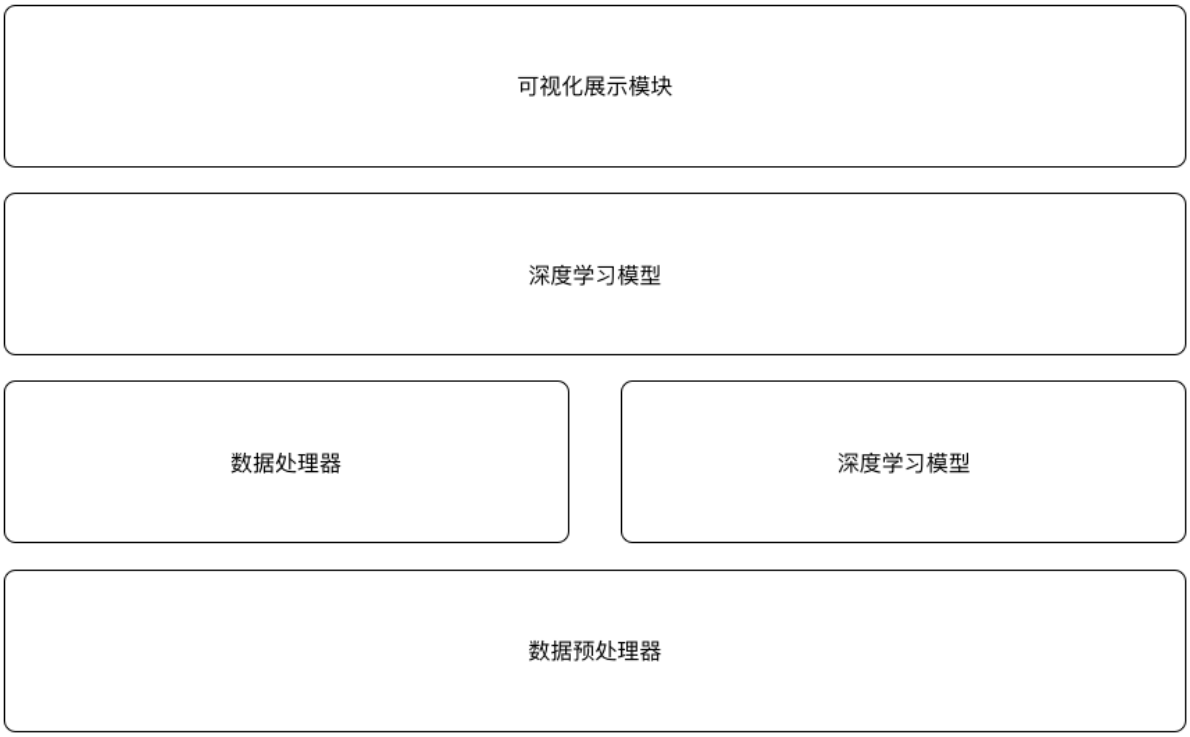
手把手教你写一个中文聊天机器人

前言：

发布这篇 Chat 的初衷是想和各位一起分享一下动手来做聊天机器人的乐趣，因此本篇文章适合用于深度机器学习的研究和兴趣发展，因为从工业应用的角度来看使用百度、科大讯飞的 API 接口会更加的适合。在这篇文章中，希望和大家一起共同交流和探索动手实践的乐趣，当然也欢迎大神来做深度的探讨以及吐槽。这篇 Chat 的基础源代码来自互联网，我进行了综合优化和部分代码的重写，我也会在这边文章发布的同时将所有源代码上传到 Git 分享出来，这样在文章中我就不占用篇幅贴出全部的源代码，大家可以从 Git 上 pull 下来对照着文章来看。

系统设计思路和框架

本次系统全部使用 Python 编写，在系统设计上遵循着配置灵活、代码模块化的思路，分为数据预处理器、数据处理器、执行器、深度学习模型、可视化展示五个模块。模块间的逻辑关系大致为：数据预处理是将原始语料进行初步的处理以满足于数据处理模块的要求；执行器是整个系统引擎分别在运转的时候调用数据处理器、深度学习模型进行数据处理、模型训练、模型运作等工作；深度学习模型是一个基于TF的seq2seq模型，用于定义神经网络并进行模型计算；可视化展示是一个用Flask前端框架写的简单的人机交互程序，在运行时调用执行器进行人机对话。整体的框架图如下：



