浙江大学

本科实验报告

课程名称：自然语言处理

姓　　名：张溢弛

学　　院：计算机科学与技术学院

专　　业：软件工程 1801

学　　号：3180103772

指导教师：汤斯亮

2021 年 6 月 17 日

浙江大学 实验报告

课程名称：自然语言处理　　实验类型：综合型

实验项目名称：实验四：基于 Seq2Seq 和 Transformer 的机器翻译

学生姓名：张溢弛　专业：软件工程 1801　学号：3180103772

同组学生姓名：独立完成　指导老师：汤斯亮

实验地点：玉泉一舍 376　实验日期：2021 年 6 月 17 日

目录

一　项目介绍

1.1　实验选题

基于 Seq2Seq 和 Transformer 模型的机器翻译实验

1.2　实验内容

本实验使用 Seq2Seq 架构和 Transformer 架构分别在同一个简单的中英文语句数据集上进行了

1.3　实验环境

- 操作系统：华为云 ModelArts 云平台
- 编程环境：Python3+Jupyter Notebook
- 使用的 Python 库：mindspore1.1.1

二　技术细节

2.1　技术原理

在正式开始实验之前，我先按照上课 PPT 和 CS224N 课程的相关资料进行了学习，下面放一些我在学习过程中记录的笔记。

2.1.1　Seq2Seq 模型

机器翻译任务是将句子从一种语言翻译成另一种语言，1990 年代到二十一世纪初的机器翻译主要运用的还是基于统计学习的机器翻译方法 (SMT，Statical Machine Translation)，使用贝叶斯方法来构建模型，需要用大量的数据分别训练翻译模型和语言模型，为了将大量数据用于模型的并行训练，需要引入一个对齐变量，使得原文和译文要是对应的，否则可能导致翻译结果的不准确。alignment 是一种隐变量，没有显式地出现在数据中，因此**可以使用 EM 算法来进行优化**。

早期的翻译模型主要依赖于统计和概率的传统方法，这种模型分成两个部分，分别是一个翻译模型和一个语言模型，翻译模型用来预测文本中的一个句子最有可能翻译成什么，而语言模型来判断当前的句子是否对于全文而言是最合适的。

这种系统的构建主要基于单词或者短语，但是基于单词的翻译系统难以捕捉到语句中单词位置导致的差异，比如否定词的位置，句子中主语和动词的关系，而基于短语的翻译系

统可以看成是 Seq2Seq 的前身，这种模型将输入和输出都看成是一系列短语并且可以构建更复杂的语法，但是这种模型下，长依赖关系依然难以提取。

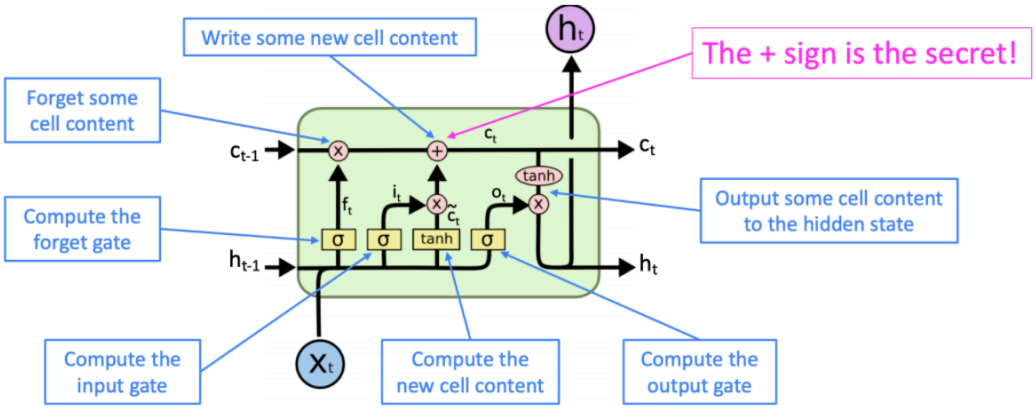
Sequence-to-sequence(简称为 Seq2Seq，中文翻译是序列到序列) 是一种相对比较新的**端到端的翻译模型**，Seq2Seq 使用两个 RNN 模型构成的，分别是：

