文章理解

- 引用透明性：输出的结果只由输入决定。在编程语言领域，引用透明性是指在不改变程序结果的情况下，一个表达式可以被其在程序中的相应值所替换的能力。例如，数学函数（如平方根函数）是引用透明的，而打印时间戳的函数则不是。平方根只与输入有关。
- RTI：定义为在不同上下文中具有固定翻译的文本。
- RTI 定义为跨文本（句子和短语）翻译不变的一段文本。
- 关键是生成包含相同的RTI的一堆文本，并检查这一对文本的翻译是否不变。作者实现了Purity，s一个从人以为标记的句子中提取短语的工具。
- Purity实现原理：给定一个源句,Purity通过constituency parser提取短语，并通过将RTI与其包含的句子或短语分组来构造RTI对。

3.1 Identifying RTIs

- 收集RTIs列表，即找到具有独特意义的文本片段，这个意义应该在不同的上下文都适用（相同的翻译场景）。本文只考虑名词短语（从源语言的一组句子中提取名词短语）。
- 使用NLP工具 constituency parser 输出字符串的句法树结构，非叶节点是成分结构关系，叶节点是单词。成分结构关系（将一个句子解析为一系列成分结构，即对句子进行层级结构分析。如主语+谓语，名词短语+动词短语，不断解析直到不能分出更小的成分为止）
- 通常一个RTI可以包含另一个较短的RTI。从一个句子中获得所有的名词短语后，根据经验过滤掉包含十个以上和三个以下的非停止词的词.

3.2 Generating Pairs in Source Language

- 生成的 RTIs 列表必须与包含短语配对，这些短语将用于引用透明性验证。具体来说，每个 RTI 对应该有两个包含相同短语的不同文本片段。将 RTI 与发现它的全文和同一个句子所有包含它的 RTI 配对，构建3对 RTI：(1) RTI1 和原句、(2) RTI2 和原句、(3) RTI1 和 RTI2。

3.3 Collecting Pairs in Target Language

- 一旦得到一组RTI对，下一步是将这些文本（以给定的源语言）输入到被测机器翻译软件中，并收集它们的翻译（以任何选定的目标语言）。使用谷歌和 Bing 提供的 api，这些 api 返回的结果与二者的 Web 接口返回的内容相同。

3.4 Detecting Translation Errors

- NLP的单词对齐技术，可以将源文本中的单词/短语映射到目标文本中的单词/短语，然而现有工具的性能很差。所以采用词袋BoW模型。
- BoW词袋模型：最早出现在自然语言处理和信息检索邻域。该模型忽略掉文本的语法和语序等要素，将其仅仅看作是若干个词汇的集合，文档中每个单词的出现都是独立的。BoW使用一组无序的单词来表达一段文字或一个文档。exp：一句话由n个单词组成，分别计算每个单词出现的频率，将其构建成为向量，向量维数由单词个数构成。
    - 计算两个词袋之间的距离,在实现过程中，采用了不同的计算方式，但最终效果与原实现方式差异不大。

4 Evaluation

- Results： 如果想要更加精确，可以使用更大的距离阈值，例如当距离阈值为5的时候，Purity在所有实验设置中都达到了100%的准确率，注意，准确率不会阈值的增加而单调增加，当阈值从2改为3时，准确率下降了1.9%，这是因为虽然假正例的数量减少了，但真正例的数量也减少了。
- 在实验中发现了五种翻译的错误：欠翻译、过度翻译、单词/短语误译、修饰错误和逻辑不清。

5 RTI for Robust Machine Translation

- 与传统软件相比，机器翻译软件的bug修复过程更加困难，因为NMT模型的逻辑包含在一个复杂的模型结构及其参数值中。即使可以识别出误译的计算之处，但往往不清楚如何改变模型以纠正错误且不引入新错误。对于在线翻译系统，修复误译的最快方法是对翻译进行硬编码。因此RTI发现的翻译错误可以作为早期预警，帮助开发人员避免可能导致负面影响的关键错误，然而这并不能够解决神经网络本身犯下的错误

6 conclusion

作者提出了一个简单、通用的概念——引用透明输入RTI——用于测试机器翻译软件，不同于现有的——扰乱一个自然句中的词（上下文固定），并假设翻译应该只有小的变化——的方法。本文假设RTIs应该在不同的上下文中具有不变的翻译，因此RTI可以发现不同种类的翻译错误，从而补充现有的方法。

7 算法伪代码

Algorithm 1 RTI implemented as *Purity*.

Require: *source_sents*: a list of sentences in source language
d: the distance threshold
Ensure: *suspicious_issues*: a list of suspicious pairs

```
suspicious_issues ← List( )                    ▷ Initialize with empty list
for all source_sent in source_sents do
    constituency_tree ← PARSE(source_sent)
    head ← constituency_tree.head( )
    RTI_source_pairs ← List( )
    RECURSIVENPFINDER(head, List( ), RTI_source_pairs)
    RTI_target_pairs ← TRANSLATE(RTI_source_pairs)
    for all target_pair in RTI_target_pairs do
        if DISTANCE(target_pair) > d then
            Add source_pair, target_pair to suspicious_issues
return suspicious_issues

function RECURSIVENPFINDER(node, rtis, all_pairs)
    if node is leaf then
        return
    if node.constituent is NP then
        phrase ← node.string
        for all container_phrase in rtis do
            Add container_phrase, phrase to all_pairs
        Add phrase to rtis
    for all child in node.children( ) do
        RECURSIVENPFINDER(child, rtis.copy( ), all_pairs)
    return all_pairs

function DISTANCE(target_pair)
    rti_BOW ← BAGOFWORDS(target_pair[0])
    container_BOW ← BAGOFWORDS(target_pair[1])
    return |rti_BOW \ container_BOW|
```