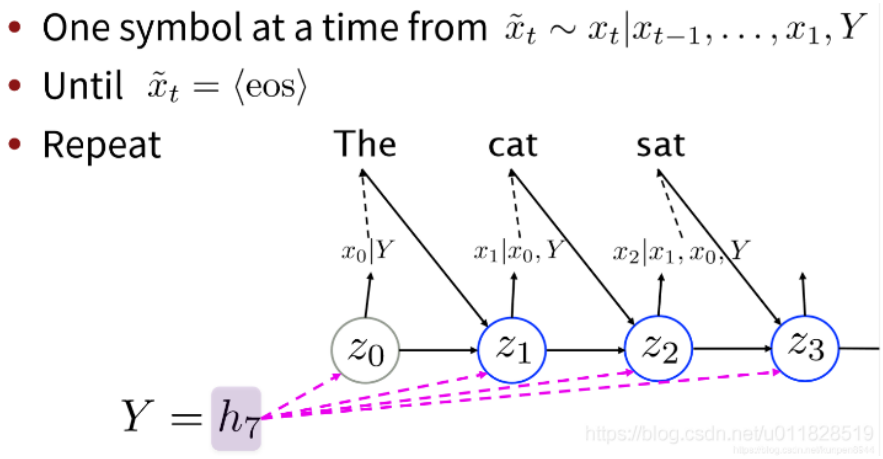

图 2: 祖先抽样 (Ancestral Sampling)

2. 贪婪搜索 (Greedy Search)

贪婪搜索，顾名思义就是在每一次选取单词的时候都根据目前条件概率选取概率最大的。虽然这样在时间和空间上都很有效率，但是这可能得不到全局的最优解，如果一个错了后面可能全错了，无法倒回去检查。从 <START> 开始解码，到 <END> 结束。

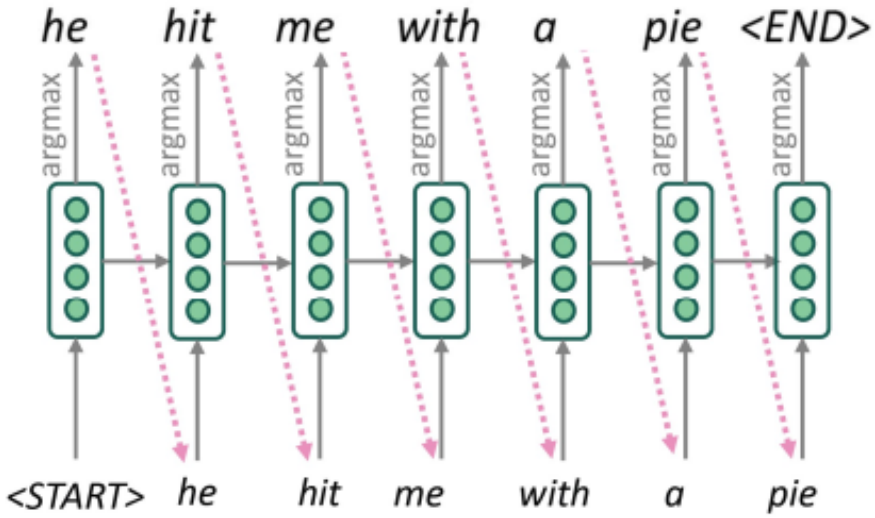

图 3: 贪婪搜索 (Greedy Search)

3. 集束搜索 (Beam Search)

核心思想：在解码器的每个 time step，保留 k 种最大可能的翻译结果。假设 k 是 beam size（一般取 5 到 10），每次计算的 log 概率公式是：

$$score(y_1, ..., y_t) = logP_{LM}(y_1, ..., y_t|x) = \sum_{i=1}^{t} logP_{LM}(y_i|y_1, ..., y_{i-1}, x)$$

scores 都是负的，越高越好，每一步保留 k 种最大可能的翻译结果。集束搜索不能保证得到最优的结果，但比贪婪搜索的效果要好很多。

在集束搜索中，不同的假设翻译结果可能在不同的 time step 产生 <END>。当一个假设翻译结果出现 <END>，该假设停止搜索。将 beam size 减 1，继续集束搜索，直到所有都出现 <END>。然而如果句子越长得到的 scores 会越低，所以要使用归一化进行比较再选出最高的 scores