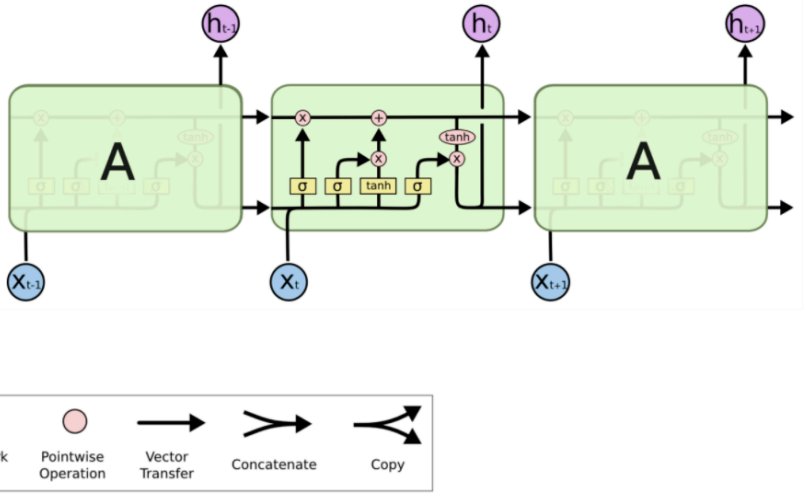
- 编码器：将输入序列编码成一个上下文向量 (context vector)
- 解码器：根据编码器生成的上下文向量生成对应的输出序列

因此 Seq2Seq 也被称为“编码器-解码器”模型。无论是编码器还是解码器，都需要先用一个嵌入层将输入的单词序列转化成词向量作为编码器和解码器的输入。

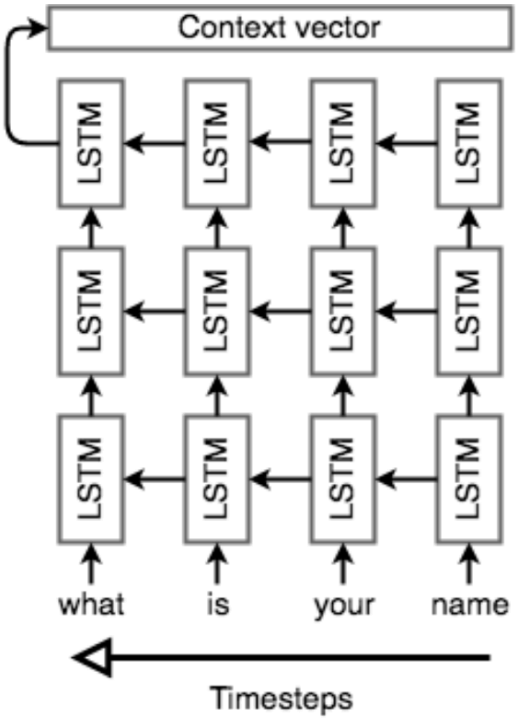
Seq2Seq 中的编码器用于将输入的单词序列编码成一个定长的上下文向量，因此编码器的组成单元通常是用 LSTM 的 RNN，这样的解码器在提取长依赖关系的时候效果更好，最终的隐藏状态就是 C，但将任意长度的输入压缩到一个固定长度的向量中是比较困难的，并且解码器往往有若干个 LSTM 层，称为 LSTM 栈 (stacked LSTM)，是一层接一层的 LSTM，也可以叫做 LSTM 网络，每一层的输入是上一层的输出，最上层是隐藏状态层，**输出的结果就是定长的向量上下文向量 C**

Seq2Seq 会对输入的训练进行反向的处理，也就是从序列的末尾从后向前处理，这样一来编码器读取到的最后一个单词将会对应输出结果中的第一个单词，这样可以方便解码器更快生成准确度更高的前面一部分译文，而译文最前面的一小部分的正确性对于译文整体的正确性的影响是非常大的，下面的图可以用来表示解码器的工作过程：





Seq2Seq 的解码器也是一个 LSTM 网络，当时相比于编码器的网络架构更复杂，首先要明确，**解码器的输入不是编码器中生成的上下文向量 C**，解码器的输入依然是一系列由序列转化过来的单词向量，并且在训练和测试的时候的内容是不同的，而上下文向量 C 是一个用于预测结果的参数，而不是解码器的输入。



- 在训练阶段，解码器中输入的是**真实的目标文本**，并且第一步使用一个特殊的 <start> 标记表明这是句子的开头，每一步根据当前正确的输出词，上一步的隐状态来预测下一步的输出
- 在训练的时候，因为没有“参考答案”了，所以只能把上一步的输出作为下一步的输入，一步步慢慢预测出最终结果

这样两种训练方式，分别称为 teacher forcing 和 free running，事实上训练阶段也可以 free running，但是这样训练出来的模型的 performance 非常糟糕，容易出现误差爆炸的问题，使用“参考答案”(实际上也可以看成是当前输入序列的一个标签) 可以提高准确度。同时训练的时候可以采用更好的办法，也就是计划采样 (Scheduled Sampling)，

