NLP 经典应用：聊天机器人的实现

实验报告

课程名称：人工智能导论

专业：数据科学与大数据技术

年级：2017 级

姓名：刘颖凡

学号：10172100123

目录

一、应用背景简介

自人工智能浪潮袭来，人们的生活也因各种智能产品得到了巨大的改观，智能聊天机器人就是其中之一。总体而言，市场上的聊天机器人大致分为三种类型：**基于规则的模型**(Rule-based model) 、**基于检索的模型**(Retrieval-based model) 、**生成式模型**(Generative model)。

1. 基于规则的模型

如下所示，若问句中包含某个特定词，则机器人给予特定回答。这种方式需要尽心编写规则，还要考虑到规则间的优先顺序

```
if  "你好吗"  in user .query :
      chatbot.say( "我很好" )
if  "天气"  in user .query :
      chatbot.say( "今天天气真好" )
```

图 1.1 规则库示例

2. 基于检索的模型

检索式模型其实就是类似问答系统，我们会维护问题与答案的配对，再根据使用者的输入的问题和 Questions 中的哪一个 q 最接近，就把 Answers 中配对的回覆丢给使用者。基于检索的模型的关键技术包括**提取关键词和相似度计算**，计算使用者的问句与问答库中那个问句最相似。

$$Questions = q_1, q_2, \dots \dots, q_n$$

$$Answers = a_1, a_2, \dots \dots, a_n$$

图 1.2 问答规则示例

3. 生成式模型

自从 google 的论文发表后，用 Sequence to Sequence 来实现聊天机器人就成为一股热潮。 Sequence to Sequence 的基本概念是**串接两个 RNN/LSTM**，一个当作**编码器**，把句子转换成隐含表示式，另一个当作**解码器**，将记忆与目前的输入做某种处理后再输出序列生成，就是把前一刻的输出当成下一刻的输入。

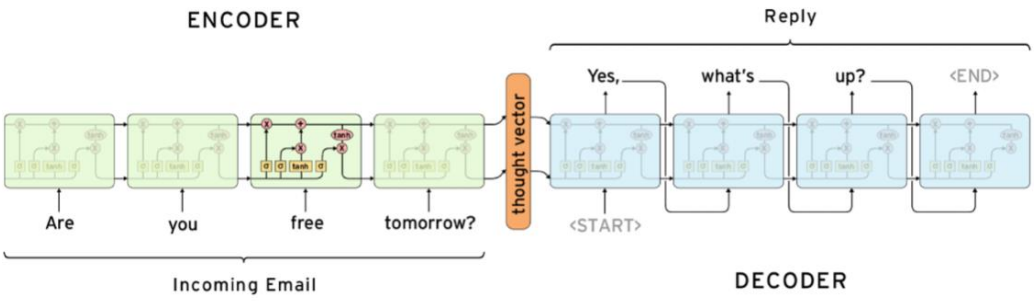

图 1.3 Seq2Seq 原理示意图

二、实验目的

1. 本次实验意在学习自然语言处理相关知识，实现了简单的生成式

聊天机器人模型和基于检索的聊天机器人模型，分析并思考最终的准确度差异；

2. 在生成式的聊天机器人模型中，利用 Tensorflow 搭建神经网络实现 Sequence to Sequence 模型；

3. 在基于检索的聊天机器人模型中，利用 python 自带的 fuzzywuzzy 模糊字符串匹配工具包，根据相似度原则在问答库中搜索中最佳的匹配答案。

三、实验前期准备

1. 训练语料：小黄鸡对话语料 chicken_and_gossip.txt

2. 开发环境：Python 3.7

Tensorflow 1.13.1

Jieba 0.39

Genism 3.6.0

四、实验内容

（一）数据预处理

1.文本预处理

在/Chatbot-Fannie/utils/text_preprocess.py 实现

编写了多个模块函数，主要对文本进行**去除无效字符（标点符号、空格等）、结巴分词、加载词向量**等处理，以便后续程序使用。

```python
#去除标点符号、空格
def clear_punctuation(text):
    """去除标点符号"""
    sentence = text.replace(' ', '')
    sentence_punctuation_clear = re.sub(filters, ' ', sentence).strip()
    sentence_punctuation_clear_replace = sentence_punctuation_clear.replace('　　', ' ').replace('  ', ' ')
    return sentence_punctuation_clear_replace

#截取中文、拼音、数字，去除特殊字符等
def getChinesel(ques):
    findAllChinese = ''.join(re.findall(u"([\u4e00-\u9fa50-9A-Za-z])", ques))
    return findAllChinese
```