$$\frac{1}{t}\sum_{i=1}^{t} logP_{LM}(y_i|y_1, ..., y_{i-1}, x)$$

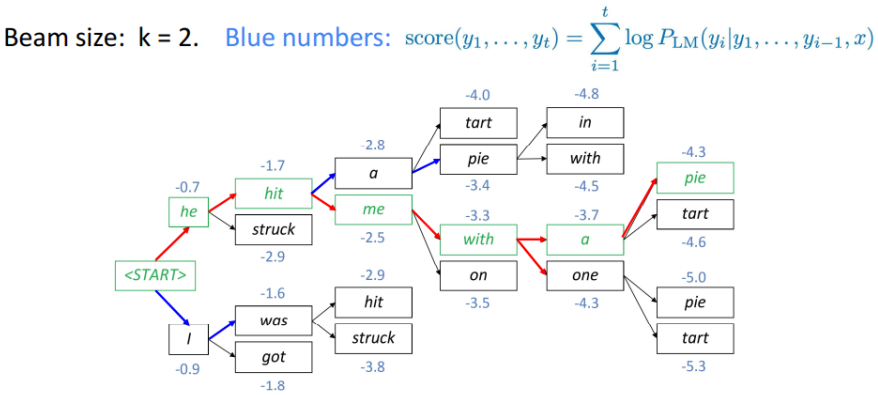

图 4: 集束搜索 (Beam Search)

当然也可以配合 Attention 机制进行解码，详见下一问。

七、为什么要在神经机器翻译中引入注意力机制 (attention)?

1. Encoder-Decoder 模型的不足

由于编码器在翻译时仅仅依赖于最后的隐藏状态，它的最后一个隐状态向量必须对源句子的所有内容进行编码，必须充分捕捉句子的含义。但是，问题在于我们无法将一个很长的句子所包含的所有信息编码成一个向量，很多的细节比如位置信息等就无法捕捉到，然后解码器仅仅根据这个向量生成很完美的翻译。假设原文句子有 50 个单词，英文译文的第一个单词可能与原文的第一个单词高度相关，这就意味着解码器必须考虑 50 步之前的信息，而且那边信息需要以某种方式编码到向量中。RNN 在处理这类长距离依赖的关系上会出现问题，LSTM 理论上能

够处理这类问题，但是在实际的工程中仍然是无效的。

2. Attention 使用

Attention 的出现成功解决了这一问题。Attention 机制，使得我们不需要将完整的原文句子编码为固定长度的向量，相反我们允许解码器在每一步输出时使用原文的不同部分，尤为重要的是让模型根据输入的句子和已经生成的内容决定使用什么。过程大致如下：

1. 首先我们有编码器的所有隐状态 $h_1, ..., h_N \in \mathbb{R}^h$
2. 在 time step t，我们有解码器隐状态 $s_t \in \mathbb{R}^h$
3. 接下来计算 attention scores

