MTG 卡牌翻译研究

杜岱玮 2011421　徐百朋 2012109

1 摘要

MTG 是一款风靡全球的集换式卡牌游戏。由于各国玩家所使用的语言各不相同，以及每年都有大量的新卡牌推出，翻译卡牌的工作量十分繁重。卡牌的规则文字描述这张牌如何在游戏过程中发挥作用，为了使它精确易读，卡牌的作者往往使用固定且风格独特的语言模式（可以类比为法律条文使用的语言）进行卡牌的描述，因此通用的机器翻译模型往往表现不佳。本研究旨在构建一种机器翻译模型，它能够相对准确地将卡牌的英文描述翻译为中文描述。首先，我们将卡牌网站上的数据整理成了数据集并将它开源。然后我们搭建了基于 RNN 和 Transformer 的翻译模型并进行了消融实验。我们根据消融实验的结果，对翻译模型进行改进以提升翻译质量。我们效果最好的模型在测试集上的 BLEU score 是 69。

2 引言

2.1 背景介绍

Magic: The Gathering（简称“MTG”，中文名“万智牌”）是一款风靡全球的集换式卡牌游戏，发明于 1993 年，由美国的威世智公司出品。然而，万智牌的大部分卡牌为英文卡面，缺少其他国家语言的翻译，给英文不好的玩家带来了很多麻烦。因此，翻译万智牌这项工作的意义在于方便各国的玩家，使得更多的玩家体验到万智牌的乐趣。

一张 MTG 卡牌的文字可以分成三个部分：卡牌名称、规则文字、风味文字。图 1 展示了一张英文卡牌”You Are Already Dead”，红色方框区域是卡牌名称，绿色方框区域是规则文字，蓝色方框区域是风味文字。卡牌名称是一张卡的独有标识；规则文字描述了这张卡如何在游戏中发挥作用；风味文字与游戏无关，仅仅为卡牌增加“风味”。

图 1　卡牌“You Are Already Dead”

图 2　MTG 卡牌”Charge of the Mites” 的英文和中文版本

2.2 任务难点

我们在本次作业中探讨了 MTG 卡牌规则文字的机器翻译，这项任务有两个主要难点。

第一，不存在直接可用的公开数据集。据我们所知，此前没有人做过 MTG 卡牌的英文到中文机器翻译，因此也不存在已经处理好的双语对照数据集。我们必须从头进行数据获取、数据清洗、数据增强等一系列工作。

第二，规则文字非常注重文字的精确性、具有特殊的语言风格，且包含大量专有名词，为翻译带来一些困难。下面是规则文字的一些特点：

- 规则文字包含有大量的专有名词以及翻译惯例。图 2 是一张典型的 MTG 卡牌，它的规则文字很好地展示了这个问题。其中，“planeswalker”和“Phyrexian”是 MTG 的专有名词，在官方中文版中被音译为“鹏洛客”和“非瑞人”；“token”则是 MTG 的术语，应当被翻译为“衍生物”（不熟悉 MTG 的人也许会把它翻译成“代币”，正如我们将在下面的百度翻译结果中看到的）。

- 规则文字的中文和英文版本都包含一些独特的文法结构，并且两种语言版本之间有固定的对应关系。例如，英文版卡牌经常用”target X with Y” 描述一张卡牌的目标，Y 是对目标 X 的限定，固定的中文译法是“目标 Y 的 X”。因此，”Destroy target creature with power 3 or less” 应当被翻译为“消灭目标力量小于等于 3 的生物”，而不是“用小于等于 3 的力量消灭目标生物”。

- 规则文字含有一些具特定含义的特殊符号，比如”{r}” 代表“红色法术力”。这些特殊符号在不同语言的版本之间应当保持一致。

由于规则文字与日常语言之间的风格差异，通用的翻译模型常常对 MTG 卡牌的规则文字部分产生糟糕的翻译结果。例如，表 1 展示了百度翻译对卡牌”Charge of the Mites”（图 2）的一段规则文字的翻译。

它的这段译文非常糟糕，这有两个方面的原因。首先是句子结构理解错误，英文版的”with” 应当修饰“Mite”，而百度翻译以为它修饰”Create”，结果是语义完全翻译错了。其次，还有很多专名和术语的翻译错误，把”token” 翻译为“代币”尤为不可接受。