本系统利用seq2seq模型的特点，结合word2vec的思路（当然这样做有点简单粗暴，没有进行分词），将训练语料分为Ask语料集和Replay语料集，并根据一定的比例分为训练语料集和验证语料集，然后就是word2vec。这些步骤都是在数据预处理器和数据处理器中完成的，关于训练语料集与验证语料集的数量都是做成可外部配置的，这也是这个系统我认为设计的最合理的地方（可惜不是我设计的）。在完成数据的处理后，执行器就会根据训练模式（这也是外部可配置的）来调用seq2seq进行神经网络的创建，神经网络的超参数同样也是可以外部进行配置的。在进行训练过程中，使用perprelixy来计算模型的loss，通过自动的调整learning rate来逐步的取得最优值，当learning rate减少为0达到最优值。最后，就是可视化展示模块启动一个进程调用执行器来实时在线提供聊天服务，在语句输入和输出利用seq2seq的特点，直接将输入seq转换成vec作为已经训练好的神经网络，然后神经网络会生成一个seq向量，然后通过查询词典的方式将生成的向量替换成中文句子。在神经网络模型这里，TF有GRU和LSTM模型可供选择，我比较了GRU和LSTM的训练效果，发现还是GRU比较适合对话语句场景的训练，这点在后面的代码分析环节会详细解释。

源码结构

- data_utls.py（数据预处理器）包含的函数convert_seq2seq_files，将主程序分好的ask语料集和response语料集转换成seq2seq文件，以便数据处理器进行进一步的数据处理。对于整个系统，一般该数据预处理器只需要运行一次即可。
- prepareData.py（数据处理器）包含的函数：create_vocabulary、convert_to_vector、prepare_custom_data、basic_tokenizer、initialize_vocabulary，这些函数的作用分别是创建字典、句子转向量、根据超参定制化训练数据、基础数据标记、初始化词典。
- execute.py（执行器）包含的函数：get_config、read_data、create_model、train、self_test、init_session、decode_line，这些函数的作用分别是获取配置参数和超参、读取数据、创建模型、训练模型、模式测试、初始化会话、在线对话。
- seq2seq_model.py（深度机器学习模型）包含的函数：**init**、sampled_loss、seq2seq_f、step、get_batch，这些函数的作用分别是程序初始化、loss函数、seq2seq函数、拟合模型、获取批量数据。
- app.py（可视化展示模块）包含的函数：heartbeat、reply、index，这些函数的作用分别是心跳、在线对话、主页入口。

源码讲解

数据预处理器（data_utls.py)

首先在代码的头部声明代码编码类型，`#!/usr/bin/env Python #coding=utf-8`，然后导入所需的依赖：

```
import os
import random
```

```
import getConfig
```

os是提供对python进行一些操作系统层面的工具，比如read、open等，主要是对文件的一些操作。
random是一个随机函数，提供对数据的随机分布操作。
getConfig是外部定义的一个函数，用来获取seq2seq.ini里的参数配置。
接下来是对原始语料的处理，主要是通过语句的奇偶行数来区分问答语句（由于语料是电影的台词，所以直接默认是一问一答的方式），然后把区分后的语句分别存储下来，保存为ask文件和response文件。

```
 gConfig = {}
gConfig=getConfig.get_config()
conv_path = gConfig['resource_data']
if not os.path.exists(conv_path):
    exit()
convs = []  # 用于存储对话集合
with open(conv_path) as f:
    one_conv = []         # 存储一次完整对话
    for line in f:
        line = line.strip('\n').replace('/', '')
        if line == '':
            continue
        if line[0] == gConfig['e']:
            if one_conv:
                convs.append(one_conv)
            one_conv = []
        elif line[0] == gConfig['m']:
            one_conv.append(line.split(' ')[1])

ask = []         # 用来存储问的语句
response = []    # 用来存储回答的语句
for conv in convs:
    if len(conv) == 1:
        continue
    if len(conv) % 2 != 0:  # 保持对话是一问一答
        conv = conv[:-1]
    for i in range(len(conv)):
        if i % 2 == 0:
            ask.append(conv[i]) #因为这里的i是从0开始的，因此偶数为问的语句，奇数为回答的语句
        else:
            response.append(conv[i])
```

然后调用convert_seq2seq_files函数，将区分后的问答语句分别保存成训练集和测试机，保存文件为train.enc、train.dec、test.enc、test.dec。convert_seq2seq_files函数就不一一贴出来，可以对照源码来看。

