Conference on Empirical Methods in Natural Language Processing(EMNLP),2017

Adversarial Learning for Neural Dialogue Generation

3120181078，郑安庆

3120181069，张　黎

3120181055，杨　芳

2019 年 5 月

Li J, Monroe W, Shi T, et al. Adversarial Learning for Neural Dialogue Generation. Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2017: 2157-2169.

一、模型描述

1.1 模型整体描述

这篇文章的主要工作在于应用了对抗训练（adversarial training）的思路来解决开放式对话生成（open-domain dialogue generation）问题。

其主要的思想就是将整体任务划分到两个子系统上：一个是生成器模型（Generative model），利用 seq2seq 式的模型以上文的句子作为输入，输出对应的对话语句；另一个则是一个判别器模型（Discriminative model），用于区分在给定上文的句子下当前的回答是否是和人类行为接近，这里可以近似地看作是一个二分类的分类器。两者结合的工作原理也很直观，生成器模型不断地根据前文生成答句，判别器模型则不断地用生成器模型的生成作为负例，原文的标准回答作为正例来强化分类。在二者的训练的过程中，生成器模型需要不断改良答案来欺骗判别器模型，判别器模型则需要不断提高自身的判别能力从而区分机器伪造和人造答案，直至最后两者的收敛达到某种均衡。

以往的模型受限于训练目标以及训练方式，其生成的结果往往是迟钝笼统的甚至都很简短（如果可以的话，所有的对话我都可以回答“我不知道”，很明显这样的回答是不符合常识的）。所以这样一种博弈式的训练方式来取代以往相对简单固定的概率似然来优化这样一种无监督的开放任务显然是很有意思的想法。和之前的 SeqGAN 类似，这篇文章也采取了强化学习来规避 GAN 在 NLP 中使用的难点，并做出了更多的尝试。

1.2 生成器模型

生成器模型 G 就是一个 SEQ2SEQ+attention 模型，输入是历史对话 x，然后通过 RNN 来对语义进行向量表示再逐一生成回答的每个词，从而形成回答 y。

如图 1 详细的描述了 SEQ2SEQ+attention 模型的全部流程。SEQ2SEQ 其实是一个 Encoder–Decoder 结构的网络，它的输入是一个序列，输出也是一个序列，Encoder 中将一个可变长度的信号序列变为固定长度的向量表达，Decoder 将这个固定长度的向量变成可变长度的目标的信号序列。