当我们获得了一个输出序列的时候，我们可以使用同样的学习策略，定义损失函数 (比如使用交叉熵)，然后用梯度下降的算法来反向传播优化参数，编码器和解码器是同时训练的，因此它们会学习到同样的上下文向量表示。

2.1.2 注意力机制

事实上当我们听到或者读写一个句子的时候，我们并不是平等的看待每一个单词，而是会把注意力集中在一些单词上，同样的，Seq2Seq 模型中，不同的部分输入也可以有不同级别的重要性，而不同部分的输入也可以用不一样的重要程度来看待不同部分的输入，比如在翻译问题中，输出的第一个单词往往是基于前几个输入单词，但是最后一个输出的单词往往取决于最后几个输入的单词。

注意力机制 (Attention Mechanisms) 可以给解码器网络在每个解码步骤中**提供查看整个输入序列的机会**，然后解码器可以决定哪部分输入单词是重要的，使得解码器可以了解到的原文信息不再局限于一个上下文向量。

我们假设输入的序列是 $(x_1, x_2, \ldots, x_n)$，目标结果 (也就是正确答案) 是 $(y_1, y_2, \ldots, y_m)$，这样一来，实现注意力机制需要如下几个步骤：

编码器 假设隐层生成的隐状态向量是 $(h_1, h_2, \ldots, h_n)$，其中 h_t 表示解码器在第 t 个编码步骤中的隐藏状态，按照传统的 Seq2Seq，最后输出的上下文向量实际上就是 h_n，并且编码器使用一个双向的 LSTM 并且提取了输入的每一个单词的上下文特征并用向量表示。

解码器 在解码器中，计算下一个隐藏状态 s_i 的过程可以表示为：

$$s_i = f(s_{i-1}, y_i, c_i) \tag{1}$$

其中 c_i 表示一个**上下文向量**，这个向量提取了第 i 步解码过程中的相关信息 (经典的 Seq2Seq 中只有一个上下文向量，而注意力机制中有若干个, 实际上 Seq2Seq 模型中的**注意力机制就是为解码器中每个特定的位置，使用解码器中生成的所有隐状态向量生成了一个特定的上下文向量**)，我们需要计算的就是这个 c_i，首先需要对于每一个隐藏向量，计算一个″ 分数″：

$$e_{i,j} = a(s_{i-1}, h_j) \tag{2}$$

这样一来就得到了一个分数序列，然后将其通过 softmax 函数进行标准化：

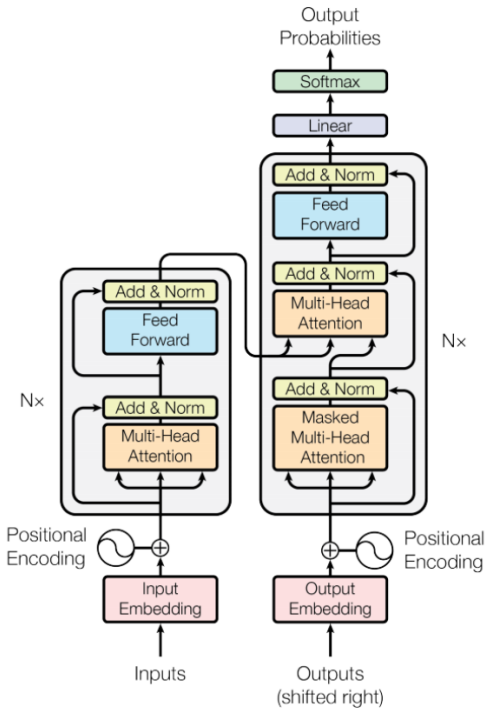
$$\alpha_{i,j} = \frac{\exp(e_{i,j})}{\sum_{k=1}^{n} \exp(e_{i,k})} \tag{3}$$

这里计算出的向量 α_i 被称为是注意力向量，然后可以计算新的上下文向量 c_i，用隐藏向量的加权和来表示：

$$c_i = \sum_{j=1}^{n} \alpha_{i,j} h_j \tag{4}$$

2.1.3 Transformer 模型

Transformer 中沿用了非常经典的编码器和解码器架构，解码器将输入的序列转化成一个表示向量，而编码器将向量转化成对应的输出结果，并且在每个步骤中模型都是自回归的，也就是使用前面生成的符号作为输入来生成后面的内容 (这和序列的特性也有关系，序列后面的内容往往和序列前面的有一定的联系)，同时 Transformer 模型使用了层级化的自注意力机制和 point-wise 的全连接层，其大致的架构如下图所示：