数据处理器（prepareData.py）

编码声明和导出依赖部分不再重复，在真正的数据处理前，需要对训练集和测试集中的一些特殊字符进行标记，以便能够进行统一的数据转换。特殊标记如下：

```
PAD = "__PAD__"#空白补位
GO = "__GO__"#对话开始
EOS = "__EOS__"  # 对话结束
UNK = "__UNK__"  # 标记未出现在词汇表中的字符
START_VOCABULART = [PAD, GO, EOS, UNK]
PAD_ID = 0#特殊标记对应的向量值
GO_ID = 1#特殊标记对应的向量值
EOS_ID = 2#特殊标记对应的向量值
UNK_ID = 3#特殊标记对应的向量值
```

接下来定义生成词典函数，这里及后续的函数只贴出函数名和参数部分，详细的代码参见Git上的源码：

```
def create_vocabulary(input_file,output_file):
```

这个函数算法思路会将input_file中的字符出现的次数进行统计，并按照从小到大的顺序排列，每个字符对应的排序序号就是它在词典中的编码，这样就形成了一个key-vlaue的字典查询表。当然函数里可以根据实际情况设置字典的大小。


```
def convert_to_vector(input_file, vocabulary_file, output_file):
```

这个函数从参数中就可以看出是直接将输入文件的内容按照词典的对应关系，将语句替换成向量，这也是所有seq2seq处理的步骤，因为完成这一步之后，不管原训练语料是什么语言都没有区别了，因为对于训练模型来说都是数字化的向量。

```
def prepare_custom_data(working_directory, train_enc, train_dec,
test_enc, test_dec, enc_vocabulary_size, dec_vocabulary_size,
tokenizer=None):
```

这个函数是数据处理器的集成函数，执行器调用的数据处理器的函数也主要是调用这个函数，这个函数是将预处理的数据从生成字典到转换成向量一次性搞定，将数据处理器对于执行器来说实现透明化。working_directory这个参数是存放训练数据、训练模型的文件夹路径，其他参数不一一介绍。

seq2seq 模型

seq2seq模型是直接参照TF官方的源码来做的，只是对其中的一些tf参数针对tf的版本进行了修正。如Chat简介中说的，考虑到大家的接受程度不一样，本次不对代码、算法进行太过深入的分析，后面我会开一个达人课专门详细的进行分析和相关知识的讲解。

```
class Seq2SeqModel(object):
```

前面的导入依赖不赘述了，这里在开头需要先定义一个Seq2SeqModel对象，这个对象实现了一个多层的RNN神经网络以及具有attention-based解码器，其实就是一个白盒的神经网络对象，我们只需拿来用即可，详细的可以参阅http://arxiv.org/abs/1412.7449这个paper。关于这个paper这次不做解读。

```
 def __init__(self, source_vocab_size, target_vocab_size, buckets,
size,
           num_layers, max_gradient_norm, batch_size,
learning_rate,
           learning_rate_decay_factor, use_lstm=False,
           num_samples=512, forward_only=False):
```

这个函数是整个模型的初始化函数（父函数），这个函数里面会定义多个子函数，简单来讲这个函数执行完之后就完成了训练模型的创建。这个函数中的大部分参数都是通过seq2seq.ini文件进行朝参数配置的，其中use_lstm这个参数是决定是使用gru cell还是lstm cell来构建神经网络，gru其实是lstm的变种，我两个cell都测试了，发现在进行语句对话训练时使用gru cell效果会更好，而且好的不是一点。由于时间的缘故，只对超参size、num_layers、learning_rate、learning_rate_decay_factor、use_lstm进行简单对比调试，大家有兴趣的话可以自己进行调参，看看最优的结果值preprlexity会不会小于10。

sampled_loss、seq2seq_f、step、get_batch这些子函数不一一的讲了，大家可以百度一下，都有很详细的解释和讲解。如果需要，我会在达人课里对这些子函数进行讲解。

执行器

```
_buckets = [(1, 10), (10, 15), (20, 25), (40, 50)]
```

这个bukets的设置特别关键，也算是一个超参数，因为这个关系到模型训练的效率。具体设置的时候，有两个大原则：尽量覆盖到所有的语句长度、每个bucket覆盖的语句数量尽量均衡。

```
def read_data(source_path, target_path, max_size=None):
```

这个函数是读取数据函数，参数也比较简单，其中max_size这个参数默认是空或者是None的时候表示无限制，同时这个参数也是可以通过seq2seq.ini进行设置。

```
def create_model(session, forward_only):
```

这个函数是用来生成模型，参数简单。model的定义也只有一句：

```
    model = seq2seq_model.Seq2SeqModel( gConfig['enc_vocab_size'],
gConfig['dec_vocab_size'], _buckets, gConfig['layer_size'],
gConfig['num_layers'], gConfig['max_gradient_norm'],
gConfig['batch_size'], gConfig['learning_rate'],
gConfig['learning_rate_decay_factor'], forward_only=forward_only)
```