$$e^t = [s_t^T h_1, ..., s_t^T h_N] \in \mathbb{R}^h$$

4. 使用 softmax 将 attention scores 转为概率分布

$$\alpha^t = softmax(e^t) \in \mathbb{R}^h$$

5. 将编码器隐状态加权求和

$$a_t = \sum_{i=1}^{N} \alpha^t h_i \in \mathbb{R}^h$$

6. 最后将 attention 输出和 decoder 隐状态拼起来得出结果

$$[a_t; s_t] \in \mathbb{R}^h$$

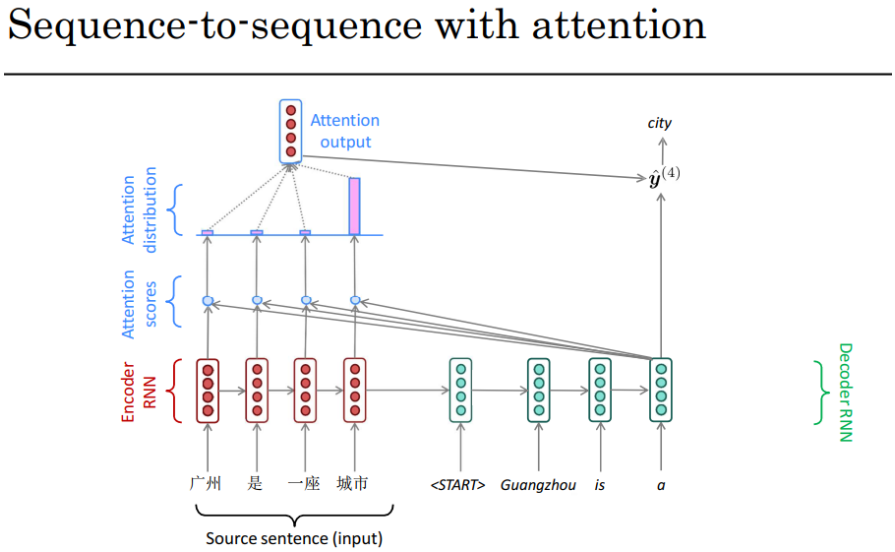

图 5: Attention 解码

3. Attention 优点

1. 极大提升 NMT 的效果，让 Decoder 关注源语言句子的局部对应信息十分有效
2. 解决了 Encoder-Decoder 的瓶颈问题：Encoder 的最后一个隐状态不足以编码长句子的全部信息，Attention 让 Decoder 可以直接关注源语言句子的局部对应信息
3. 解决梯度消失问题，可以砍掉很远的隐状态防止梯度消失
4. 提供可解释性
   (a) 观察 Attention 的概率分布，我们可以看到 Decoder 正在关注哪些信息
   (b) 可以自动让 Decoder 和 Encoder 进行软对齐
   (c) 而我们本来就不能显式训练一个 Decoder 和 Encoder 对齐的系统
   (d) 神经网络自己学习如何对齐

八、神经机器翻译（NMT）有什么优缺点?

1. 优点

1. 翻译效果更好：语句更流畅，更好利用语境信息
2. 编码器和解码器能组合成一个端到端网络进行优化，不用单独优化子部分
3. 需要更少的人力：不用设计特征，相同的方法适用所有的语言

2. 缺点

1. 可解释性差，难以 debug
2. 难以控制：难以制定合适的翻译规则，会有安全问题
3. 需要 GPU 计算资源和大量训练数据

九、解释机器翻译的评价指标 BLEU.

BLEU(Bilingual Evaluation Understudy) 是一种常用的机器翻译评价指标，其根据一或多个人工参考翻译，计算机器翻译结果和参考结果的相似度。BLEU 使用的是一种 n-gram 的 precision，即对照参考翻译结果，计算句子中正确的 n-gram 数量占整个句子中 n-gram 的比例，并且 BLEU 忽视了 recall。precision 越高，说明翻译结果越好。

$$BLEU = BP * exp(\sum_{n=1}^{N} w_n \log p_n).$$

$$\log BLEU = \min(1 - \frac{r}{c}, 0) + \sum_{n=1}^{N} w_n \log p_n$$

$$BP = \begin{cases} 1 & c > r \\ e^{1-\frac{r}{c}} & c \leq r \end{cases}, c \leftarrow len(candidate), r \leftarrow len(reference)$$

$$p_n = \frac{\sum_{C \in \{Candidates\}} \sum_{n-gram \in C} Count_{clip}(n-gram)}{\sum_{C' \in \{Candidates\}} \sum_{n-gram' \in C'} Count(n-gram')}$$