编码器 Transformer 模型中的编码器采取了层级化架构，一共使用了 N=6 个编码器，每个编码器包含两个子层，分别是多头注意力层和一个全连接层的前馈网络，同时每一个编码器的每一个子层中使用了残差连接和标准化，也就是说假设每一层的输入是 x，那么每一层的输出结果可以表示为：

$$\text{LayerNorm}(x + \text{SubLayer}(x)) \tag{5}$$

解码器 Transformer 中的解码器也采用了层级化的架构，一共使用了 N=6 个解码器，并且每个解码器包含三个子层，其中的新增的多头注意力层会对编码器层输入解码器的表示向量进行注意力的计算，而原本的多头注意力层需要加上 mask，因为在预测生成结果的过程中，不能让前面的东西感知到后面 (还没生成) 的东西的存在，这就是 mask 的作用，这样一来解码器层的注意力只能集中在已知的内容中。

按比例的点积注意力机制 Scaled Dot-product Attention Transformer 模型中使用的是按比例缩小的点积注意力机制，注意力计算需要维度 d_k 的 query 和 key 以及维度 d_v 的 value，然后在传统的注意力机制运算方式基础上加上一个规模参数 $\sqrt{d_k}$ ，我们用向量化的方式可以将这种注意力机制的计算过程表示成：

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{6}$$

多头注意力机制 Multi-head Attention Transformer 采用了多头注意力机制，这种机制将 query，key 和 value 都进行 h 次投影，然后对 h 个投影进行并行的注意力计算，最后将 h 个投影组合之后进行线性投影生成最后的多头注意力机制。多头注意力机制允许 Transformer 在不同的表示空间中得到不同的注意力学习到不同的信息。多头注意力机制最终的结果可表示成：

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_\text{h})W^O \tag{7}$$

而每一个 head 中的注意力的计算方式可以表示成：

$$\text{head}_\text{i} = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right) \tag{8}$$

位置编码 尽管 Transformer 模型中没有循环和卷积单元，为了充分利用序列的信息，我们必须在嵌入向量中注入一些和位置有关的信息，因此本论文提出了一种位置编码的 idea，通过在嵌入向量中加入位置编码的方式来保留一些位置特征，在 Transformer 中的位置编码使用了 sin 函数和 cos 函数，即：

$$PE_{(pos,2i)} = \sin\left(pos/10000^{2i/d_\text{model}}\right) \tag{9}$$

$$PE_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_\text{model}}\right) \tag{10}$$

2.2 实验细节与关键步骤

本次实验的细节和关键的代码主要分为如下几个部分，其中实验结果将在第三部分具体给出。

2.2.1 实验环境配置

注册华为云平台的账号，由于代金券还没有到账因此自费充值来使用 ModelArts 平台，按照教程中创建好 OBS 桶并在 ModelArts 中配置好实验环境之后开始实验，其中数据集需要通过 moxing 库从 OBS 桶中复制到云平台当前的工作目录下。本次实验使用了新版实验手册，因此实验总体来说进行的比较顺利。

2.2.2 代码文件结构

本次实验的代码分为 Seq2Seq 和 Transformer 两个部分，其中 Seq2Seq 部分的代码仅有一个 notebook 组成，而 Transformer 模型包含了若干个代码文件。

beam_search.py	2 months ago
data_utils.py	2 months ago
lr_schedule.py	2 months ago
tokenization.py	2 months ago
train_util.py	2 months ago
transformer_for_train.py	2 months ago
transformer_model.py	2 months ago
weight_init.py	2 months ago

2.2.3 模型结构

Seq2Seq Seq2Seq 模型的代码中编写了一个 GRU(门控循环单元) 类作为最基本的组件，并编写了 Encoder 类和 Decoder 类作为模型的核心组成部分，Encoder 主要包含一个 Embedding 层和一个 GRU 层，Decoder 中有一个 Embedding 层，softmax 层，注意力层和 GRU 层组成，将 Encoder 和 Decoder 进行组合并使用 Encoder 的输出结果作为 Decoder 的输入，就可以得到了 Seq2Seq 模型，同时文件中还定义了 NLLLoss 用来计算损失函数。

Transformer Transformer 模型的代码中具体实现了基于多头注意力机制的 Transformer 模型，并为其配置了训练参数，data_utils 实现了数据的读入和预处理，weight_init 实现了权重参数的初始化，lr_schedule 实现了自适应学习率的生成，beam_search 实现了语句生成过程中的搜索算法。

2.2.4 数据集

Seq2Seq 和 Transformer 两个实验中都使用了一个简单的中英文翻译数据集，数据集中包含数万条中英文语句，Transformer 的数据集中还对句子预先做了分割，并建立了一个词汇表。