其中，左侧为 Encoder+输入，右侧为 Decoder+输出，中间为 Attention。

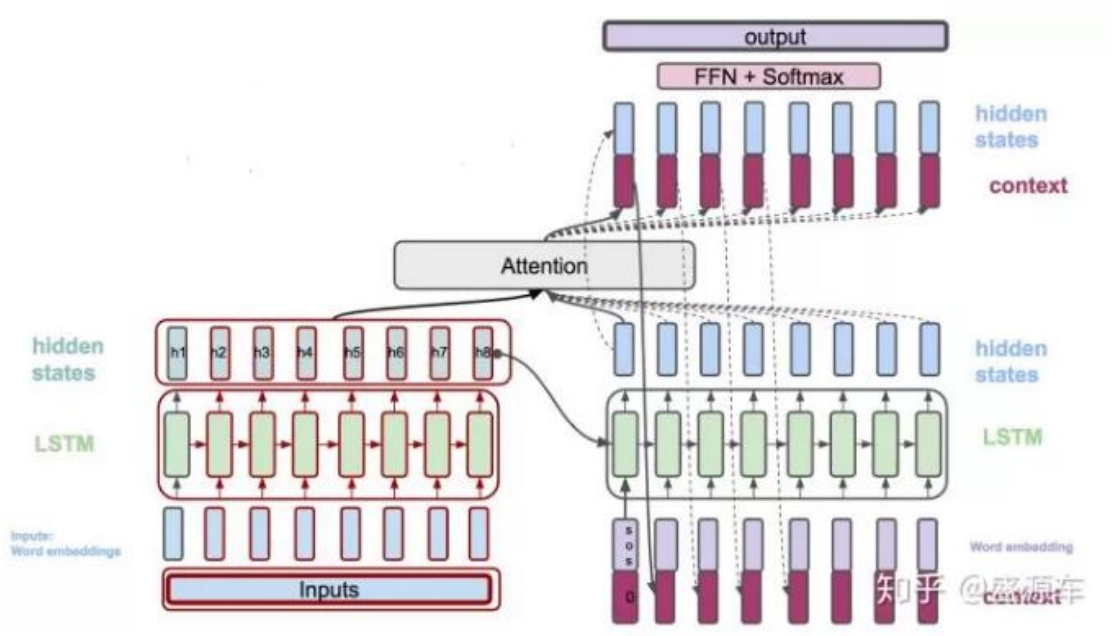

图 1 SEQ2SEQ 模型

输入：$x = (x_1, \dots, x_{T_x})$

输出：$y = (y_1, \dots, y_{T_y})$

(1) $h_t = RNN_{enc}(x_t, h_{t-1})$，Encoder 方面接受的是每一个单词的 embedding 和上一个时间点的 hidden state。输出的是这个时间点的 hidden state。

(2) $s_t = RNN_{dec}(\hat{y_{t-1}}, s_{t-1})$，Decoder 方面接受的是目标句子中的 embedding 和上一个时间点的 hidden state。

(3) $c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$，context vector 是一个对于 Encoder 输出的 hidden states 的一个加权平均。

(4) $\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$，每一个 Encoder 的 hidden states 的对应的权重。

(5) $e_{ij} = score(s_{i-1}, h_j)$，通过 Decoder 的 hidden states 加上 Encoder 的 hidden states 来计算一个分数，用于计算权重（4）。

(6) $\hat{s_t} = \tanh(W_c[c_t : s_t])$，将 context vector 和 Decoder 的 hidden states 拼接起来。

(7) $p(y_t | y_{<t}, x) = \text{softmax}(W_s \hat{s_t})$，计算最后的输出概率。

(8)

1.3 判别器模型

判别器模型 D 是一个二分类器，将对话序列{历史对话 x，当前回答 y}作为输入，并输出一个标签来判断该回答 y 是机器伪造的答案$Q_-(\{x, y\})$还是人造的答案$Q_+(\{x, y\})$。该部分使用分层编码器（hierarchical encoder）将输入编码成一个向量表示，最后返回属于这两种分类的概率。

1.4 强化学习

1.4.1 Policy-Gradient-Training

本篇文章的关键思想就是促使生成器模型能够生成与人类话语混淆的语句，作者使用策略梯度方法来达到这个目标。可以将 sentence 生成的过程视为一个动作（action）序列，将生成器模型视为一个策略（policy），判别器模型将生成器模型生成的句子看做人生成的得分作为奖励（reward），该奖励可以作为生成模型的奖励。本文采用了 policy gradient 的方法（强化学习的方式之一）来进行强化学习的训练，其优化目标是最大化回报期望。

$$\mathrm{J}(\theta) = \mathbb{E}_{\mathrm{y}\sim\mathrm{p}(\mathrm{y}|\mathrm{x})}(Q_{+}(\{x,y\})|\theta) \quad (1)$$

这里可以使用似然率来近似公式（1）。

$$\begin{aligned}\nabla \mathrm{J}(\theta) &\approx [Q_{+}(\{x,y\}) - b(\{x,y\})]\nabla \log \pi(\mathrm{y}|\mathrm{x}) \\ &= [Q_{+}(\{x,y\}) - b(\{x,y\})]\nabla \textstyle\sum_{\mathrm{t}} \log \mathrm{p}(y_t|x, y_{1:t-1})\end{aligned} \quad (2)$$

该式表示给定一个对话输入 x，生成器模型通过该策略生成回答 y，x 生成 y 的概率等于逐词生成的概率，如公式（2）所示，这里也可以很好地和 seq2seq 的工作机理对应上。之后再将对话序列{历史对话 x，当前回答 y}作为判别器模型的输出。其中，$b(\{x,y\})$则是一个使得训练稳定的平衡项。

