# 自然语言处理课程实验报告

*Koji Tadokoro*
*Shimokitazawa College*

## 1. 背景

机器翻译（machine translation）的目标是将给定的语句从一种源语言自动转换为一种目标语言。不同于涉及了翻译模型和语言模型的的基于统计方法的机器翻译，神经机器翻译（neural machine translation）主要基于神经网络方法，强调端到端的学习。处理此类问题的序列转换（sequence transduction）模型中的主流做法是基于卷积神经网络或循环神经网络实现编码器-解码器（encoder-decoder）架构 [2]。其中，编码器接受一个长度可变的序列作为输入，并将其转换为具有固定形状的编码状态，解码器则将固定形状的编码状态映射到长度可变的序列，如图 1 所示 [6]。

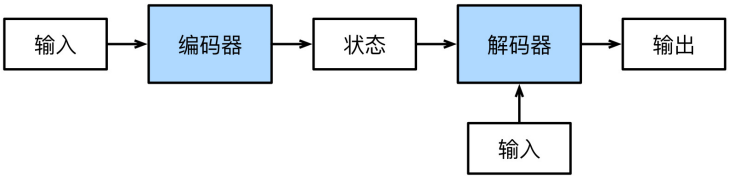


图 1. 编码器-解码器架构。

原始的编码器-解码器架构将输入序列编码为一个固定形状的状态，也称为上下文向量（context vector），随后解码器在每个时间步的解码中都会利用到这个上下文向量。而上下文向量受限于维度固定，难以将输入序列的信息都保存下来，同时如果采用循环神经网络作为编码器，由于循环神经网络的长程依赖问题，也容易丢失输入序列的信息。通过引入注意力机制，可以在每一步的解码过程中在输入序列中选取有用的信息，每个时间步都可以获得一个不同的上下文向量作为输入，从而获得更丰富的输入序列信息。

在本次实验中，主要参考了 Bahdanau 等人 [1]和 Luong 等人 [3]的工作，他们都采用了引入了注意力机制（attention mechanism）的基于循环神经网络的序列到序列模型，但在解码器与注意力机制结合的细节方面有所差别。

## 2. 要点简述

本小节就阅读文章时和实验中遇到的一些要点做出和讨论。

**模型的训练流程（Luong 等）.** 首先梳理一下 Luong 等人 [3]的模型的训练流程，用一个 1 层的 Bi-LSTM 作为编码器，用另一个 1 层的 vanilla LSTM 作为解码器，如图 2 所示。

给定一条用源语言表述的句子，我们根据源序列从嵌入矩阵（embedding matrix）中查询到对应的词嵌入 $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_m$（$\boldsymbol{x}_i \in \mathbb{R}^{e\times 1}$），其中 $m$ 为源序列的长度，$e$ 为嵌入向量的维度。再将这些嵌入送入双向编码器，得