2.2.5 数据预处理

Seq2Seq 实验中的数据预处理需要先对句子进行分割，获取词汇表，并在得到单词和 id 的映射关系，然后将一个句子用单词的 id 来表示，并进行扩增统一长度，然后将处理之后的数据保存为 mindsporerecord 格式的数据就可以了。

```
if not os.path.exists("./preprocess"):
    os.mkdir('./preprocess')
convert_to_mindrecord("./data/cmn_zhsim.txt", './preprocess', MAX_SEQ_LEN)
```

```
[1, 86, 644, 0, 0, 0, 0, 0, 0, 0]
[1, 86, 644, 0, 0, 0, 0, 0, 0, 0]
[1, 1108, 644, 0, 0, 0, 0, 0, 0, 0]
[1, 512, 517, 0, 0, 0, 0, 0, 0, 0]
[1, 89, 517, 0, 0, 0, 0, 0, 0, 0]
[1, 239, 509, 644, 0, 0, 0, 0, 0, 0]
[1, 239, 508, 517, 0, 0, 0, 0, 0, 0]
[1, 282, 1148, 517, 0, 0, 0, 0, 0, 0]
[1, 965, 517, 0, 0, 0, 0, 0, 0, 0]
[1, 520, 211, 644, 0, 0, 0, 0, 0, 0]
en_vocab_size:  1154
ch_vocab_size:  1116
```

```
(1154, 1116)
```

而 Transformer 实验的数据预处理过程也类似，不过区别在于需要给输入的语料进行 mask 操作，防止生成过程中注意力作用在未知的单词中。

6. 数据处理，随机选20%作为测试数据

```
sample_num = 23607
eval_idx = np.random.choice(sample_num, int(sample_num*0.2), replace=False)
data_prepare(data_cfg, eval_idx)
```

```
finish 23607/23607
```

2.2.6 模型的训练

在设置好训练参数之后就可以进行模型的训练，Seq2Seq 模型的训练过程如下：

```
loss_cb = LossMonitor()
config_ck = CheckpointConfig(save_checkpoint_steps=cfg.save_checkpoint_steps, keep_checkpoint_max=cfg.keep_checkpoint_max)
ckpoint_cb = ModelCheckpoint(prefix="gru", directory=cfg.ckpt_save_path, config=config_ck)
time_cb = TimeMonitor(data_size=ds_train.get_dataset_size())
callbacks = [time_cb, ckpoint_cb, loss_cb]

model.train(cfg.num_epochs, ds_train, callbacks=callbacks, dataset_sink_mode=True)
```

```
epoch: 1 step: 125, loss is 2.6813457
epoch time: 77515.006 ms, per step time: 620.120 ms
epoch: 2 step: 125, loss is 2.4996283
epoch time: 11283.411 ms, per step time: 90.267 ms
epoch: 3 step: 125, loss is 1.8586909
epoch time: 11285.403 ms, per step time: 90.283 ms
epoch: 4 step: 125, loss is 1.7388054
epoch time: 11278.686 ms, per step time: 90.229 ms
epoch: 5 step: 125, loss is 1.245958
epoch time: 11318.234 ms, per step time: 90.546 ms
epoch: 6 step: 125, loss is 1.1044618
epoch time: 11295.832 ms, per step time: 90.367 ms
epoch: 7 step: 125, loss is 1.022195
epoch time: 11270.235 ms, per step time: 90.162 ms
epoch: 8 step: 125, loss is 0.68615824
epoch time: 11274.050 ms, per step time: 90.192 ms
epoch: 9 step: 125, loss is 0.4144318
epoch time: 11505.521 ms, per step time: 92.044 ms
epoch: 10 step: 125, loss is 0.33371535
epoch time: 11529.052 ms, per step time: 92.232 ms
epoch: 11 step: 125, loss is 0.17097613
epoch time: 11362.408 ms, per step time: 90.899 ms
epoch: 12 step: 125, loss is 0.13595285
```

Transformer 的训练过程如下：

```
train(train_cfg)
time: 428285, epoch: 15, step: 8832, outputs are [2.8881903]
time: 428318, epoch: 15, step: 8833, outputs are [2.8008752]
time: 428351, epoch: 15, step: 8834, outputs are [2.8877385]
time: 428384, epoch: 15, step: 8835, outputs are [2.863949]
time: 428417, epoch: 15, step: 8836, outputs are [2.8900979]
time: 428451, epoch: 15, step: 8837, outputs are [2.743007]
time: 428484, epoch: 15, step: 8838, outputs are [2.795813]
time: 428517, epoch: 15, step: 8839, outputs are [2.8064942]
time: 428550, epoch: 15, step: 8840, outputs are [2.9820175]
time: 428583, epoch: 15, step: 8841, outputs are [3.1082027]
time: 428616, epoch: 15, step: 8842, outputs are [2.987619]
time: 428649, epoch: 15, step: 8843, outputs are [3.0379832]
time: 428682, epoch: 15, step: 8844, outputs are [2.9345937]
time: 428715, epoch: 15, step: 8845, outputs are [3.10096]
time: 428749, epoch: 15, step: 8846, outputs are [3.0255766]
time: 428782, epoch: 15, step: 8847, outputs are [3.4345267]
time: 428815, epoch: 15, step: 8848, outputs are [3.2471128]
time: 428848, epoch: 15, step: 8849, outputs are [3.4878123]
time: 428881, epoch: 15, step: 8850, outputs are [3.834058]
epoch time: 21339.632 ms, per step time: 36.169 ms
```