1.4.2 Reward for Every Generation Step (REGS)

但是，简单使用 policy gradient 方法是有缺陷的，因为我们的评估都是针对一整个回答的，而判别器只会给出一个近似于对或者不对的反馈。这样的方法存在一个很大的问题是，即使是很多被判断为有问题的句子，但是其中有很大一部分语言成分是有效的，如文中的例子“what’s your name”，人类回答“I am John”，机器回答“I don’t know”。判别器模型会给出“I don’t know”是有问题的，但无法给出 I 是对的，而后面的 don’t know 是错的。事实上机器没有回答 he/she/you/they，所以需要对 I 给一个肯定的正反馈。但是判别器模型只告诉机器对或错，却不告知哪部分对和哪部分错，这对训练带来了很大隐患。作者对于 reward 只来自于整体句子生成后判别器模型的判别，对于生成每个词的每一步没有一个明确的 reward 去评价句子中每个词生成的好坏的问题，提出了 Reward for Every Generation Step 方法。文中提出了两种实现 REGS 的途径：1.蒙特卡洛搜索（Monte Carlo search）；2.训练一个可以给不完整序列以奖励的判别器。在复现论文时，我们使用了效果更好的蒙特卡洛搜索。

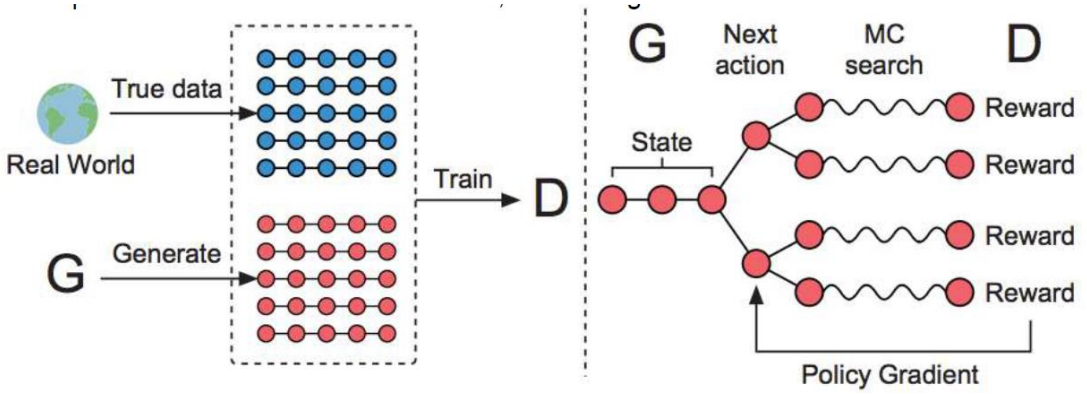

图 2 蒙特卡洛搜索

对于当前的一个已生成的部分文本，剩下的部分采用随机采样的方法采样到结束。即在生成到当前时刻 t 时，假定已经得到一个前缀，为了生成当前这个词的 reward，利用 Monte Carlo search 方法，继续在该前缀的基础上利用同一个 seq2seq 模型（参数就是当前这个 seq2seq 状态下的参数）采样得到 N 个完整的句子，也就是说生成当前这个词后，为了判别当前这个词的 reward，继续在该词的基础上接着跑完整个句子的生成，每一步都是采样一个词，而不是取概率最大的那个词。这样得到 N 个句子后，再用判别器模型进行判别得到 N 个整体句子的 reward，然后平均值便作为当前这个词的 reward。这种 Monte Carlo search 方法也是另一篇稍早些同样比较知名工作 Seq-GAN 的做法。实际上本文与 Seq-GAN 的框架基本一致，只有若干细节不太一样。这里对于每一步生成一个词选用的 reward 方法便是其中之一，本文认为 Monte Carlo search 太过于耗时，因为每生成一个词，都要采样 N 个完整句子。

1.4.3 Teacher Forcing

事实上，通过 1.4.1 提到的 RL 或者 1.4.2 的 REGS 方法来训练生成器会非常不稳定。主要原因是生成器模型根据判别器模型反馈的 reward 值间接搜索最优目标，并且该 reward 值被用来促进或者抑制生成器自身的序列生成。根据判别器的输出分数，生成器模型知道它生成的模型很差，但是它不知道什么是好的，并且也不知道怎样才能让自己生成好的序列，况且真实的搜索空间却又无比巨大，因此生成一条好句子的概率是很小的，因此我们在利用强化学习之前需要先给生成器一个基础，也就是先告诉生成器什么是好的。