英文版： Create two 1/1 colorless Phyrexian Mite artifact creature tokens with toxic 1 and ”This creature can’t block.” (Players dealt combat damage by them also get a poison counter.)
中文版： 派出两个 1/1 无色非瑞人/虫械衍生神器生物，且具有下毒 1 与“此生物不能进行阻挡。”（受到其战斗伤害的牌手还会得到一个中毒指示物。）
百度翻译： 用有毒的 1 和“这个生物不能阻挡”创建两个 1/1 无色的菲雷仙 Mite 神器生物代币。（受到战斗伤害的玩家还可以获得一个毒药计数器。）

表 1 百度翻译对一段规则文字的翻译

2.3 我们工作的亮点

我们工作的亮点主要在于根据问题的特性提出创新性的改进方法，并且最终达到了相当好的翻译效果。下面列举几个亮点，它们都会在后文中被着重解释：

- 创建了 MTG 卡牌规则文字的双语对照数据集，并且尝试了多种数据增强方法，创建了数据集的多个版本并评估它们的效果。

- 针对卡牌名称风格多变、翻译难度极高的特点，我们决定放弃卡牌名称的翻译，直接把英文的卡牌名称嵌入中文翻译中。为此，我们提出了多模型组合的翻译方法，在使用序列到序列翻译模型进行翻译之前，先由名称检测模型标记出输入文本中的所有卡牌名称，最后用一些技巧把英文名称嵌入翻译结果。

- 针对规则文字中专有名词数量多、样本少的特点，我们提出了标签替换方法，用专有名词字典改善翻译结果，同时为用户提供了一定的控制模型翻译结果的能力。

- 由于规则文字的语言模式较为固定，翻译效果相当好，在测试数据集的 bleu score 高达 69。

3 相关工作

万智牌规则翻译属于特定领域的翻译任务，我们很难找到同一主题的相关工作。与此同时机器翻译这一任务由来已久，因此本文参考的主要是机器翻译领域的相关工作。我们的翻译模型基于 Encoder-decoder 架构，使用了 transformer 和双向 RNN 这两种实现。

下面是这两种架构的工作介绍：

3.1 Neural machine translation by jointly learning to align and translate

Bahdanau 等人 2015 年在这篇论文中提出了 RNNsearch 模型[1]，这是第一个采用注意力机制的机器翻译模型。RNNsearch 模型采用编码器-解码器结构，其中编码器和解码器都使用 RNN。

3.2 Attention is all you need

这篇文章主要介绍了”transformer“网络结构。”transformer“是一种先进的深度学习网络结构，引入了自注意力机制，自注意力模型可以通过神经网络自己来生成注意力，以此来判断词与词，句与句之间的依赖关系。此外，transformer 还解决了两个重要的问题：

- 长程依赖问题。由于词与词之间的联系直接取决于注意力，而不是像 rnn 或者 lstm 依靠那样跨越多层的长连接，因而避免了由于梯度消失导致的问题。
- 并行计算问题：由于 transformer 大量使用矩阵计算，可以简单高效地进行并行计算，提高了模型训练的速度。

4 问题定义

- 任务定义：给定万智牌的英文，我们将万智牌的规则文字由英文翻译为中文，尽可能用词准确，逻辑正确，且符合万智牌规则的语言惯例。
- 评价标准：我们在测试集上计算 BLEU 指标，BLEU 越高，说明翻译的越好。
- 数据集：我们使用了自制的数据集，原始数据从相关网站下载，划分为训练集、验证集和测试集三部分，使用 json 格式存储。每条数据包含一张卡片上的所有规则文字的中英文对照，测试集约 37000 条数据，验证集和测试集各有约 1000 条数据。其中，测试集选用了最新的两个系列的卡牌进行验证，而训练集避开了这两个系列。这样更符合未来模型实际工作的场景，也对模型的泛化能力提出了更高要求。

5 方法

我们先从总体上介绍翻译系统的架构，然后再就技术细节进行详细讨论。

5.1 翻译流程

我们构建的翻译系统并不是一个完全的端到端系统。在端到端的深度学习模型之外，我们还创新性地添加了一些额外的组件，用来针对卡牌特性提高翻译质量，并提高翻译系统的可扩展性。