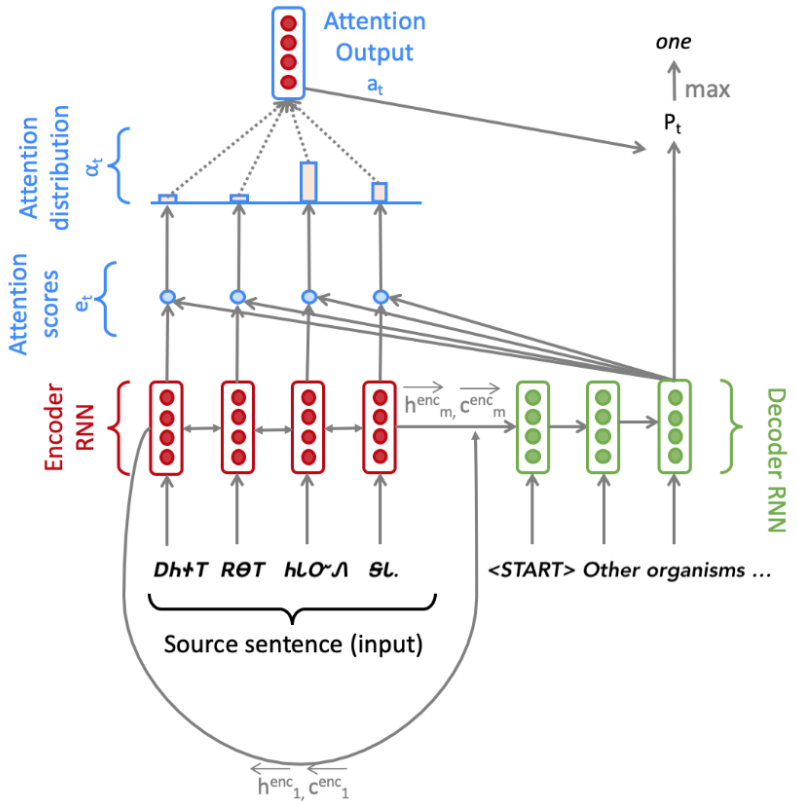


图 2. 引入注意力机制的序列到序列神经机器翻译模型，图例中正处于解码器的第 3 个时间步。

到前向 LSTM 和后向 LSTM 的隐状态（hidden states）和记忆单元状态（cell states），每个时间步的双向的隐状态和记忆单元状态都被拼接在一起（PyTorch 中 LSTM 的返回值是已拼接完的版本），即

$$\boldsymbol{h}_i^{enc} = [\overleftarrow{\boldsymbol{h}_i^{enc}}; \overrightarrow{\boldsymbol{h}_i^{enc}}], \quad \text{where } \boldsymbol{h}_i^{enc} \in \mathbb{R}^{2h\times 1}, \overleftarrow{\boldsymbol{h}_i^{enc}}; \overrightarrow{\boldsymbol{h}_i^{enc}} \in \mathbb{R}^{h\times 1} \qquad 1 \le i \le m \tag{1}$$

$$\boldsymbol{c}_i^{enc} = [\overleftarrow{\boldsymbol{c}_i^{enc}}; \overrightarrow{\boldsymbol{c}_i^{enc}}], \quad \text{where } \boldsymbol{c}_i^{enc} \in \mathbb{R}^{2h\times 1}, \overleftarrow{\boldsymbol{c}_i^{enc}}; \overrightarrow{\boldsymbol{c}_i^{enc}} \in \mathbb{R}^{h\times 1} \qquad 1 \le i \le m \tag{2}$$

在解码器端，我们将编码器最后的隐状态和记忆单元状态做一次线性变换以匹配解码器的输入维度，再作为解码器初始的隐状态 $\boldsymbol{h}_0^{dec}$ 和记忆单元状态 $\boldsymbol{c}_0^{dec}$：

$$\boldsymbol{h}_0^{dec} = \boldsymbol{W}_h[\overleftarrow{\boldsymbol{h}_1^{enc}}; \overrightarrow{\boldsymbol{h}_m^{enc}}], \quad \text{where } \boldsymbol{h}_0^{dec} \in \mathbb{R}^{h\times 1}, \boldsymbol{W}_h \in \mathbb{R}^{h\times 2h} \tag{3}$$

$$\boldsymbol{c}_0^{dec} = \boldsymbol{W}_c[\overleftarrow{\boldsymbol{c}_1^{enc}}; \overrightarrow{\boldsymbol{c}_m^{enc}}], \quad \text{where } \boldsymbol{c}_0^{dec} \in \mathbb{R}^{h\times 1}, \boldsymbol{W}_c \in \mathbb{R}^{h\times 2h} \tag{4}$$