2.3 自己的 Contribution

我在这次实验中自己做出的探索有：

- 对机器翻译模型的训练结果进行了效果的评估，在结果推理阶段使用不同类型的句子来检测模型的准确性和鲁棒性
- 调整训练的参数并进行模型训练效果的评估
- 学习了外国名校公开课如 Stanford-CS224N 和 CS231N，并完成了对应的 Seq2Seq 和 Transformer 相关的作业，对标上课所学内容进一步巩固所学知识，并实现了 Transformer 模型中的关键代码，将其用在 Image Capitioning 的任务上

三 实验结果

3.1 Seq2Seq 实验结果

Seq2Seq 模型的实验中提供了一个 evaluate 函数用来获得句子翻译的结果，然后可以用语法常识来判断翻译效果的好坏，首先我对于样例代码中给出的一个句子进行了多次变形并查看输出的翻译结果，得到如下结果：

```
translate('i love tom')
```

```
English ['i', 'love', 'tom']
中文 我爱汤姆。
```

```
translate('i very love tom')
```

```
English ['i', 'very', 'love', 'tom']
中文 我遇见汤姆了。
```

```
translate('i love tom so much')
```

```
English ['i', 'love', 'tom', 'so', 'much']
中文 我爱汤姆。
```

```
translate('i love tom very much')
```

```
English ['i', 'love', 'tom', 'very', 'much']
中文 我爱汤姆。
```

```
translate('i love very tom much lazy apple')
```

```
English ['i', 'love', 'very', 'tom', 'much', 'lazy', 'apple']
中文 我爱汤姆。
```

从这一系列翻译结果来看，我们可以得到如下结论：

- 使用了注意力机制的 Seq2Seq 模型具备了一定的简单句子翻译能力
- 该模型只拥有对句子大致意思的翻译能力，而对 very，so much 等表示程度大小的修饰词不敏感
- 最后一句很明显是一个不通顺的句子，但是 Seq2Seq 依然可以翻译出其核心的含义而忽略一些噪声词，说明该模型具有一定的鲁棒性
- 总的来说该模型具有一定的翻译能力和鲁棒性，但是对于句子中的一些修饰词等细节的刻画并不到位，我认为这和模型本身的架构有关，模型的编码器和解码器都只用了一层 GRU，可能对于单词信息的提取能力还是不太高，只能对一些出现频率比较高并且作为句子主干 (名词，动词) 的单词进行编解码。比如下面两句中的单词因为在语料库中出现比较少，导致翻译的效果也不是很好。

```
translate('I am still dream')
```

```
English ['i', 'am', 'still', 'dream']
中文 我还很好。
```

```
translate('He was made fair.')
```

```
English ['he', 'was', 'made', 'fair', '.']
中文 他很勇敢。
```