图 4 展示了我们的翻译系统的总体架构，它以某张英文卡牌的规则文字为输入，以翻译为中文的规则文字为输出。

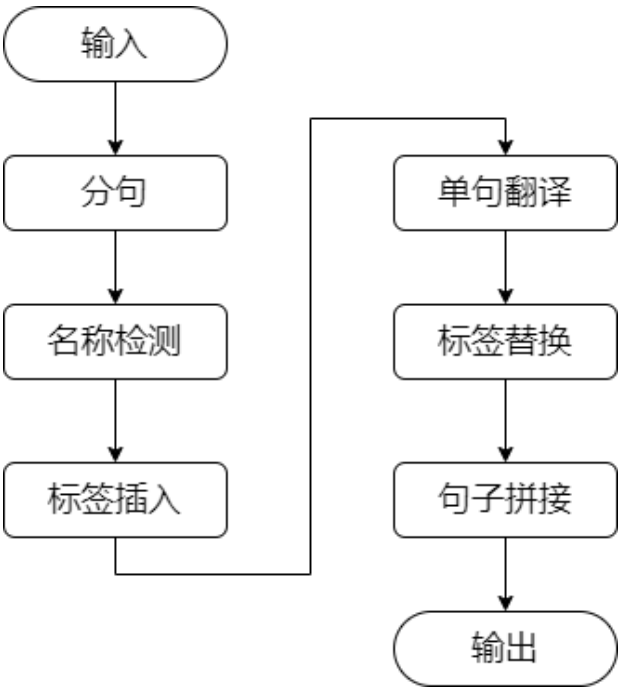

图 3 翻译流程

接下来我们还会具体介绍每一部分的功能和细节，同时以卡牌 Serum Sovereign 为例进行演示，它的规则文字英文版是：Flying Whenever you cast a noncreature spell, put an oil counter on Serum Sovereign. {U}, Remove an oil counter from Serum Sovereign: Draw a card, then scry 2.

标准中文翻译是：飞行 每当你施放非生物咒语时，在浆液君王上放置一个烁油指示物。{U}，从浆液君王上移去一个烁油指示物：抓一张牌，然后占卜 2。

5.1.1 分句

MTG 卡牌的规则文字长度不一，比较长的规则文字可能包含多个句子。序列到序列翻译模型在翻译较长的文本时可能遇到问题，例如，使用可学习位置编码的 Transformer 对输入序列的长度有限制。为了避免这些问题，我们决定把输入的规则文字拆分为句子，以句子为单位分别进行翻译，最后再把所有句子拼接在一起。

句子拆分也有坏处，它导致模型不能利用句子间的依赖关系。然而，万智牌规则文字表述严谨，通常每个句子都是完整的表意单位，把句子拆开翻译不会对模型造成很大的影响。

Serum Sovereign 的规则文字被按照换行符和句号分割为三个句子：

1. Flying

图 4 Serum Sovereign

2. Whenever you cast a noncreature spell, put an oil counter on Serum Sovereign.

3. {U}, Remove an oil counter from Serum Sovereign: Draw a card, then scry 2.

5.1.2 名称检测

万智牌的卡牌名称包含大量人名地名，词汇表很大，语法灵活多变，经常涉及背景知识。对于我们的小规模翻译模型而言，翻译卡牌名称是非常困难的，因此我们决定放弃卡牌名称翻译，直接把英文名称嵌入中文翻译中。

直接训练序列到序列模型原封不动输出卡牌名称是困难的，因为输入词汇表和输出词汇表不同，不可能把卡牌名称包含的英文单词都加入输出词汇表。所以，我们在翻译模型之外额外训练了一个卡牌名称检测模型。名称检测模型也是序列到序列模型，它判断输入的每个单词是否是卡牌名称的一部分，然后把所有检测到的卡牌名称标注出来。

我们把所有检测到的名称用标签 <id> 代替。标签是一种特殊的 token，代表一个模型不认识的词，不同标签以 id 区分。翻译模型会在翻译结果中保留标签，把它放在卡牌名称应该在的位置。最后我们在标签替换阶段把英文卡牌名插进中文翻译就可以了。

在 Serum Sovereign 的例子中，第二和第三个句子都包含卡牌名“Serum Sovereign”，所以它们都被标签 <0> 替换：

1. Flying

2. Whenever you cast a noncreature spell, put an oil counter on <0>.

3. {U}, Remove an oil counter from <0>: Draw a card, then scry 2.

5.1.3 标签插入

我们希望用一个专有名词字典解决专有名词翻译问题。字典包含一些英文专有名词和它们的中文标准译名，我们希望确保模型把每个在字典中出现的专有名词准确翻译成它的译名。

与名称检测步骤的做法类似，我们在标签插入步骤中用标签 <id> 代替输入中的所有专有名词。我们会训练翻译模型在翻译结果中保留标签，并把这些标签放到正确的位置。

继续考虑 Serum Sovereign 的例子。假设我们的专有名词字典包括一个词“oil”，它的译名是“烁油”，那么规则文字第二和第三个句子中的 oil 也要被替换成标签。由于 <0> 已经用过了，这次替换成 <1>：

1. Flying

2. Whenever you cast a noncreature spell, put an <1> counter on <0>.

3. {U}, Remove an <1> counter from <0>: Draw a card, then scry 2.