图 4.1 数据预处理部分代码示例

2.词向量训练

在/Chatbot-Fannie/utils/word2vec_vector.py 实现

利用 gensim 库函数**加载词向量**。

（二）生成式模型聊天机器人

1. 搭建 LSTM 神经网络

在/Chatbot-Fannie/utils/model_utils/model_seq2seq.py 实现

1.1 构建编码器

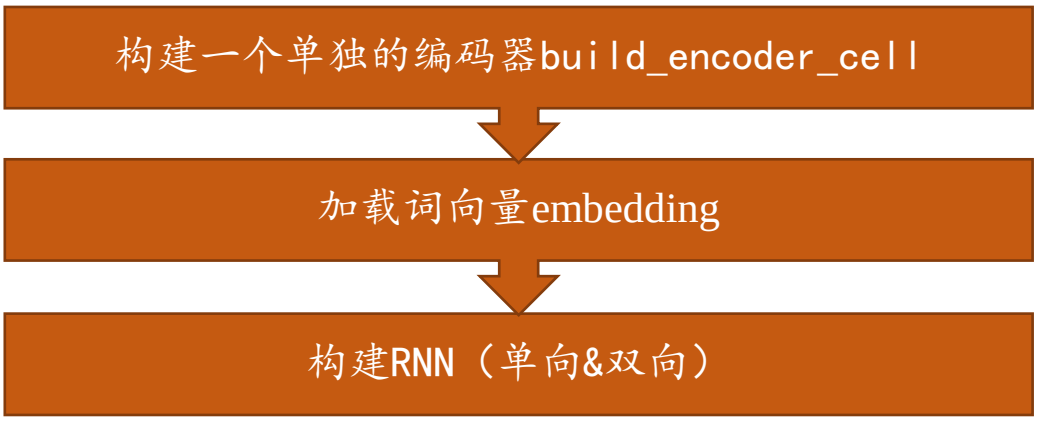

图 4.2 编码器构造思路

```
# 单向 RNN
(
    encoder_outputs,
    encoder_state
) = tf.nn.dynamic_rnn(
    cell=encoder_cell,
    inputs=inputs,
    sequence_length=self.encoder_inputs_length,
    dtype=tf.float32,
    time_major=self.time_major,
    parallel_iterations=self.parallel_iterations,
    swap_memory=True
)
```

图 4.3 编码器 RNN 网络代码示例

1.2 构建解码器

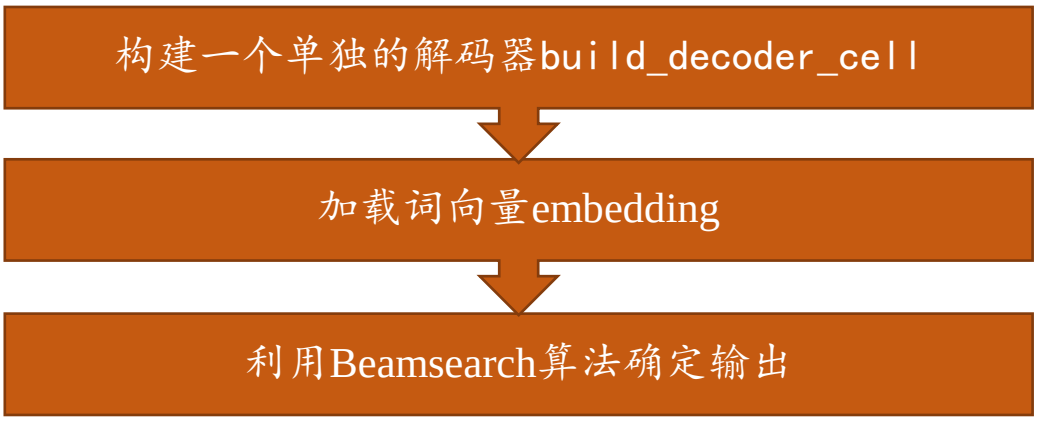

图 4.4 解码器构造思路

```
else:
    # Beamsearch is used to approximately
    # find the most likely translation
    # print("building beamsearch decoder..")
    inference_decoder = BeamSearchDecoder(
        cell=self.decoder_cell,
        embedding=embed_and_input_proj,
        start_tokens=start_tokens,
        end_token=end_token,
        initial_state=self.decoder_initial_state,
        beam_width=self.beam_width,
        output_layer=self.decoder_output_projection,
    )
```