在解码的第 $t$ 个时间步，查询到输出序列中第 $t$ 个 token 的词嵌入 $\boldsymbol{y}_t \in \mathbb{R}^{e\times 1}$（采取 teacher forcing 的训练方式），但并不直接将 $\boldsymbol{y}_t$ 送入解码器，我们将 $\boldsymbol{y}_t$ 与上一时间步的组合输出向量（combined-output vector）$\boldsymbol{o}_{t-1} \in \mathbb{R}^{h\times 1}$ 拼接得到 $\overline{\boldsymbol{y}}_t \in \mathbb{R}^{(e+h)\times 1}$，组合输出向量 $\boldsymbol{o}_{t-1}$ 包含了第 $(t-1)$ 个时间步解码时的注意力信息，在随后会叙述到如何得到 $\boldsymbol{o}_t$，特别地，对于首个时间步即输入为起始标记 <START> 来说，$\boldsymbol{o}_0$ 为零向量。总之，在第 $t$ 个时间步，将 $\overline{\boldsymbol{y}}_t$ 输入解码器，即

$$\boldsymbol{h}_t^{dec}, \boldsymbol{c}_t^{dec} = \text{Decoder}\left(\overline{\boldsymbol{y}}_t, \boldsymbol{h}_{t-1}^{dec}, \boldsymbol{c}_{t-1}^{dec}\right), \quad \text{where } \boldsymbol{h}_t^{dec}, \boldsymbol{c}_t^{dec} \in \mathbb{R}^{h\times 1} \tag{5}$$

我们用解码器输出的当前时间步的隐状态 $\boldsymbol{h}_t^{dec}$ 作为 *query*，编码器的所有时间步的隐状态 $\boldsymbol{h}_1^{enc}, \cdots, \boldsymbol{h}_m^{enc}$ 作为 *key* 和 *value* 来计算注意力分布 $\boldsymbol{\alpha}_t$ 和上下文向量 $\boldsymbol{context}_t$:

$$\boldsymbol{score}_{t,i} = (\boldsymbol{h}_t^{dec})^{\mathrm{T}} \boldsymbol{W}_{attnProj} \boldsymbol{h}_i^{enc}, \text{ where } \boldsymbol{score}_t \in \mathbb{R}^{m\times 1}, \boldsymbol{W}_{attnProj} \in \mathbb{R}^{h\times 2h} \qquad 1 \le i \le m \tag{6}$$

$$\boldsymbol{\alpha}_t = \mathrm{softmax}(\boldsymbol{score}_t), \text{ where } \boldsymbol{\alpha}_t \in \mathbb{R}^{m\times 1} \tag{7}$$

$$\boldsymbol{context}_t = \sum_{i=1}^{m} \boldsymbol{\alpha}_{t,i} \boldsymbol{h}_i^{enc}, \text{ where } \boldsymbol{context}_t \in \mathbb{R}^{2h\times 1} \tag{8}$$

其中，$\boldsymbol{score}_{t,i}$ 为标量，是 $\boldsymbol{e}_t \in \mathbb{R}^{m\times 1}$ 的第 $i$ 个元素，由解码器的第 $t$ 个时间步的隐状态 $\boldsymbol{h}_t^{dec}$ 和编码器的第 $i$ 个时间步的隐状态 $\boldsymbol{h}_i^{enc}$ 经由双线性（bilinear）注意力打分函数计算得到。

在预测输出序列中的下一个 token 时，将上下文向量 $\boldsymbol{context}_t$ 和当前的隐藏状态 $\boldsymbol{h}_t^{dec}$ 拼接之后经过线性层、激活函数 *tanh* 和 *dropout*，得到组合输出向量 $\boldsymbol{o}_t$。

$$\boldsymbol{u}_t = [\boldsymbol{context}_t; \boldsymbol{h}_t^{dec}], \text{ where } \boldsymbol{u}_t \in \mathbb{R}^{3h\times 1} \tag{9}$$

$$\boldsymbol{v}_t = \boldsymbol{W}_u \boldsymbol{u}_t, \text{ where } \boldsymbol{v}_t \in \mathbb{R}^{h\times 1}, \boldsymbol{W}_u \in \mathbb{R}^{h\times 3h} \tag{10}$$

$$\boldsymbol{o}_t = \mathrm{dropout}\left(\tanh(\boldsymbol{v}_t)\right), \text{ where } \boldsymbol{o}_t \in \mathbb{R}^{h\times 1} \tag{11}$$