5.1.4 单句翻译

我们把已经插入标签的英文输入句子送入序列到序列翻译模型，生成包含标签的中文句子。我们在这一步尝试了多种序列到序列模型，包括完全基于 RNN 的 Encoder-Decoder 模型、Transformer 模型以及 Transformer Encoder 和 RNN Decoder 结合的模型。

继续 Serum Sovereign 的例子，把每个包含标签的英文句子翻译成包含标签的中文句子：

1. 飞行

2. 每当你施放非生物咒语时，在 <0> 上放置一个 <1> 指示物。

3. u，从 <0> 上移去一个 <1>：抓一张牌，然后占卜 2。

5.1.5 标签替换

这一步的输入是包含标签的中文句子。对于每个标签，如果它是在名称检测阶段引入的，则用对应的英文卡牌名替换此标签；如果它是在标签插入阶段引入的，那么它代表一个专有名词字典中的专有名词，我们用字典中的相应译名替换它。

对于 Serum Sovereign 的例子而言，我们需要把标签 <0> 替换成“Serum Sovereign”，把标签 <1> 替换成“烁油”：

1. 飞行

2. 每当你施放非生物咒语时，在 <serum sovereign> 上放置一个烁油指示物。

3. u，从 <serum sovereign> 上移去一个烁油：抓一张牌，然后占卜 2。

5.1.6 句子拼接

最后一步，把所有句子的中文翻译拼接起来，产生整段规则文字的翻译。

5.2 搭建数据集

5.2.1 原始数据获取

我们的原始数据从卡牌网站 scryfall.com 获取。该网站整理了万智牌所有系列、所有语言的卡牌数据。

5.2.2 数据整理与清洗

卡牌的原始数据中一条数据描述了一张单卡的一个发行版本。一张单卡除了卡牌名和规则文字外，还包括了使用语言，所在系列，卡牌编号，发行版本编号等等。我们先按照语言，将中文和英文版本的牌的数据写入不同文件，滤掉其他语言的牌。即使是同一张牌的同一个语言版本也会存在多个版本，因此我们需要进行数据去重。在此之后，我们将同一张牌的中英文版本绑定在一起，并删去那些无用的数据项。然后我们检查同一张牌的描述中英文句子数是否一致，以过滤掉那些不符合规范的牌。最终，我们将数据集划分为训练集、验证集和测试集三部分，使用 json 格式存储。每条数据包含一张卡片上的所有规则文字的中英文对照，测试集约 37000 条数据，验证集和测试集各有约 1000 条数据。我们将整理好的数据集开源到了 github 上。

5.2.3 数据增强

正如我们在翻译流程一节所描述的，我们的翻译系统在将英文文本输入序列到序列翻译模型之前会进行预处理，在文本中插入 <id> 标签。序列到序列翻译模型要学会在输出中保留输入的标签，并把它们摆放在恰当的位置。需要注意的是，为了正确处理输入输出中的标签，模型需要具备推断标签所代表的词的词性和类别的能力。考虑这样一个包含标签 <0> 和 <1> 的英文句子：Put a <0> on target <1>. 这句话应当被翻译为“在 <1> 上放置一个 <0>。”模型必须推断出标签 <1> 代表一个物体，标签 <0> 代表要放在那个物体上的东西，它才能学会在翻译过程中交换两个标签的位置。

为了训练模型处理标签的能力，我们需要进行数据增强，在训练数据中引入 <id> 标签。训练数据集中的标签分布应当涵盖实际翻译过程中的所有情形，也就是说，训练数据中的标签需要代表名词、动词、形容词等不同类别的词，这样才能教会模型处理各种类别的标签。

我们通过替换英文和中文句子中的对应词语来产生带标签的训练样例。比如说，英文关键字与中文译名严格对应，我们可以把英文句子中的关键字和中文句子中的译名同时替换为同一个标签。我们挑选了一部分常见关键字，通过把它们全部替换为标签产生一批新数据，然后把这些新数据加入数据集。