$$Count_{clip}(n-gram) = \min(Count, Max\_Reference\_Count)$$

图 6: BLEU 公式

下面是 BLEU 计算的几个例子：

在 Unigram 中，机器翻译的结果的每一个词在参考结果中共有 7 个匹配，而其本身有 10 个词，所以为 7/10。

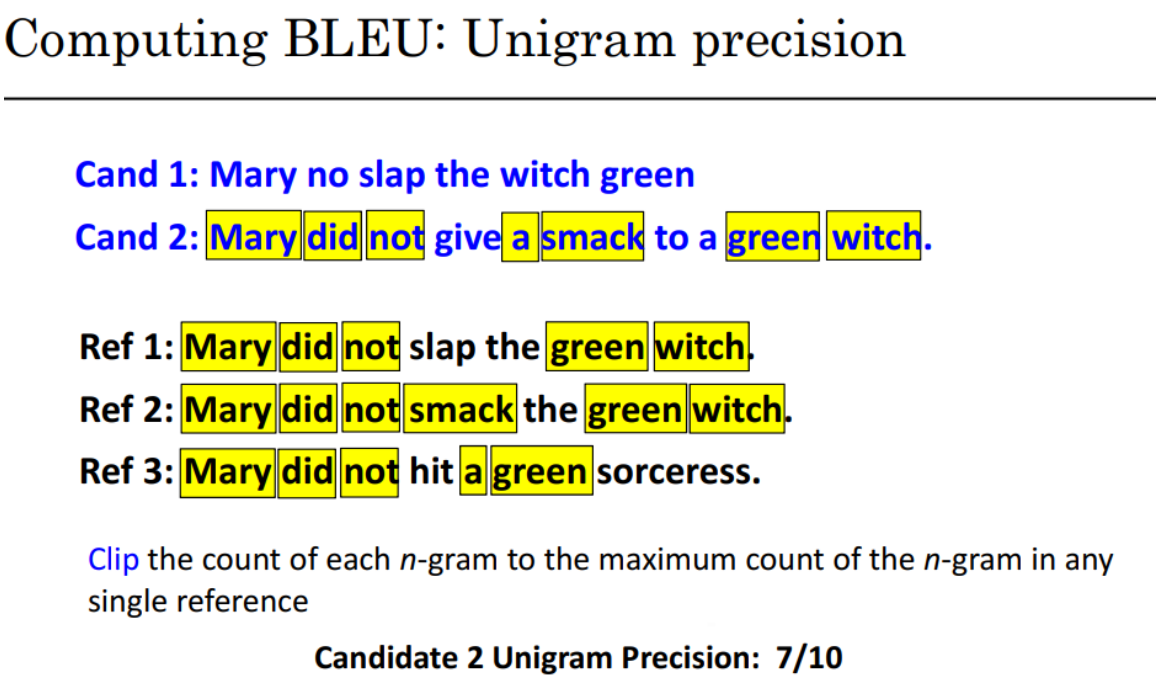

图 7: Unigram precision

在 Bigram 中，机器翻译的结果的每两个词在参考结果中共有 4 个匹配，而其本身有 9 个 Bigram，所以为 4/9。

Computing BLEU: Bigram precision

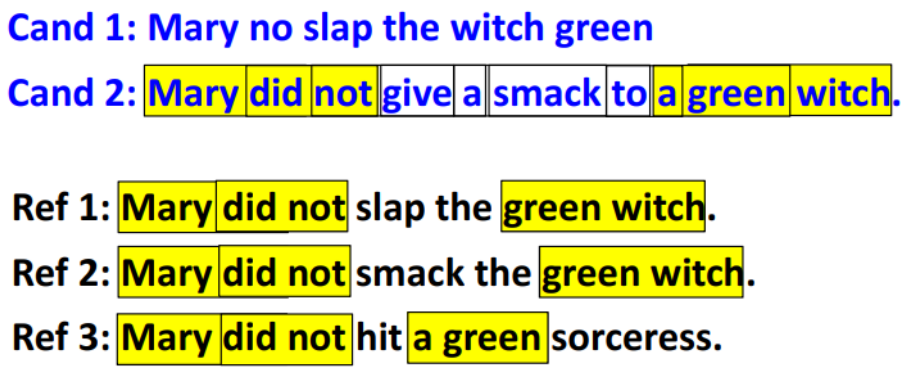

图 8: Bigram precision

另外 BLEU 当机器翻译结果的句子长度比参考结果短时，会有一个惩罚项。

$$\text{brevity-penalty} = \min\left(1\,,\ \frac{\text{output-length}}{\text{reference-length}}\right)$$

图 9: Penalty

让原来的 BLEU 乘以机器翻译句子长度与参考翻译长度的比值，获得新的 BLEU 公式。

$$\text{precision}_n = \frac{\sum_{C \in corpus} \sum_{\text{n-gram} \in C} \text{count-in-reference}_{\text{clip}}(\text{n-gram})}{\sum_{C \in corpus} \sum_{\text{n-gram} \in C} \text{count}\ (\text{n-gram})}$$

$$\text{BLEU-4} = \min\left(1\,,\ \frac{\text{output-length}}{\text{reference-length}}\right) \prod_{i=1}^{4} \text{precision}_i$$

图 10: 带惩罚项的 BLEU 计算公式