图 4.5 Beamsearch 算法示例

1.3 构建优化器（只在训练时构建）

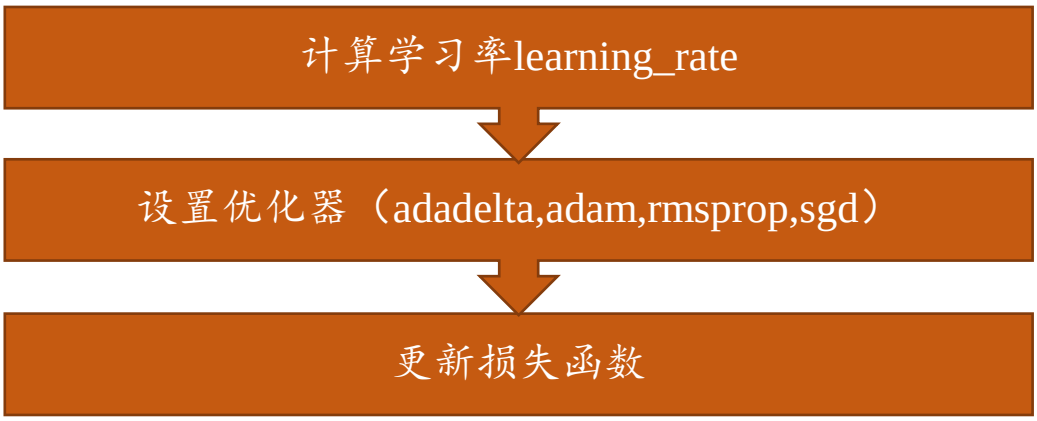

图 4.6 优化器构造思路

```
# 设置优化器,合法的优化器如下
# 'adadelta', 'adam', 'rmsprop', 'momentum', 'sgd'
trainable_params = tf.trainable_variables()
if self.optimizer.lower() == 'adadelta':
    self.opt = tf.train.AdadeltaOptimizer(
        learning_rate=learning_rate)
elif self.optimizer.lower() == 'adam':
    self.opt = tf.train.AdamOptimizer(
        learning_rate=learning_rate)
elif self.optimizer.lower() == 'rmsprop':
    self.opt = tf.train.RMSPropOptimizer(
        learning_rate=learning_rate)
elif self.optimizer.lower() == 'momentum':
    self.opt = tf.train.MomentumOptimizer(
        learning_rate=learning_rate, momentum=0.9)
elif self.optimizer.lower() == 'sgd':
    self.opt = tf.train.GradientDescentOptimizer(
        learning_rate=learning_rate)
```

图 4.6 优化器算法示例

2. 模型语料文本处理

在 /Chatbot-Fannie/Chatbot/Generate-Chatbot/seq2seq/code_seq2seq_char/preprocess_char.py 实现

对文本进行去除无效字符的正则化处理，以及把文本构建成词序列并创建成模型可以识别的文本形式。

3. 模型训练

在 /Chatbot-Fannie/Chatbot/Generate-Chatbot/seq2seq/code_seq2seq_char/train_char.py 实现

```
for epoch in range(1, n_epoch + 1):
    costs = []
    bar = tqdm(range(steps), total=steps,
               desc='epoch {}, loss=0.000000'.format(epoch))
    for _ in bar:
        x, xl, y, yl = next(flow)
        x = np.flip(x, axis=1)

        add_loss = model.train(sess, dummy_encoder_inputs, dummy_encoder_inputs_lengths, y, yl, loss_only=True)
        add_loss *= -0.5
        cost, lr = model.train(sess, x, xl, y, yl,return_lr=True, add_loss=add_loss)
        costs.append(cost)
        bar.set_description('epoch {} loss={:.6f} lr={:.6f}'.format(epoch,np.mean(costs),lr))

    model.save(sess, save_path)

flow.close()
```

图 4.7 训练部分代码示例

n_epoch ： 设置训练层数为 10

batch_size：设置 batch_size 大小为 128

steps：设置步长

4. 模型预测

在 /Chatbot-Fannie/Chatbot/Generate-Chatbot/seq2seq/code_seq2seq_char/predict _char.py 实现

```
with tf.Session(config=config) as sess:
    sess.run(init)
    model_pred.load(sess, save_path)

    while True:
        user_text = input('YOU SAY:')
        if user_text in ('exit', 'quit'):
            exit(0)
        x_test = [list(user_text.lower())]
        bar = batch_flow([x_test], ws, 1)
        x, xl = next(bar)
        x = np.flip(x, axis=1)

        print(x, xl)
        pred = model_pred.predict( sess,np.array(x),np.array(xl) )
        print(pred)
        print(ws.inverse_transform(x[0]))
        for p in pred:
            ans = ws.inverse_transform(p)
            print(ans)
```

