机器翻译_基于 transformer

一．概述

本次实验完成了英文到中文的翻译任务，基于 transformer 结构实现。

二．方法论

Transformer 使用注意力机制能够很好地完成 seq2seq 任务。

三．实验细节

（1） 数据预处理

经过对于数据集的观察，我发现有一些小问题需要进行面向任务的数据清洗。

- 对于中文数据：

首先需要去除句子中的空格，这些空格是不必要的；其次中文句子标点符号有些问题，有的句子结尾有标点有的没有，于是直接对文末没有标点的句子增加句号；再次对中文句子进行分词，直接使用 jieba 库进行分词即可。

- 对英文数据：

首先考虑到英文句子首字母都会大写，这就导致了模型可能需要保存两个完全一样的词一个首字母大写一个小写，所以我们需要将英文句子首字母转小写（考虑到句中可能有专有名词需要大写，所以没有对整个句子做小写处理）；其次英文句子可以直接用空格来分词，但是英文的标点符号一般和前一个词之间没有空格，所以我们需要预处理将英文标点前增加空格。

经过以上中文英文预处理步骤，得到了结构如下所示数据。

```
print(source_tokens[8])
print(target_tokens[8])
```

```
['we', 'piled', 'enormous', 'amounts', 'of', 'pollution', 'into', 'the', 'air', 'and', 'into', 'the', 'water', '.']
['我们', '堆积', '了', '大量', '的', '污染物', '在', '空气', '中', '和', '水中', '。']
```

- 处理超长句子：

数据集中有一些超长的句子甚至会超过一千个词，这将导致后续进行所有句子 padding 长度时，整个训练数据大小会成倍增加，加重模型负担。因此在不严重影响数据集分布的情况下，限定中文句子长度最长为 80 词，并删除超过 80 词的中文和英文样本。操作结束后训练集样本数由 143920 降低到 125445。

- 面向 transformer 数据预处理：

首先需要对数据中出现的词构建词典，这里为了减小模型的复杂度，将整个词典大小设置为 32000 词，对于出现次数排在 32000 以后的词直接丢弃（观察到后面的词其实只出现了一次），词典映射的时候如果出现集外词就用<OOV>表示；其次在每条句子的开头和结尾分别增加<START>和<END>的标志位；再次将每条句子 padding 成最长的句子长度；最后对样本句子进行词典映射。如上操作结束后，上文的中文英文句子转化成如下图。其中 0,1,2,3 分别表示<PAD>、<START>、<END>和<OOV>。

```
print(encode_input[8])
print(decode_input[8])
```

```
[1, 15, 18843, 1427, 2714, 6, 5256, 84, 4, 823, 8, 84, 4, 329, 156, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 9, 5676, 11, 510, 5, 5881, 10, 1072, 38, 18, 2794, 4, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

（2） 模型结构

Transformer 分为 encoder 和 decoder。由于我没有使用预训练好的词向量，所以需要增加嵌入层将词典下标转化为词向量，并且词向量是随着任务一起训练的，对于 encoder 端和 decoder 端的嵌入层神经元数均设置为 120 维；encoder 和 decoder 堆叠数量均设

置为 4（比标准 transformer 小）；多头自注意力机制头数设置为 4；前向网络神经元数量设置为 128 个；为了防止过拟合并更好的训练，dropout 设置为 0.05；注意力机制后的激活函数使用 relu 函数；优化器使用 adam；损失函数使用交叉熵函数；度量标准初步使用的精确率；模型共有 8662468 个参数 batch 大小设置为 8，在一张 NVIDIA TITAN XP 上训练了 30 轮，并保存了 15 轮和 30 轮的模型。

前后 15 轮训练过程如下