这里可以看到，模型的生成直接调用了seq2seq_model中的对象Seq2SeqModel，将相应的参数按照要求传进去就可以，具体这个对象的作用以及详细的细节如前面所说可以参照具体的paper来研究，但是作为入门的兴趣爱好者来说可以先不管这些细节，先拿来用就可以了，主要关注点建议还是在调参上。

```
    def train():
```

train函数没有参数传递，因为所有的参数都是通过gconfig来读取的，这里面有一个特殊的设计，就是将prepareData函数调用放在train()函数里，这样做的话就是每次进行训练时都会对数据进行处理一次，我认为这是非常好的设计，大家可以参考，因为这个可以保证数据的最新以及可以对增长的数据进行训练。具体代码如下：

```
      enc_train, dec_train, enc_dev, dec_dev, _, _ =
prepareData.prepare_custom_data(gConfig['working_directory'],gConf
ig['train_enc'],gConfig['train_dec'],gConfig['test_enc'],gConfig['
test_dec'],gConfig['enc_vocab_size'],gConfig['dec_vocab_size'],tok
enizer=None)
```

def self_test():和def init_session(sess, conf='seq2seq.ini'): 这两个函数分别是进行测试以及初始会话用的。由于TF的特殊机制，其每次图运算都是要在session下进行的，因此需要在进行图运算之前进行会话初始化。


```
def decode_line(sess, model, enc_vocab, rev_dec_vocab, sentence):
```

这个函数就是我们整个对话机器人的最终出效果的函数，这个函数会加载训练好的模型，将输入的sentence转换为向量输入模型，然后得到模型的生成向量，最终通过字典转换后返回生成的语句。

由于执行器包含多种模式，因此我们在最后加上一个主函数入口并对执行模式判断，

```
if __name__ == '__main__':
if gConfig['mode'] == 'train':
    # start training
    train()
elif gConfig['mode'] == 'test':
    # interactive decode
    decode()
 else:
    print('Serve Usage : >> python3 webui/app.py') #当我们使用可视化
模块调用执行器时，需要在可视化模块所在的目录下进行调用，而是可视化模块由于包含很
多静态文件，所以统一放在webui目录下，因此需要将执行器与可视化模块放在同一个目录
下。
    print('# uses seq2seq_serve.ini as conf
file')#seq2seq_serve.ini与seq2seq.ini除了执行器的模式外所有配置需要保持一
致。
```

可视化展示模块

```
def heartbeat():
```

由于可视化展示模块需要账期在线运行，为了了解运行状态加了一个心跳函数，定时的输出信息来check程序运行情况。

```
def reply():
```

这个函数是人机对话交互模块，主要是从页面上获取提交的信息，然后调用执行器获得生成的回答语句，然后返回给前端页面。

其中有一点设计可以注意一下，就是在进行语句转向量的过程中，为了保证准确识别向量值，需要在向量值中间加空格，比如123，这个默认的会识别成一个字符，但是1 2 3就是三个字符。因此在获取到前端的语句后，在传给执行器之前需要对语句进行字符间加空格处理，如下：

```
req_msg=''.join([f+' ' for fh in req_msg for f in fh])
```

```
def index():
```


这个函数是可视化展示模块的首页加载，默然返回一个首页html文件。

另外，由于TF的特殊机制，需要在可视化模块中初始化会话，这样执行器才能运行，如下：

```
    import tensorflow as tf
    import execute
    sess = tf.Session()
    sess, model, enc_vocab, rev_dec_vocab =
 execute.init_session(sess, conf='seq2seq_serve.ini')
```

最后和执行器一样，需要加一个主函数入口启动可视化模块，并配置服务地址和端口号，如下：

```
if (__name__ == "__main__"):
app.run(host = '0.0.0.0', port = 8808)
```

训练过程和最优值

cell model	layers	units	first earning rate	last leaning rate	trand	steps	preplexity
GRU	3	256	0.5	0	down	1300000	10
GRU	3	512	0.5	0.0625	down	13000	20
GRU	3	512	1		up		inf
GRU	3	512	0.8	0.0118	down	123300	15
GRU	2	512	0.8	0.0002	down	148800	16.57
GRU	5	512	0.8	0.0638	down	92400	25.02
LSTM	3	256	0.5	0	down	166500	21

最优效果展示

展示地址：http://115.231.97.140:8999/