我们在研究过程中尝试了很多种数据增强方法，所以在原版数据集的基础上产生了很多增强版本。我们为所有数据集版本命名并进行了测试。下面是所有版本的名称和描述：

数据集版本	描述
plain	对齐的规则文字句子，没有额外处理。
v1	中文句子中的卡牌名被替换为标记 <cn>
v2	英文句子中的卡牌名被加上尖括号标记，中文句子中出现的卡牌名被替换为英文名
v2.1	一些中英文卡牌名被替换为 <id> 标签
v2.2	在 v2.1 的基础上做了更多替换，并且保留了未做替换的版本

所有这些版本都可以通过我们自定义的 python 类 RuleText 加载，如有兴趣请查看我们 github 仓库中的 dataset 文件夹。

5.3 名称检测模型

名称检测模型就是我们用于检测输入的英文句子中的卡牌名称的模型，它为输入的每个 token 进行二分类，判断它们是否属于卡牌名称的一部分。

与翻译模型一样，名称检测模型也是序列到序列模型，只不过它输出的是一个长度与输入相等的 01 序列，0 表示这个位置的 token 不是卡牌名的一部分，1 则表示它是卡牌名的一部分。

输入与输出长度相等的限制表明 Encoder-Decoder 架构不适合名称检测模型。每个位置的输出与该位置附近的信息密切相关。考虑到这些因素，最终我选择使用双向 RNN 实现名称检测模型。

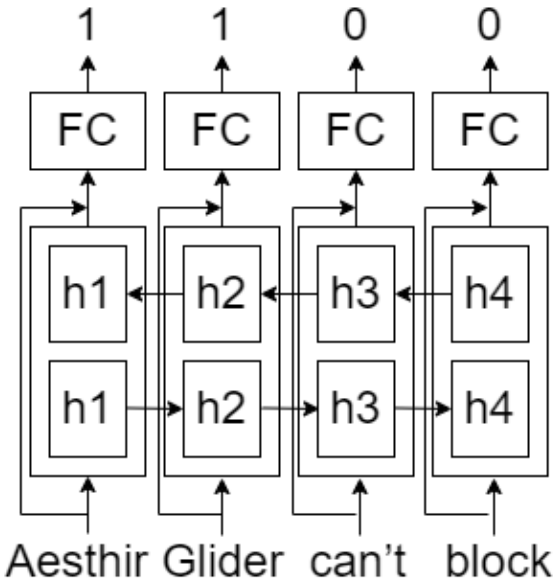

图 5 名称检测模型的架构.

我使用双向 RNN 提取特征，然后把词嵌入和 RNN 隐藏状态一起输入一个全连接层得到分类结果，如图所示。

实验表明基于 RNN 的名称检测模型准确率相当高，因此我们没有尝试其他可能的模型架构，比如基于 Transformer 的模型。我们的名称检测模型位于 github 仓库的 models.card_name_detector 目录下。

5.4 RNNsearch 模型

我们尝试了好几种序列到序列翻译模型，其中之一是 Bahdanau 等人提出的 RNNsearch 模型。

RNNsearch 模型的编码器与解码器都基于 RNN。但是，与传统的编码器只产生一个编码向量不同，RNNsearch 编码器对输入句子的每个词 x_i

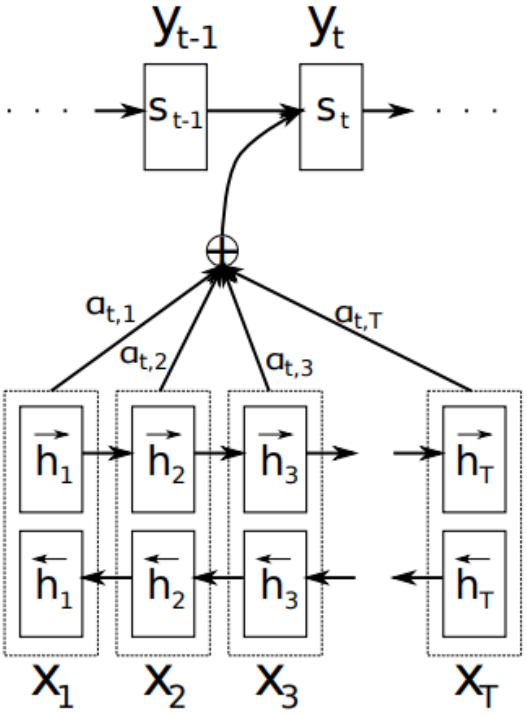

图 6 RNNsearch 模型的架构. 图片引自[1].