3.2 Transformer 实验结果

Transformer 的实验中，测试的时候将训练集中的所有的语句都进行了一次测试，因为数量比较多，我们按照句子的长度大致分成短中长三类，来比较 Transformer 在不同情况下的 performance，结果如下：

```
evaluate(eval_cfg)
```
```
source: Run .   你 用 跑 的 。
result: 坚 持 冷 点 。
source: Wait !  等 一 下 !
result: 等 等 !
source: I quit .        我 退 出 。
result: 我 已 经 不 辞 职 了 。
source: Really ?        你 确 定 ?
result: 记 得 特 别 的 消 息 疗 方 式 吗 ?
source: Be nice .       友 善 点 。
result: 和 雄 气 点 好 点 。
source: Go away .       走 开 !
result: 走 开 !
source: Goodbye !       再 见 !
result: 再 见 !
source: Goodbye !       告 辞 !
result: 再 见 !
source: Help me .       帮 我 一 下 。
result: 帮 我 吧 。
```

```
evaluate(eval_cfg)
```
```
source: He doesn ' t need to work .     他 不 需 要 工 作 。
result: 他 不 需 要 努 力 工 作 。
source: He got tired of reading .       他 厌 倦 了 读 书 。
result: 他 累 了 很 累 ， 看 得 累 了 。
source: He has no eye for women .       他 对 女 人 没 有 眼 光 。
result: 他 没 有 认 识 的 女 性 似 乎 没 有 看 着 的 女 人 。
source: He has written a letter .       他 写 了 一 封 信 。
result: 他 写 信 写 了 封 信 写 信 。
source: He is a friendly person .       他 是 一 个 友 善 的 人 。
result: 他 是 一 个 友 友 好 的 人 对 友 友 友 友 友 友 友 友 友 友 友 友 友 友 友 友 友 友 友 友 友 友 友 友 友 友 友 友 友 友
source: He is drawing a picture .       他 正 在 画 一 幅 画 。
result: 他 画 画 画 画 画 画 画 画 画 画 画 画 画 画 画 画 画 画 画 画 画 画 画 画 画 画 画 画 画 画 画 画 画 画 画 画 画 画 画
source: He is eager to go there .       他 渴 望 去 那 里 。
```

```
evaluate(eval_cfg)
```
```
source: A few years ago , our room had little furniture in it . 几 年 前 ， 我 们 的 房 间 里 没 有 什 么 家 具 。
result: 几 年 前 ， 我 们 家 的 房 间 里 有 一 点 钟 就 有 一 点 钟 前 家 家 家 家 里 的 家 家 家 家 家 家 家 里 里 就 有 1 0
source: Can you wake me up at seven o ' clock tomorrow morning ?       你 明 天 早 上 七 点 可 不 可 以 叫 我 起 床 。
result: 你 能 在 明 天 早 上 七 点 起 床 吗 ?
source: How long did it take you to finish reading that book ? 你 看 完 这 本 书 要 多 少 时 间 ?
result: 你 多 长 时 间 完 成 那 本 书 ?
source: I am not quite sure if we can meet your requirements .  我 不 太 确 定 我 们 是 否 能 够 满 足 你 的 要 求 。
result: 我 不 太 清 楚 我 们 能 满 足 你 的 高 。
source: I hear you passed the entrance exam . Congratulations ! 我 听 说 你 通 过 了 入 学 考 试 。 恭 喜 你 了 !
result: 我 听 到 你 在 考 试 的 声 音 ， 你 听 得 到 了 学 生 的 声 音 乐 。
source: If you need my advice , I ' d be glad to give it to you .       如 果 你 需 要 我 的 建 议 ， 我 很 愿 意 告 诉 你 。
result: 如 果 你 需 要 我 的 建 议 ， 我 需 要 你 可 能 会 很 高 把 它 给 你 打 给 你 可 以 把 它 送 你 可 以 把 它 可 以 把 它
source: My father has been in good shape since his operation .  自 从 手 术 后 ， 我 父 亲 一 直 处 于 良 好 的 状 态 。
result: 我 父 亲 让 自 己 的 好 认 为 照 顾 好 手 术 。
source: My father will often read the newspaper during meals .  我 父 亲 常 在 吃 饭 时 看 报 纸 。
result: 我 父 亲 经 常 在 吃 饭 后 读 报 纸 会 吃 饭 。
source: The Tigers lost the game , which was a surprise to us . 我 们 觉 得 很 惊 讶 老 虎 队 竟 然 输 掉 了 比 赛 。
result: 军 队 失 去 了 这 场 比 赛 ， 我 们 被 惊 讶 了 。
source: Tom bought a camera at a store not too far from here .  T o m 在 离 这 里 不 远 的 一 家 商 店 里 买 了 一 台 相 机 。
result: 汤 姆 在 这 里 买 了 一 台 相 机 器 的 照 相 机 不 太 远 了 。
```