图 4.8 预测部分代码示例

将预处理过的文本语料带入模型中训练，观察结果。

5. 结果展示

5.1 模型训练结果展示 ：

```
epoch 1 loss=1.483715 lr=0.000995: 100%|██████████| 5348/5348 [2:38:27<00:00,  1.27s/it]
epoch 2 loss=1.120372 lr=0.000989: 100%|██████████| 5348/5348 [1:48:24<00:00,  1.14s/it]
epoch 3 loss=1.026043 lr=0.000984: 100%|██████████| 5348/5348 [1:44:21<00:00,  1.14s/it]
epoch 4 loss=0.966093 lr=0.000978: 100%|██████████| 5348/5348 [1:41:21<00:00,  1.17s/it]
epoch 5 loss=0.927935 lr=0.000973: 100%|██████████| 5348/5348 [1:44:36<00:00,  1.17s/it]
epoch 6 loss=0.899570 lr=0.000967: 100%|██████████| 5348/5348 [1:44:35<00:00,  1.15s/it]
epoch 7 loss=0.877637 lr=0.000962: 100%|██████████| 5348/5348 [1:42:17<00:00,  1.23s/it]
epoch 8 loss=0.854844 lr=0.000956: 100%|██████████| 5348/5348 [1:38:56<00:00,  1.14s/it]
epoch 9 loss=0.834742 lr=0.000951: 100%|██████████| 5348/5348 [1:39:02<00:00,  1.08s/it]
epoch 10 loss=0.816646 lr=0.000945: 100%|██████████| 5348/5348 [1:39:06<00:00,  1.10s/it]
```

图 4.9 模型训练结果展示

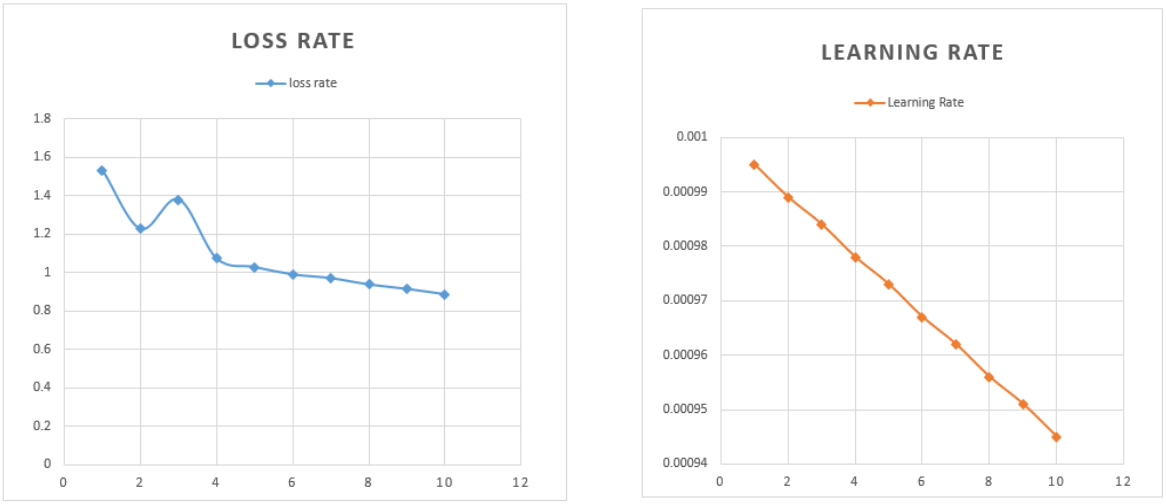

图 4.10 损失率、学习率变化趋势图

由上图可知，目标函数的损失率随着学习率的改变呈现**滑梯式**的下降，基本符合正常损失率变化的要求，没有出现断崖式的下降

或者是过于平缓的下降，说明本次实验参数的选择较为合理。

5.2 聊天机器人结果展示

```
Input Chat Sentence:你好
[[   3 1326  564]] [3]
[[1859 2157 2342 2342 1045 3461  937 2093 1990    3]]
['</s>', '好', '你']
['我', '是', '棒', '棒', '哒', '聚', '合', '数', '据', '</s>']
Input Chat Sentence:今天天气怎么样
[[   3 2306  434 1752 2448 1295 1295  498]] [8]
[[ 29 290  29   3]]
['</s>', '样', '么', '怎', '气', '天', '天', '今']
['=', '。', '=', '</s>']
Input Chat Sentence:你在说什么
[[   3  434  492 3970 1208  564]] [6]
[[1859 2157  564 3018    3]]
['</s>', '么', '什', '说', '在', '你']
['我', '是', '你', '的', '</s>']
```

图 4.11 生成式机器人训练结果

(三) 基于检索的聊天机器人模型

在 /Chatbot-Fannie/Chatbot/Search-Chatbot/seq2seq/chatbot_fuzzy.py 实现

1. 数据预处理.