产生一个输出 h_i，称为注解（annotation）。每个注解都编码了整个输入序列，但是特别偏重于相应单词附近的信息。由于 RNN 在序列位置 i 的输出只与位置 i 之前的单词有关，为了使注解包含位置 i 之后的信息，编码器采用双向 RNN 的结构。双向 RNN 其实就是两个 RNN 组合起来，其中一个从前往后读入序列，另一个从后往前读入序列，然后把两个 RNN 在同一位置的输出拼接起来作为该位置的注解，解码器会使用这一系列注解来生成上下文向量。

5.5 transformer 模型

我们尝试的另一种序列到序列翻译模型是 Transformer 模型。

如图 4，Transformer 使用编码器和解码器的模式。我们实现的网络中编码器和解码器各为 3 层。编码器每个层包含两个子层，一层是多头自注意力机制，一层是逐位置的全连接前馈网络。这两个子层的连接需要使用残差连接和归一化。

解码器每个层包含三个子层。多头自注意力机制和逐位置的全连接前馈网络外，还有一层对 encoder 的输出执行注意力机制。同样的，这三层的连接也使用残差和归一化。

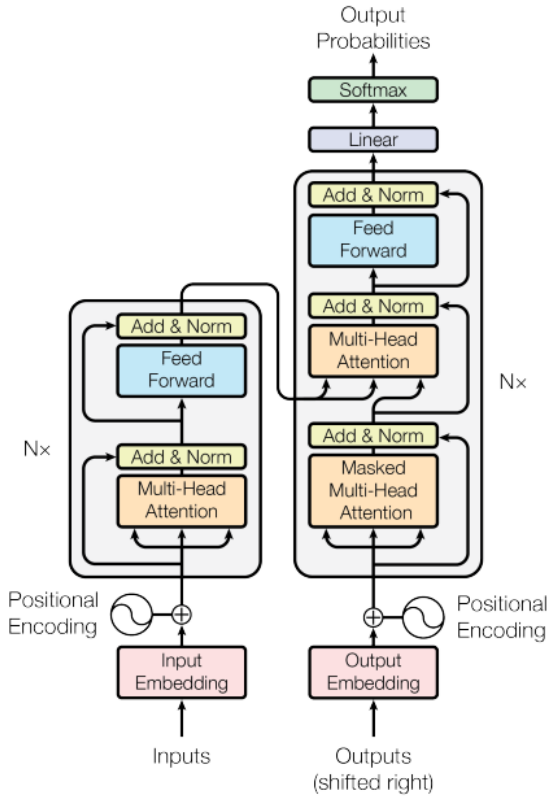

图 7 Transformer 模型的架构. 图片引自[2].

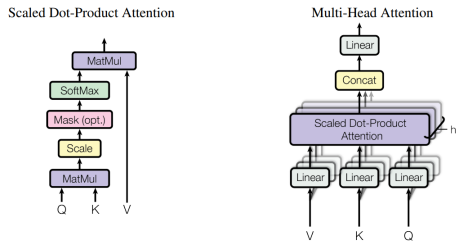

图 8 MultiHeadAttention. 图片引自[2].

5.5.1 多头自注意力

如图五，左侧的图是单头自注意力。query 和 key 做矩阵乘法后进行 scale。scale 的目的是防止高维矩阵乘法得到的结果过大。Mask 只用在 decoder 中，作用是防止注意力注意到这次输出之后的输出。softmax 进行归一化，得到注意力。将注意力作用于 value 上得到结果。

右侧的图为多层注意力，由多个单层注意力得到。实际上就是把多个单层注意力直接拼接在一起然后做线性变换。

5.5.2 位置编码

transformer 不同于 rnn，它在结构上并没有保留位置信息。因此，我们有必要在输入中添加位置信息的表示。我们尝试了两种方案，一种是使用正余弦波的固定位置编码方案，一种是可学习的位置编码方案。结果上看固定方案更好，我们将在实验结果部分展示这一数据。而且，由于固定的使用正余弦波的方案可以扩展到任意的输入长度上，因此我们认为正余弦波方案更好。

6 实验过程与结果

6.1 RNNsearch 与 v2.1 数据集

我们的第一个模型是在 v2.1 数据集上训练的 RNNsearch 模型，我们将它命名为 model4-v2.1。我们在测试过程中发现，虽然模型很好地学会了中文规则文字的语气和常用短语，但是它没能掌握标签的处理规则，经常出现丢弃标签或者把标签放置在错误位置的情况。

v2.1 数据集基本只通过替换中英文卡牌名字产生包含标签的样例，标签在训练数据中出现的位置相当固定，基本只在主语位置出现。因此，我们猜测模型可能没能学会一般的标签处理规则，遇见出现在宾语或定语位置的标签就不会处理了。