最后，利用组合输出向量 $\boldsymbol{o}_t$ 得到一个与词表相关联的概率分布 $\boldsymbol{P}_t$:

$$\boldsymbol{P}_t = \mathrm{softmax}(\boldsymbol{W}_{vocab} \boldsymbol{o}_t), \text{ where } \boldsymbol{P}_t \in \mathbb{R}^{V_t\times 1}, \boldsymbol{W}_{vocab} \in \mathbb{R}^{V_t\times h} \tag{12}$$

其中，$V_t$ 为目标语言词表的大小。

**与 Bahdanau Attention 的不同.** 两篇工作中在注意力机制的计算方面最大的区别在于 *query* 的选取不同和计算路径不同。

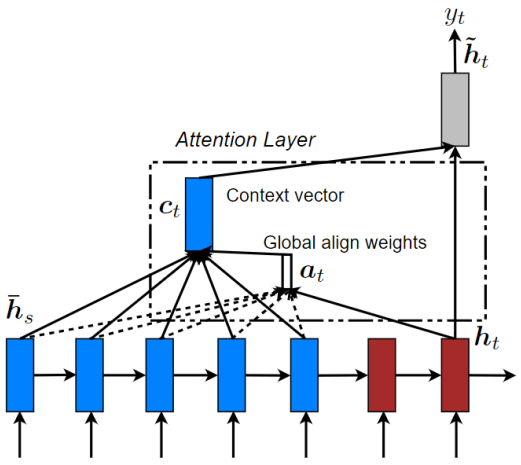


图 3. Luong 等人提出的注意力模型（全局注意力）。

Luong 等人 [3]的全局注意力模型如图 2 所示（符号记法与实验报告中不全相同）。在刚才的流程梳理中，Luong 等人提出的模型中计算的路径更为简单：在第 $t$ 个时间步，将 $\boldsymbol{y}_t$ 与上一时间步的组合输出向量 $\boldsymbol{o}_{t-1}$ 拼接作为解码器的输入，将解码器得到的当前时间步的隐状态 $\boldsymbol{h}_t^{dec}$ 作为 *query*，编码器的所有隐状态

$\boldsymbol{h}_1^{enc}, \cdots, \boldsymbol{h}_m^{enc}$ 作为 *key* 和 *value* 来计算注意力分布 $\boldsymbol{\alpha}_t$ 和上下文向量 $\boldsymbol{context}_t$，再将上下文向量 $\boldsymbol{context}_t$ 和 $\boldsymbol{h}_t^{dec}$ 拼接后变换得到 $\boldsymbol{o}_t$，直接用以做预测的决策。计算路径可以归纳为：$\boldsymbol{h}_t^{dec} \rightarrow \boldsymbol{\alpha}_t \rightarrow \boldsymbol{context}_t \rightarrow \boldsymbol{o}_t$。

在 Bahdanau 等人 [1]的工作中，计算过程则是这样的：在第 $t$ 个时间步，用解码器上一时间步的隐状态 $\boldsymbol{h}_{t-1}^{dec}$ 作为 *query*，编码器的所有隐状态 $\boldsymbol{h}_1^{enc}, \cdots, \boldsymbol{h}_m^{enc}$ 作为 *key* 和 *value* 来计算注意力分布 $\boldsymbol{\alpha}_t$ 和上下文向量 $\boldsymbol{context}_t$，再将输入 token 的词嵌入 $\boldsymbol{y}_t$ 与上下文向量 $\boldsymbol{context}_t$ 拼接之后送入解码器，得到当前时间步的隐状态 $\boldsymbol{h}_t^{dec}$，以 $\boldsymbol{h}_t^{dec}$ 作为预测的决策依据。过程如图 4 所示。计算路径可以归纳为：$\boldsymbol{h}_{t-1}^{dec} \rightarrow \boldsymbol{\alpha}_t \rightarrow \boldsymbol{context}_t \rightarrow \boldsymbol{h}_t^{dec}$。