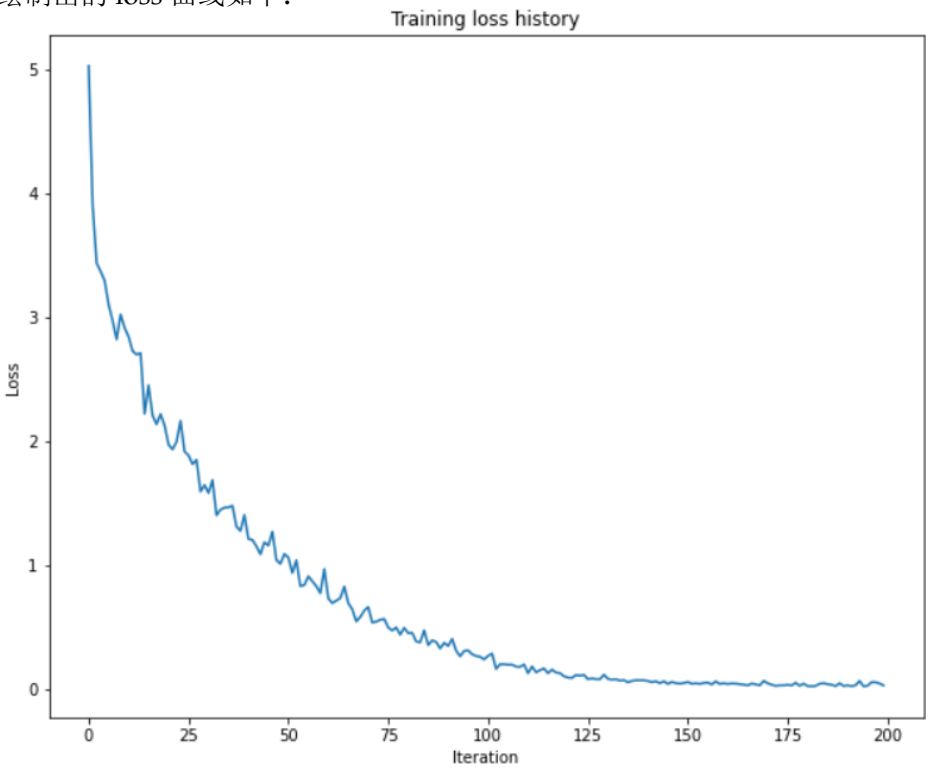
我们不难发现随着句子逐渐变得长和复杂 Transformer 的翻译效果逐渐变差，并且就算是比较短的句子，Transformer 也可能翻译的并不好，我认为这实际上和 Transformer 本身的架构有关，因为 **Transformer 模型本身没有卷积单元和循环单元等可以提取局部特征的架构，只依靠注意力机制进行依赖信息的提取**，而这样的做法需要更大量的训练数据去拟合整个模型，我们现在所使用的数据集对于 Transformer 的训练而言还远远不够，因此 Transformer 的效果并不好，不过作为一个尝试性质的实验也基本足够了。

3.2.1 自己做出的一些探索

我在完成基本的实验要求的基础上，完成了 CS231N 课程作业中的 Transformer 模型，并完成了其中的多头注意力机制和位置编码等关键的代码，并使用 adam 优化器对模型进行了训练和优化，得到的训练过程如下：

```
base dir  /Users/randomstar/Desktop/A-Walk-Through-ML/code/CS231N-labs/assignment3/cs231n/datasets/coco_captioning
(Iteration 1 / 200) loss: 5.023862
(Iteration 11 / 200) loss: 2.838943
(Iteration 21 / 200) loss: 1.969206
(Iteration 31 / 200) loss: 1.578379
(Iteration 41 / 200) loss: 1.207541
(Iteration 51 / 200) loss: 1.057905
(Iteration 61 / 200) loss: 0.728208
(Iteration 71 / 200) loss: 0.658397
(Iteration 81 / 200) loss: 0.447799
(Iteration 91 / 200) loss: 0.344530
(Iteration 101 / 200) loss: 0.264209
(Iteration 111 / 200) loss: 0.123995
(Iteration 121 / 200) loss: 0.089447
(Iteration 131 / 200) loss: 0.078627
(Iteration 141 / 200) loss: 0.061927
(Iteration 151 / 200) loss: 0.053362
(Iteration 161 / 200) loss: 0.037125
(Iteration 171 / 200) loss: 0.042968
(Iteration 181 / 200) loss: 0.020684
(Iteration 191 / 200) loss: 0.026406
```

绘制出的 loss 曲线如下：



在测试阶段，我们将一些图片输入模型中，就可以得到对应的 Capitioning，并且可以和标注好的原文进行一定的对比，这是其中一张图片的结果：

可以看到模型给图片做出了比较合适的 captioning，所以这次探索还是比较成功的。

四　总结思考

本次实验中，我完成了 Seq2Seq 模型和 Transformer 模型在机器翻译 task 上的应用，我深入地学习了这些模型的架构，并且探求模型的优点和不足之处，针对实验所提供的小规模数据集进行了适当的训练，并对训练结果进行评估和分析，得到了一系列结论来表明为什么有的时候模型的表现不尽人意。

此外我还学习了 Stanford CS224N 和 CS231N 中对应的内容，并手动实现了 Seq2Seq 和 Transformer 模型的关键代码，这对我而言也是一个不小的挑战。

参考文献

[1] http://web.stanford.edu/class/cs224n/index.html

[2] http://web.stanford.edu/class/cs224w/