实验报告

基于深度学习的中英机器翻译
17341046 郭梓煜

环境配置

- python3.6
- vscode
- nltk，jieba
- TensorFlow1.3.0，Keras
- coding: utf-8

实验要求

1. 模型要求：
   - 两个LSTM分别作为Encoder和Decoder
   - 实现基于注意力机制的机器翻译
   - 自行选择分词工具

- 改变teacher forcing ratio，观察效果
- Beam Search策略

2. 评估指标：BLEU值（BLEU-4）
3. 词向量：随机初始化或自选预训练词向量
4. 设备：CPU/GPU
5. 数据集规模：10000
   10K版本，train: 8000, test: 1000, dev: 1000
   100K版本（非必须），train: 80000, test: 10000, dev: 10000
   可根据提供的数据处理脚本自定义数据集大小（不能太小）
   数据格式： source: 每行一条中文句子
   target: 每行一条source中对应行数的英文句子

实验过程

数据预处理

1. 定义特殊符号
   定义一些特殊符号。其中“<pad>”加在较短序列后，直到同一batch内每个样本序列等长。而“<bos>”和“<eos>”符号分别表示序列的开始和结束，要求每个句子开头为“<bos>”，结尾为“<eos>”
2. 分词
   对语料集内的句子进行分词，可以选择根据空格分词或者使用Spacy，NLTK等工具进行分词
3. 创建词典
   根据上述分词结果分别为源语言和目标语言创建词典；源语言单词的索引和目标语言单词的索引相互独立

- 根据文件路径，读取文档
  - 使用nltk，jieba分词
  - 在句子开头插入<bos>，结尾插入<bos>

```
def get_target_sentense(file_path):

target_sentense = []
f = open(file_path, 'r', encoding='utf-8')
for sentense in f.readlines():
    sentense = sentense.strip()
    if len(sentense) == 0:
        continue
    temp = nltk.word_tokenize(sentense.lower())
    # 用nltk进行分词，单词转换成小写
    temp.insert(0, '<bos>')  # 开头插入<bos>
    temp.append('<eos>')  # 结尾插入<eos>
```

```
        target_sentense.append(temp)
    return target_sentense
```

```
def get_score_sentense(file_path):

    score_sentense = []
    f = open(file_path, 'r', encoding='utf-8')
    for sentense in f.readlines():
        sentense = sentense.strip()
        word_list = jieba.cut(sentense, cut_all=False)  # 用jieba分词
        sentense = ' '.join(word_list)
        temp_list = sentense.split()
        temp_list.insert(0, '<bos>')  # 在句子开头插入<bos>
        temp_list.append('<eos>')  # 在句子结尾插入<eos>
        score_sentense.append(temp_list)
    return score_sentense
```

- 创建词典
  - 将单词转为数字

```
def word2num(word_to_num, sentense_list):

    num_sentense = []
    for sentense in sentense_list:
        num_sentense = []
        for word in sentense:
            vocab_num = word_to_num[word]
            num_sentense += [vocab_num]
        num_sentense += [num_sentense]
    return num_sentense
```

  - 排序得到字典

```
def build_dict(data):

    counter = collections.Counter(data)
    count_pairs = sorted(counter.items(), key=lambda x: (-x[1], x[0]))
    words, _ = list(zip(*count_pairs))
    word_to_num = dict(zip(words, range(len(words))))
    return word_to_num
```

模型

Encoder

1. 根据源语言词典大小设置word embedding矩阵；用预训练词向量初始化

```
def _init_embedding(self):
    self.embedding_encoder = tf.Variable(tf.random_uniform(
        [self.source_vocab_size,
         self.embedding_size]))
    self.encoder_embedding_inputs = tf.nn.embedding_lookup(
        self.embedding_encoder,
        self.encoder_inputs)

    self.embedding_decoder = tf.Variable(tf.random_uniform(
        [self.target_vocab_size,
         self.embedding_size]))

    self.decoder_embedding_inputs = tf.nn.embedding_lookup(
        self.embedding_decoder,
        self.decoder_targets)
```

2. Encoder使用双向LSTM;

3. Encoder的初始隐藏状态h0选择全零或者随机向量；源句子的每个单词的embedding作为Encoder的相应时间步输入;

4. Encoder返回output向量，其维度大小为 [src_legth, batch_size, hid_dim*num_directions];
   *这里可以将（hid_dim*num_directions）看成是前向、后向隐藏状态的拼接; 该向量的第一维中第i个分量作为每个batch下源句子的第i时间步的隐藏状态。*

```
def _init_encoder(self):
    def make_cell(rnn_size):
        enc_cell = tf.nn.rnn_cell.BasicLSTMCell(rnn_size)
        return enc_cell

    with tf.variable_scope("Encoder") as scope:
        num_layers = self.num_layers
        encoder_cell_fw = make_cell(self.encoder_num_units)
        encoder_cell_bw = make_cell(self.encoder_num_units)
        #state 是每一层最后一个step的输出
        enc_outputs, enc_state = tf.nn.bidirectional_dynamic_rnn(
            encoder_cell_fw,
            encoder_cell_bw,
            inputs=self.encoder_embedding_inputs,
            sequence_length=self.encoder_inputs_length,
            dtype=tf.float32,
        )
        self.encoder_outputs = tf.concat([enc_outputs[0],enc_outputs[1]],
```

```
-1)
        self.encoder_state = enc_state
```