为了解决这个问题，我们新创建了 v2.2 数据集，在 v2.1 的基础上增加了标签出现位置的多样性，请参考数据增强一节。

6.2 RNNsearch 与 v2.2 数据集

接下来我们尝试在 v2.2 数据集上训练 RNNsearch 模型，我们将它命名为 model4-v2.2。这个模型处理标签的能力明显提升，能够把标签放在正确的位置，丢弃标签的错误也不再发生了。

我们举一个例子说明这一点。我们设置反抗军（rebel）为标签，然后测试模型。我们发现 v2.1 版本把标签弄丢了，但是 v2.2 版本则产生了正确的翻译。

1. 原文：When this Equipment enters the battlefield, create a 2/2 red Rebel creature token, then attach this to it.
2. 标准答案：当此武具进战场时，派出一个 2/2 红色，然后将它装备上去。
3. model4-v2.1 的答案：当此武具进战场时，派出一个 2/2 红色，然后将它装备上去。
4. model4-v2.2 的答案：当此武具进战场时，将一个 2/2 红色反抗军衍生生物放进战场，然后将它装备上去。

但是 model4-v2.2 还有一个与 model4-v2.1 共通的问题：它们有时会生成重复的词句，导致翻译质量降低。我们认为这个问题是源于模型本身的，不能通过改良数据集来解决。所以我们接下来尝试基于 Transformer 的模型。

下面是产生重复词句的一个例子：

1. 原文：U, Remove an oil counter from Serum Sovereign: Draw a card, then scry 2.
2. 标准答案：U，从浆液君王上移去一个烁油指示物：抓一张牌，然后占卜 2。
3. model4-v2.2 的答案：u，从 <serum sovereign> 上移去一个烁油上移去一个烁油：抓一张牌，然后占卜 2。

6.3 采用可学习位置编码的 Transformer 与 v2.2 数据集

我们在 v2.2 数据集上训练了 transformer 模型，将它命名为 model6-v2.2。

我们本以为基于 transformer 的模型一定会优于基于 RNN 的模型，但是它的表现有点出乎我们的预料。model6-v2.2 生成的翻译语言质量确实更高，很少犯语法错误，但是它经常犯语义错误，比如搞反主语宾语的顺序，看起来颠三倒四。下面这个例子就是这样：

1. 原文：Then if it has no oil counters on it, you lose the game.
2. 标准答案：然后如果其上没有烁油指示物，则你输掉这盘游戏。
3. model6-v2.2 的答案：然后如果烁油上没有指示物，则你输掉这盘游戏。

我们想到，transformer 本身的结构不具备捕捉序列特征的能力，所以它需要提供位置编码；这有可能意味着它在序列性质强的翻译任务上表现较差。我们想到了两种改进方法：

1. model6-v2.2 采用可学习的位置编码，或许我们的数据集较小，不足以学到高质量的位置编码，所以接下来可以尝试改用 Attention is all you need 论文中采用的固定位置编码
2. 我们可以尝试混合模型，Encoder 端仍然采用 transformer encoder，Decoder 端改用基于 RNN 的 Decoder，我们希望这样产生的模型可以同时继承 model6-v2.2 语言质量高的优点和 model4-v2.2 处理序列关系能力强的优点

6.4 固定位置编码的 Transformer

我们首先尝试在 v2.2 数据集上训练固定位置编码的 Transformer 模型，将它命名为 model6.1-v2.2。我们惊喜地发现这个模型取得了最好的翻译效果，不仅语言质量高、很少犯语法错误，而且与 model6-v2.2 相比，翻译搞反顺序、颠三倒四的情形也大大减少了。

测试结果表明，model6.1-v2.2 在测试集上的 bleu score 是所有模型中最高的，具体数据请参见实验结果一节。

6.5 Transformer Encoder 与 RNN Decoder 混合模型

最后我们尝试了基于 Transformer Encoder 和 RNN Decoder 的混合模型，将它命名为 hybrid-v2.2。比起 model6-v2.2，这个模型犯的顺序错误确实减少了，但是它产生的翻译结果又开始包含 model4 的常见问题，即生成重复词句，语言质量降低了。