因此，为了使生成器更有方向性的达到最有目标，在生成器模型的训练中，作者选择将人生成的回应句子也加入其中。对于判别器而言，最直接的策略就是对人生成的回应句子奖励值设为 1 或其他的正数值；对于生成器模型而言，可以使用该奖励值基于人生成的样例来更新自身。这可以视为有一个老师干预的过程，这期间会迫使生成器生成类人的回应句子。

1.5 实验结果

1.5.1 效果较好的对话

```
> what are you doing now ?
i ' m going to be fine
```

```
> why don 't you say ' i ' m sorry ' ?
i ' m not gonna hurt you
```

```
> you ' r e just fooling
i ' m not going to be a good man
```

```
> but why do you care ?
i don ' t care about it
```

```
> are you with me or not ?
i ' m not a stranger
```

```
> what are you going to do ?
i ' m going to kill you
```

```
> who do you think i am ?
i ' m not a nurse
```

```
> how many bugs you have ?
i don ' t know
```

```
> i love you
i ' m not a little girl
```

1.5.2 效果较差的对话

```
> i love you .
i ' m a little bit of a bit
```

```
> what ' s your name ?
i ' m not my friend
```

```
> tell me ... how long have you had this falling sickness ?
i ' m not . . . . . .
```

1.5.3 与论文实验结果对比：

原文对话生成 result(只考虑效果最好的 REGS Monte Carlo 方法)：

Input	tell me ... how long have you had this falling sickness ?
Vanilla-MLE	i 'm not a doctor .
Vanilla-Sample	well everything you did was totally untrue .
REINFORCE	i don 't know how long it 's been .
REGS Monte Carlo	A few months, I guess .
Input	so i had the doctors test sammy 's response to conditioning .
Vanilla-MLE	sammy wrote the test sammy wrote the test .
Vanilla-Sample	objects pick objects up objects objects objects objects objects
REINFORCE	i 'm not sure that 's relevant .
REGS Monte Carlo	so he took the pills .
Input	what are you going to do ?
Vanilla-MLE	i 'm going to the movies .
Vanilla-Sample	get him outta here first !
REINFORCE	i 'm going to get you .
REGS Monte Carlo	i 'm going to get you a new car .
Input	they fear your power your intellect .
Vanilla-MLE	you 're the only one who knows what 's going on .
Vanilla-Sample	when they are conquered and you surrender they will control all of us .
REINFORCE	i 'm afraid i 'm not ready yet .
REGS Monte Carlo	i 'm not afraid of your power .

我们对话生成结果：

```
> tell me ... how long have you had this falling sickness ?
i ' m not . . . . . .
> so i had the doctors test sammy ' s response to conditioning .
i ' m sure it ' s okay .
> what are you going to do ?
i ' m going to go
> they fear your power your intellect .
you ' r e right
```

结果分析：

从二者结果可知，我们的结果还远远未到作者的水平，主要原因就在于OpenSubtitles 数据集有很多版本，文中没有明确说明使用了哪个版本。而且由于硬件和训练时间的限制，我们只能选择较小的数据集。我们查看了数据集，发现句子中有一些"UNknown"，明显与句子脱节，可能数据集本身或者我们处理有一些问题。两方面的因素最终导致了达不到文中的效果。但是从 1.5.1 节中好的对话结果来看，我们复现的结果较多可以通过图灵测试，因此还是不错的。

二、开发环境

操作系统：Ubuntu 18.04.2 LTS

开发语言：Python 3.6.7

所需软件包：tensorflow 1.12.0

numpy 1.15.4

三、数据集说明

原文中使用了 OpenSubtitles 数据集(http://opus.lingfil.uu.se/OpenSubtitles.php)，该数据集是开放的电影字幕的数据集，它包含 XML 格式的电影中人物所说的话，我们使用作者提供的未处理的数据集得到 8000 万条对话的数据集。由于轮流说话没有明确表示，我们将连续的句子看做是不同的角色的人说的，将上一句当成 query，下一句当成 answer，我们针对每一句都这么做。由于该 OpenSubtitles 是相当大的和嘈杂的，因为连续的句子有可能是同一个人物所说的，在此基础上我们得到的对话生成系统较差。