Decoder

1. Decoder使用单向LSTM或者单向GRU，根据目标语言词典大小设置word embedding矩阵W_i，使用预训练词向量初始化;

2. 由于希望Decoder能尽量获取encoder的编码信息，所以选择Encoder的最后一个隐藏状态（双向）ht作为Decoder的初始隐藏状态s0;

3. 训练时候，Decoder的输入有如下两种方式:
   a. Teacher Forcing:直接使用训练数据的标准答案(ground truth)的对应上一项作为当前时间步的输入;
   b. Curriculum Learning:使用一个概率p,随机决定选择使用ground truth还是前一个时间步模型生成的预测，来作为当前时间步的输入。 4.不使用attention机制的情况下，可以直接将RNN每个时间步下的隐藏状态ht，经过全连接层后输出
   $$logit=W_o h_t$$
   W_o：全连接层，将每个时间步下的隐藏状态" 转化"为维度 V 的" 输出向量logit，V是目标语言的词典大小；输出向量logit可看作为有关各个输出词的预测概率，维度为(batch_size, V)；这里的W_o可以设置为W_i 的转置矩阵，或者独立的矩阵。

```
def _init_decoder(self):
    def make_cell(rnn_size):
        dec_cell = tf.nn.rnn_cell.BasicLSTMCell(rnn_size)
        return dec_cell

    def create_decoder_cell():
        cell =
tf.contrib.rnn.MultiRNNCell([make_cell(self.decoder_num_units) for _ in
range(self.num_layers)])

        if self.beam_search and self.mode == "Infer":
            dec_start_state = seq2seq.tile_batch(self.encoder_state,
self.beam_width)
            #将encoder_state复制beam_width份
            enc_outputs = seq2seq.tile_batch(self.encoder_outputs,
self.beam_width)
            #将encoder_outputs复制beam_width份
            enc_lengths = seq2seq.tile_batch(self.encoder_inputs_length,
self.beam_width)
            #将encoder_inputs_length复制beam_width份
        else:
            dec_start_state = self.encoder_state #encoder最后一个隐藏状态作为
decoder的初始状态
            enc_outputs = self.encoder_outputs
            enc_lengths = self.encoder_inputs_length
```

```
        if self.attention:
            attention_states = enc_outputs

            attention_mechanism = tf.contrib.seq2seq.LuongAttention(
                self.decoder_num_units,
                attention_states,
                memory_sequence_length=enc_lengths)

            decoder_cell = tf.contrib.seq2seq.AttentionWrapper(
                cell,
                attention_mechanism,
                attention_layer_size=self.decoder_num_units)

            if self.beam_search and self.mode == "Infer":
                initial_state = decoder_cell.zero_state(self.batch_size *
self.beam_width, tf.float32)
            else:
                initial_state = decoder_cell.zero_state(self.batch_size,
tf.float32)

            initial_state = initial_state.clone(cell_state=dec_start_state)
        else:
            initial_state = dec_start_state

        return decoder_cell, initial_state

    with tf.variable_scope("Decoder") as scope:
        projection_layer = layers_core.Dense(units=self.target_vocab_size,
use_bias=False)

        decoder_cell, initial_state = create_decoder_cell()

        if self.mode == "Train":
            training_helper = tf.contrib.seq2seq.TrainingHelper(
                self.decoder_embedding_inputs,
                self.decoder_train_length)

            training_decoder = tf.contrib.seq2seq.BasicDecoder(
                cell=decoder_cell,
                helper=training_helper,
                initial_state=initial_state,
                output_layer=projection_layer)

            (self.decoder_outputs_train,
             self.decoder_state_train,
             final_sequence_length) = tf.contrib.seq2seq.dynamic_decode(
                decoder=training_decoder,
                impute_finished=True,
                scope=scope
            )

            self.decoder_logits_train =
self.decoder_outputs_train.rnn_output
            decoder_predictions_train =
```

```
tf.argmax(self.decoder_logits_train, axis=-1)
            self.decoder_predictions_train =
tf.identity(decoder_predictions_train)

        elif self.mode == "Infer":
            start_tokens = tf.tile(tf.constant([2], dtype=tf.int32),
[self.batch_size])

            if self.beam_search == True:
                inference_decoder = tf.contrib.seq2seq.BeamSearchDecoder(
                    cell=decoder_cell,
                    embedding=self.embedding_decoder,
                    start_tokens=tf.ones_like(self.encoder_inputs_length) *
tf.constant(2, dtype=tf.int32), #decoder第一个编码器的输入
                    #start_tokens = start_tokens,
                    end_token=tf.constant(0, dtype=tf.int32), #终止符
                    initial_state=initial_state,
                    beam_width=self.beam_width,
                    output_layer=projection_layer)

                self.decoder_outputs_inference, __, ___ =
tf.contrib.seq2seq.dynamic_decode(
                    decoder=inference_decoder,

maximum_iterations=tf.round(tf.reduce_max(self.encoder_inputs_length)),
                    #maximum_iterations = 30,
                    impute_finished=False,
                    scope=scope)

                self.decoder_predictions_inference =
tf.identity(self.decoder_outputs_inference.predicted_ids)

            else:
                inference_helper =
tf.contrib.seq2seq.GreedyEmbeddingHelper(
                    self.embedding_decoder,
                    start_tokens=start_tokens,
                    end_token=0)  # EOS id

                inference_decoder = tf.contrib.seq2seq.BasicDecoder(
                    cell=decoder_cell,
                    helper=inference_helper,
                    initial_state=initial_state,
                    output_layer=projection_layer)

                self.decoder_outputs_inference, __, ___ =
tf.contrib.seq2seq.dynamic_decode(
                    decoder=inference_decoder,

maximum_iterations=tf.round(tf.reduce_max(self.encoder_inputs_length)) * 2,
                    impute_finished=False,
                    scope=scope)
```

```
                    self.decoder_predictions_inference =
tf.identity(self.decoder_outputs_inference.sample_id)
```