测试集上的 bleu score 结果表明，混合模型仅仅略好于 model6-v2.2，显著劣于 model6.1-v2.2。

6.6 实验结果

下面是各个模型在测试集上的 bleu score。可以看到 bleu score 最高的模型是 model6.1-v2.2，其次是 model4-v2.2。

模型	解释	BLEU	模型参数
model4-v2.1	RNNsearch+2.1 版本数据集	64.96	11M
model4-v2.2	RNNsearch+2.2 版本数据集	68.77	11M
model6-v2.2	Transformer(可学习位置编码)+2.2 版本数据集	65.29	5.3M
model6.1-v2.2	Transformer（固定位置编码）+2.2 版本数据集	69.41	5.2M
hybrid-v2.2	Transformer Encoder（固定位置编码）+ RNN Decoder +2.2 版本数据集	66.73	12M

7 成员分工

我们团队包括两个人：杜岱玮（学号 2011421）和徐百朋（学号 2012109）。

杜岱玮同学负责准备数据集，预处理中的“分句”环节，翻译中的“RNN”模型以及最后的词典替换环节。

徐百朋同学负责预处理中的“分词”环节，翻译中的“Transformer”模型以及最后的句子组装环节。

参考文献

[1] BAHDANAU D, CHO K, BENGIO Y. Neural machine translation by jointly learning to align and translate[J]. arXiv preprint arXiv:1409.0473, 2014.

[2] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is all you need [J]. Advances in neural information processing systems, 2017, 30.

[3] STAHLBERG F. Neural machine translation: A review[J]. Journal of Artificial Intelligence Research, 2020, 69:343-418.

[4] SUTSKEVER I, VINYALS O, LE Q V. Sequence to sequence learning with neural networks[J]. Advances in neural information processing systems, 2014, 27.

[5] WU Y, SCHUSTER M, CHEN Z, et al. Google's neural machine translation system: Bridging the gap between human and machine translation [J]. arXiv preprint arXiv:1609.08144, 2016.

[6] SCRYFALL. All cards[EB/OL]. https://scryfall.com/docs/api/bulk-data.

[7] Mtg wiki[EB/OL]. https://mtg.fandom.com/wiki/.

8 附录

五种模型翻译对比：

原文	Flying Archfiend of the Dross enters the battlefield with four oil counters on it. At the beginning of your upkeep, remove an oil counter from Archfiend of the Dross. Then if it has no oil counters on it, you lose the game. Whenever a creature an opponent controls dies, its controller loses 2 life.
标准翻译	飞行 深沼翼邪鬼进战场时上面有四个烁油指示物。 在你的维持开始时，从深沼翼邪鬼上移去一个烁油指示物。 然后如果其上没有烁油指示物，则你输掉这盘游戏。 每当一个由对手操控的生物死去时，其操控者失去 2 点生命。
model4-v2.1	飞行 <archfiend of the dross> 进战场时上面有四个充电指示物。 在你的维持开始时，从 <archfiend of the dross> 上放置一个烁油指示物。 然后如果其上有烁油上，则你输掉这盘游戏。 每当一个由对手操控的生物死去时，其操控者失去 2 点生命。
model4-v2.2	飞行 <archfiend of the dross> 进战场时上面有四个烁油指示物。 在你的维持开始时，从 <archfiend of the dross> 上移去一个烁油指示物。 然后若其上没有烁油指示物，则你输掉这盘游戏。 每当一个由对手操控的生物死去时，其操控者失去 2 点生命。
model6-v2.2	飞行 <archfiend of the dross> 进战场时上面有四个烁油指示物。 在你的维持开始时，从 <archfiendfiend of the dross> 上移去一个指示物。 然后如果烁油上没有指示物，则你输掉这盘游戏。 每当一个由对手操控的生物死去时，其操控者失去 2 点生命。
model6.1-v2.2	飞行 <archfiend of the dross> 进战场时上面有四个烁油指示物。 在你的维持开始时，从 <archfiend of the dross> 上移去一个烁油指示物。 然后如果其上没有烁油指示物，则你输掉这盘游戏。 每当一个由对手操控的生物死去时，其操控者失去 2 点生命。
hybrid-v2.2	飞行 <archfiend of the dross> 进场时上面有四个烁油。 在你的维持开始时，从 <archfiend of the dross> 上移去一个烁油上移去一个指示物。 然后如果其没有没有指示物指示物，则你输掉这盘游戏。 每当一个由对手操控的生物死去时，其操控者失去 2 点生命。