Cornell 数据集(http://www.cs.cornell.edu/~cristian/Cornell_Movie-Dialogs_Corpus.html)同样也是开放的电影字幕的数据集，但该数据集的每句话包含人物信息，因此我们可以很简单的得到真实的对答，但该数据集只包含 30 万条对话，数量较少，最后生成的对话系统较为固定。

因此最后我们使用了 Cornell 数据集和 OpenSubtitles 数据集，并将处理后的数据集按照训练集和验证集以及上文和下文存放到 gen_data 下，分别命名为

chitchat.train.query 、 chitchat.train.answer 、 chitchat.dev.query 和 chitchat.dev.answer。

四、源码说明

本项目共有 config.py、dis_pre_train.py、train.py、test.py 等 12 个 py 文件，还包含了 gen_data/chitchat.train.query 、 gen_data/chitchat.train.answer 等数据集文件，为了便于检查和测试，还包含了 dis_data/train.query 、 dis_data/train.answer 等中间数据文件，和 gen_data/checkpoints、dis_data/checkpoints 等已训练好的权值文件。接下来具体介绍这些文件。

model/dis_model.py 定义了判别器模型结构，model/gen_model.py 定义了生成器模型结构，model/seq2seq.py 定义了 attention、sequence_loss 等基础的组件供 model/gen_model.py 使用。

config.py 定义了生成器和判别器训练中使用的超参数、路径等。

dis_pre_train.py 为判别器的预训练代码，它读取生成器生成的数据，创建判别器模型并预训练后将参数存至 dis_data/checkpoints。

discriminator.py 提供了判别器的高级封装，包括 read_data、get_batch、create_model 等函数。

gen_data.py 为生成器生成数据的代码，它读取生成器预训练后的权值，并运行生成器，生成数据并保存为 dis_data/train.answer、dis_data/train.gen、dis_data/train.query 、 dis_data/dev.answer 、 dis_data/dev.gen 、 dis_data/dev.query 供判别器预训练。

gen_pre_train.py 为生成器预训练代码，它读取原始数据集文件（ gen_data/chitchat.dev.answer 、 gen_data/chitchat.dev.query 、 gen_data/chitchat.train.answer、gen_data/chitchat.train.query），创建生成器模型并预训练，然后将预训练后的权值保存至 gen_data/checkpoints。

generator.py 提供了生成器的高级封装，包括 read_data、create_model 等函数。

test.py 为测试代码，它读取训练后的生成器参数，与用户交互，输出回答。

train.py 为训练代码，它读取经过预训练的生成器和判别器模型参数，进行对抗训练，然后保存训练后的新参数。

util.py 为其他代码提供了一些实用函数，包括 create_vocabulary、sentence_to_token_ids 等函数。

其他文件，如 chitchat.dev.ids35000.answer、vocab35000.all 等为预处理

后的数据文件。

训练过程说明

训练过程分为四个步骤，分别如下：

1. python gen_pre_train.py 预训练生成器。需要有gen_data/chitchat.dev.answer等四个文件才能运行,也可在config.py中修改各种设置。运行后，gen_data/checkpoints 中将储存经过预训练的权值文件。
2. python gen_data.py 读取生成器预训练后的权值，为判别器预训练过程生成数据。运行后，将会生成 dis_data/dev.answer 等六个文件。
3. python dis_pre_train.py 预训练判别器。需要有上一步生成的六个文件才能运行。运行后，dis_data/checkpoints 中将储存经过预训练的权值文件。
4. python train.py 读取经过预训练的生成器和判别器权值，进行对抗训练，并将权值文件保存。

五、测试程序说明

测试程序为 test.py。执行 python test.py 后，程序将读取 gen_data/checkpoints 中训练好的权值文件，进行人机交互——程序将等待用户输入，然后根据用户输入，输出回应，直到用户输入 Ctrl+Z 退出。

权值文件的获得方式已经在上文的训练过程说明中阐述了。为了方便测试，我们附带了已经训练好的权值文件，可供直接测试。