```
15681/15681 [==============================] - 885s 56ms/step - loss: 2.2580 - accuracy: 0.2791
Epoch 2/15
15681/15681 [==============================] - 888s 57ms/step - loss: 1.9781 - accuracy: 0.3410
Epoch 3/15
15681/15681 [==============================] - 885s 56ms/step - loss: 1.8314 - accuracy: 0.3678
Epoch 4/15
15681/15681 [==============================] - 887s 57ms/step - loss: 1.7306 - accuracy: 0.3855
Epoch 5/15
15681/15681 [==============================] - 887s 57ms/step - loss: 1.6567 - accuracy: 0.3979
Epoch 6/15
15681/15681 [==============================] - 885s 56ms/step - loss: 1.6005 - accuracy: 0.4075
Epoch 7/15
15681/15681 [==============================] - 887s 57ms/step - loss: 1.5551 - accuracy: 0.4158
Epoch 8/15
15681/15681 [==============================] - 884s 56ms/step - loss: 1.5182 - accuracy: 0.4225
Epoch 9/15
15681/15681 [==============================] - 886s 57ms/step - loss: 1.4871 - accuracy: 0.4284
Epoch 10/15
15681/15681 [==============================] - 885s 56ms/step - loss: 1.4608 - accuracy: 0.4333
Epoch 11/15
15681/15681 [==============================] - 887s 57ms/step - loss: 1.4376 - accuracy: 0.4379
Epoch 12/15
15681/15681 [==============================] - 885s 56ms/step - loss: 1.4166 - accuracy: 0.4418
Epoch 13/15
15681/15681 [==============================] - 888s 57ms/step - loss: 1.3995 - accuracy: 0.4450
Epoch 14/15
15681/15681 [==============================] - 890s 57ms/step - loss: 1.3835 - accuracy: 0.4484
Epoch 15/15
15681/15681 [==============================] - 888s 57ms/step - loss: 1.3684 - accuracy: 0.4515
```

```
Epoch 1/15
15681/15681 [==============================] - 889s 57ms/step - loss: 1.3550 - accuracy: 0.4541
Epoch 2/15
15681/15681 [==============================] - 891s 57ms/step - loss: 1.3432 - accuracy: 0.4565
Epoch 3/15
15681/15681 [==============================] - 889s 57ms/step - loss: 1.3322 - accuracy: 0.4590
Epoch 4/15
15681/15681 [==============================] - 888s 57ms/step - loss: 1.3219 - accuracy: 0.4613
Epoch 5/15
15681/15681 [==============================] - 889s 57ms/step - loss: 1.3123 - accuracy: 0.4631
Epoch 6/15
15681/15681 [==============================] - 892s 57ms/step - loss: 1.3037 - accuracy: 0.4649
Epoch 7/15
15681/15681 [==============================] - 891s 57ms/step - loss: 1.2955 - accuracy: 0.4669
Epoch 8/15
15681/15681 [==============================] - 890s 57ms/step - loss: 1.2873 - accuracy: 0.4684
Epoch 9/15
15681/15681 [==============================] - 893s 57ms/step - loss: 1.2792 - accuracy: 0.4701
Epoch 10/15
15681/15681 [==============================] - 892s 57ms/step - loss: 1.2720 - accuracy: 0.4720
Epoch 11/15
15681/15681 [==============================] - 890s 57ms/step - loss: 1.2646 - accuracy: 0.4737
Epoch 12/15
15681/15681 [==============================] - 889s 57ms/step - loss: 1.2575 - accuracy: 0.4749
Epoch 13/15
15681/15681 [==============================] - 887s 57ms/step - loss: 1.2518 - accuracy: 0.4765
Epoch 14/15
15681/15681 [==============================] - 888s 57ms/step - loss: 1.2464 - accuracy: 0.4775
Epoch 15/15
15681/15681 [==============================] - 886s 56ms/step - loss: 1.2404 - accuracy: 0.4795
```

（3） 解码策略

解码策略使用贪心策略选择下一个词。

四．结果

经过 30 轮的训练，模型已经能够翻译一些简单的句子了，如下图（输入英文，下一行是模型给出的中文翻译结果）

```
Input english sentence:         good morning
早上好。
Input english sentence:         I want to buy a computer.
我想买一台电脑。
Input english sentence:         China is in the east of Asia
在亚洲的东部是亚洲。
Input english sentence:         December is winter
<OOV>:是冬天。
Input english sentence:         I read a very interesting book today
今天我看到了一个非常有趣的书。

Input english sentence:
```

使用 bleu 值衡量模型性能，达到了 17.86（nltk 的 bleu-1）。