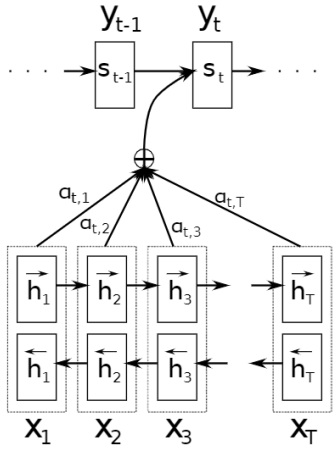


图 4. Bahdanau 等人提出的注意力模型。

可以看出，在 Luong 等人的工作中，模型在输入经过解码器之后计算注意力，再将上下文向量与隐状态融合之后经过一系列变换用以决策；而在 Bahdanau 等人的工作中，模型计算注意力之后，再将得到的上下文向量和输入融合送入解码器。两者的区别可以看作是模型中注意力层和 LSTM 层的先后次序的选择问题。

**对过去时刻注意力信息的利用.** 在 Bahdanau 等人的工作中，可以注意到每个时间步的注意力信息的计算都是相互独立的，没有考虑到当前时间步之前的输出序列的注意力信息。在统计机器翻译（statistical machine translation）中，系统会维护一个覆盖集（coverage set），用来追踪哪些源序列中的词已经在过去的时间步中翻译过了，减少出现过翻译（over-translation）或欠翻译（under-translation）现象。过翻译指在生成的目标序列中，源序列的一些词被不必要地翻译出多次的现象；欠翻译指的是源序列中一些词没有被翻译出来的现象，也有尝试在 NMT 系统中对覆盖集进行建模以改善翻译效果的工作 [5]。

在 Luong 等人的工作中，提出了一种在解码过程中近似实现覆盖集提示作用的方法。具体来说，将每个时间步的对齐信息（即注意力信息）和下个时间步的输入用拼接的方式融合在一起，再将融合后的表示送入解码器中，如式 5 所示。总之，采用这种方式可以让模型充分利用到之前时间步做出的对齐决策，也有利于模型做得兼有深度和广度。

**BLEU分数.** BLEU（Bi-Lingual Evaluation Understudy）是一种衡量模型生成序列和参考序列之间的 $n$ 元语法（$n$-gram）重合度的算法 [4]，最早用来评价机器翻译模型的译文质量。

在计算过程中，首先计算 $n$ 元组合的精度（precision），即生成序列中的 $n$ 元组合在参考序列中出现的次数；再对生成序列中长度短于参考序列长度的这部分进行惩罚；最后，对不同长度的 $n$ 元组合（$n = 1, 2, \cdots$）

的精度进行几何加权平均得到 BLEU 分数。一般来说，最多取到 $n = 4$，在 nltk 和 sacrebleu 中实现的就是 BLEU-4 分数。另外，BLEU 分数按照原始计算方式得到的是介于 0 到 1 之间的值，但也有采用将原始分数乘 100 之后的值作为 BLEU 分数的表达，本文中的 BLEU 分数用的是后一种表达方式。

考虑一个这样的例子：数据集中提供了 2 条参考的翻译序列，模型在两次训练中对同一句源语言序列生成了 2 条不同的翻译序列，即：

> **参考翻译 1:** *the light shines in the darkness and the darkness has not overcome it*
>
> **参考翻译 2:** *and the light shines in the darkness and the darkness did not comprehend it*
>
> **模型翻译 1:** *and the light shines in the darkness and the darkness can not comprehend*
>
> **模型翻译 2:** *the light shines the darkness has not in the darkness and the trials*

不难看出**模型翻译 1** 的语义更加符合两句参考序列，也在语义上是通顺并正确的。通过计算可以得到**模型翻译 1** 和**模型翻译 2** 的 BLEU-2 分数分别为 87.71 和 79.66，**模型翻译 1** 取得了更高的 BLEU 分数，这种情况下 BELU 分数和人评估的结果是一致的。