文本进行**去除无效字符（标点符号、空格等）、结巴分词、加载**

词向量等处理。

2. 模型搭建

fuzzywuzy 是一个简单易用的模糊字符串匹配工具包。它依据 Levenshtein Distance 算法计算两个序列之间的差异。Levenshtein Distance 算法，又叫 Edit Distance 算法，是指两个字符串之间，由一个转成另一个所需的最少编辑操作次数。许可的编辑操作包括将一个字符替换成另一个字符，插入一个字符，删除一个字符。一般来说，编辑距离越小，两个串的相似度越大。

```
def fuzzy_fuzzywuzzy_list(fuzz, user_input, qa_list, collection, topn=50):
    '''编辑距离，速度比较慢，比起匹配方法，能够处理字符不一样的问题'''

    user_input_set = [user_input_one for user_input_one in user_input]

    same_char_list = []
    max_data = 0
    max_data_list = []
    count_collection_new_one = 0
    for collection_new_one in collection: # 获取相同字符串多的问题
        count_same_char_one = len([x for x in user_input_set if x in collection_new_one])

        if count_same_char_one > 0:
            same_char_list.append((count_collection_new_one, count_same_char_one))
        if count_same_char_one > max_data:
            max_data_list.append(count_same_char_one)
            max_data = count_same_char_one
        count_collection_new_one += 1

    list_max_count = []
    len_max_data_list = len(max_data_list)
    for x in range(len_max_data_list):  # 获取前20排名
        for k,l in same_char_list:
            if l == max_data_list[len_max_data_list -1 - x]:
                list_max_count.append(qa_list[k]) #问答重这里取出来
        if len(list_max_count) >= 5000:
            list_max_count = list_max_count[0:5000]
            break

    result =  process.extract(user_input, list_max_count, scorer=fuzz.token_set_ratio, limit=topn)
    return result
```

图 4.12 计算相似度代码示例

3. 结果展示

```
推荐大学
菜菜 SAY ： 还有什么比BCNU更美的学校吗
推荐结果:
[('推荐大学\t还有什么比BCNU更美的学校吗\n', 100), ('大学\t==\n', 67), ('大学\t=。=\n', 67), ('云南大学\t==\n', 50), ('大学物理\t=。=\n', 50)]
YOU SAY ：
BCNU是什么
菜菜 SAY ： 中国东方正常大学你都没听说过吗
推荐结果:
[('BCNU是什么\t中国东方正常大学你都没听说过吗\n', 100)]
YOU SAY ：
你喜欢数据科学吗
菜菜 SAY ： 喜欢是喜欢 就是有点头冷
推荐结果:
[('你喜欢数据科学吗\t喜欢是喜欢 就是有点头冷\n', 100), ('喜欢数学\t=。=\n', 67), ('你喜欢我吗\t你喜欢我吗\n', 62), ('你喜欢我吗\t==\n', 62), ('你喜欢我吗\t) \n', 62)]
YOU SAY ：
嘻嘻
菜菜 SAY ： 我喜欢你的笑容
```

图 4.13 检索式机器人实验代码示例

五、总结与体会

1. 整个实验主要通过理论学习神经网络相关知识，了解原理机制，在此基础上通过调用 tensorflow 库函数以直接实现相关算法，以辅助对理论学习的理解；

2. 观察结果发现，生成式机器人的实现效果不是很理想，回答出的语料缺乏逻辑性和准确性，思考原因之一可能为训练的文本语料不够丰富，带有偏向性；同时，中文分词精确度仍需提高也是原因之一；基于以上原因，生成式机器人适用于闲聊机器人场景之下，即对回复语料无准确度的高要求；

3. 而对于基于检索的机器人而言，回复语料是根据问答库进行相

似度的匹配搜索得出，精确度明显高于生成式机器人，适用于特定场合下（如医疗、金融、银行等）的智能客服机器人问答；同时，因为该模型高度依赖问答库，回答不具有创新性；

4. 对模型的优化

考虑可将两种模型相结合，利用相似度计算出回答的文本，作为生成式模型的输入语料，从而提高语料文本的质量，提高模型的准确度。