但如果提供的参考序列中，只提供了**参考翻译 1**，没有**参考翻译 2**，即目前情况如下：

> **参考翻译 1:** *the light shines in the darkness and the darkness has not overcome it*
>
> **模型翻译 1:** *and the light shines in the darkness and the darkness can not comprehend*
>
> **模型翻译 2:** *the light shines the darkness has not in the darkness and the trials*

这样的情况下计算得到的**模型翻译 2** 的 BLEU-2 分数依然保持在 79.66，而**模型翻译 1** 只得到了 71.61 的 BLEU 分数，这样的结果显然和人的评估结果不一致。

从这样的例子中可以看出：BLEU 分数仅依靠统计序列中的 $n$ 元语法组合计算得到，只要一条翻译序列与参考序列的 $n$ 元语法组合重合度比其他候选序列更高，它就能取得更高的 BLEU 分数，所以高的 BLEU 分数并不能代表这句话在语法和语义等方面组织和表达得更好。不过当有多条参考序列可供比对时，好的翻译序列还是更可能得到一个公平的的 BLEU 分数。

## 3. 实验设置与实验结果

### 3.1. 实验 1

**实验设置.** 数据集来源于 Tatoeba（たとえば）在线数据库[1]的“英-中”双语平行语料，其中每行都由一条英文句子和中文句子组成，在实验中，约定英文为源语言（source language），中文为目标语言（target language）。基于 PyTorch Tutorial 相关章节[2]实现，实现的为英-中翻译版本[3]。

原始数据中繁体中文和简体中文混杂，首先使用了 OpenCC 工具将繁体中文转换为简体中文在预处理阶段分别尝试了使用 jieba 和 LTP 工具对原文本进行分词，jieba 分词速度快于 LTP，但在有些常见分词场景如对“这本书不错”进行切分，用 jieba 处理得到的结果是“这/本书/不错”，LTP 则能给出正确的划分“这本书/不错”。

[1] http://www.manythings.org/anki.

[2] NLP from scratch: translation with a sequence to sequence network and attention.

[3] 实验实现的英-中翻译版本.

数据集中共有 29155 条双语文本对。原数据集中序列长度的方差较大，初期尝试将所有数据都送入模型中全量训练，但效果很差，所以在后续处理中沿用了 PyTorch Tutorial 里的做法，筛选出文本对中的英文句子中以“*I am*”、“*You are*”等开头的这部分数据作为训练数据，共 5346 条文本对。

**实验结果.** 用训练完的模型对测试集进行预测，得到的 Corpus BLEU 为 79.81。图 5 和图 6 是两句英文-中文翻译的注意力分数的可视化。

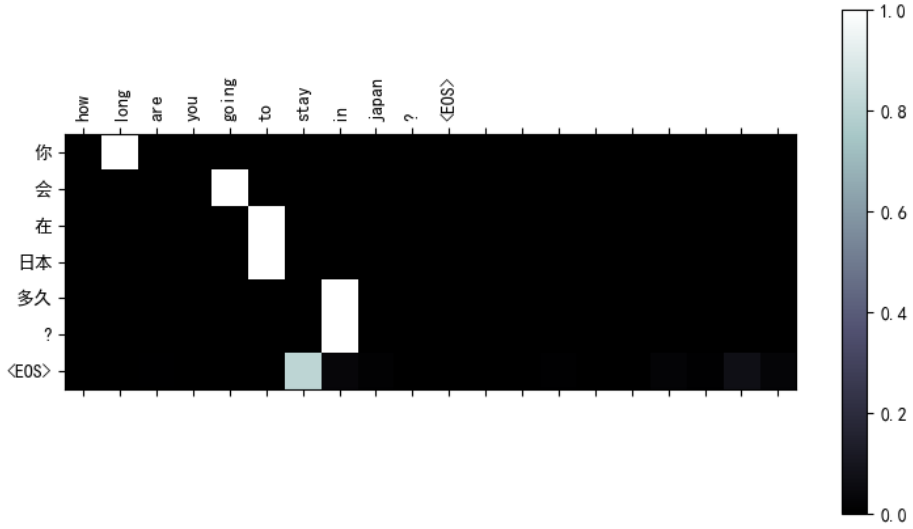


图 5. 翻译“*how long are you going to stay in japan ?*”的可视化注意力。

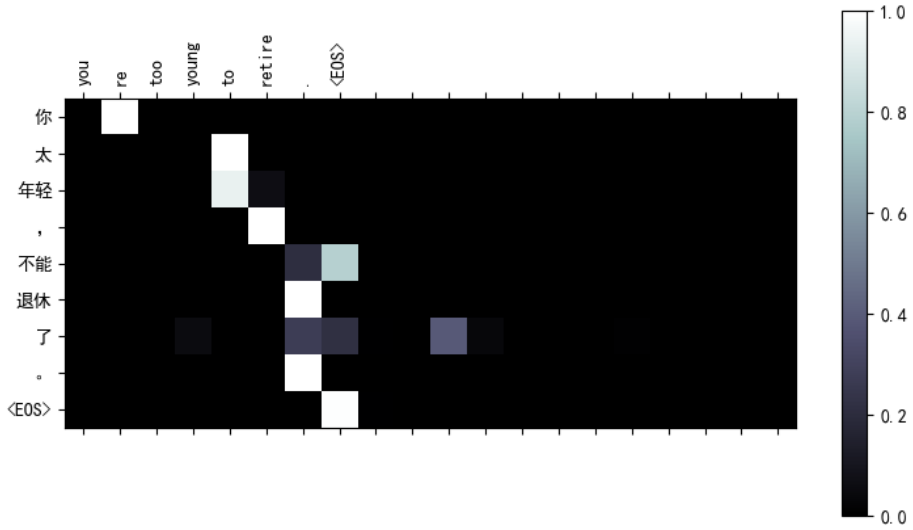


图 6. 翻译“*what do you think about this ?*”的可视化注意力。

**实验分析.** 我们的模型翻译“*Do you know how many people in the world starve to death every year?*”这句话的结果和参考翻译如下：

**参考翻译：** 你 /知不知道 /全世界 /每年 /有 /多少 /人 /饿死 /呀 /？

**模型翻译：** 你 /知不知道 /全世界 /有 /多少 /人 /饿死 /呀 /饿死 /呀 /呀 /呀 /饿死 /呀 /呀 /呀 /饿死 /呀 /呀

其中，“饿死”和“呀”被不断重复，直到解码出的序列达到设定的序列最大长度才停止，这就是之前所提到过的过翻译现象。在我们的神经机器翻译系统中出现这种现象的原因可能在于模型在后面一些时间步的解码过程中一直关注源序列中的同一部分，会一直计算出非常相近的注意力分布，通过引入自注意力可能有助于改善这一现象。

### 3.2. 实验 2

**实验设置.** 数据集为“Cherokee 语-英语”数据集，共 18696 条双语文本对，采用了 90:5:5 的数据划分。实验 2 是对 CS224n 作业 4 的补全。

**实验结果.** 模型在测试集上的 Corpus BLEU 达到了 12.73（BLEU 分数范围为 0 到 100）。

**实验分析.** 如下这句翻译在开头（“*Wilbur was*”）还与参考序列一样，但后面的内容则完全偏离了，但是生成的翻译没有语法问题，并且也表达了清晰通顺的语义，这显示机器翻译系统可能通过某种方式学习到了在解码时根据语言模型在目标语言上构建输出序列。

**参考翻译:** *Wilbur was going to sleep.*

**模型翻译:** *Wilbur was pleased to receive so much attention.*

## 参考文献


[1] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations (ICLR)*, 2015. 1, 4

[2] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014. 1

[3] T. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1412–1421, 2015. 1, 3

[4] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics(ACL)*, pages 311–318, 2002. 4

[5] Z. Tu, Z. Lu, Y. Liu, X. Liu, and H. Li. Modeling coverage for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics(ACL)*, pages 76–85, 2016. 4

[6] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021. 1