Attention Module

注意力计算发生在解码步骤中的每一步，它包含下列步骤：

1. 当前目标隐状态（target hidden state）和所有源状态（source hidden state）进行比较，以计算注意力权重(attention weight)。
2. 基于注意力权重，我们计算了一个背景向量(context vector)，作为源状态的加权均值。
3. 将背景向量与当前目标隐蔽态进行结合。

```
if self.attention:
                attention_states = enc_outputs

                attention_mechanism = tf.contrib.seq2seq.LuongAttention(
                    self.decoder_num_units,
                    attention_states,
                    memory_sequence_length=enc_lengths)

                decoder_cell = tf.contrib.seq2seq.AttentionWrapper(
                    cell,
                    attention_mechanism,
                    attention_layer_size=self.decoder_num_units)

                if self.beam_search and self.mode == "Infer":
                    initial_state = decoder_cell.zero_state(self.batch_size
* self.beam_width, tf.float32)
                else:
                    initial_state =
decoder_cell.zero_state(self.batch_size, tf.float32)

                initial_state =
initial_state.clone(cell_state=dec_start_state)
            else:
                initial_state = dec_start_state
```

```
# 注意力权重制图函数
def plot_attention(attention, sentence, predicted_sentence):
    fig = plt.figure(figsize=(10,10))
    ax = fig.add_subplot(1, 1, 1)
    ax.matshow(attention, cmap='viridis')

    fontdict = {'fontsize': 14}

    ax.set_xticklabels(['']  + sentence, fontdict=fontdict, rotation=90)
    ax.set_yticklabels(['']  + predicted_sentence, fontdict=fontdict)
```

```
    ax.xaxis.set_major_locator(ticker.MultipleLocator(1))
    ax.yaxis.set_major_locator(ticker.MultipleLocator(1))

    plt.show()
```

训练

1. 对Decoder的每个时间步，由label和上述logit向量，计算交叉熵作为损失;
2. 由于“<pad>”是用于填充的人为添加符号，所以不能对其计算损失，因此需要使用"掩码"避免填充项对句子总体损失函数计算的影响;
3. 计算梯度，并将其应用于优化器和反向传播;
4. 训练多个epoch，根据在验证集的表现选择最佳模型用作测试。

```
def train():
    step = 0
    g = tf.Graph()
    with g.as_default():
        model = Seq2SeqModel(
            encoder_num_units=256,
            decoder_num_units=256,
            embedding_size=256,
            num_layers=2,
            source_vocab_size=source_vocab_length,
            target_vocab_size = len(target_id2word),
            batch_size=batch_size,
            attention=True,
            beam_search=True,
            beam_width=beam_width,
            mode="Train"
        )

        with tf.Session(config=tf.ConfigProto()) as sess:
            sess.run(tf.global_variables_initializer())
            saver = tf.train.Saver()
            #summary_writer = tf.summary.FileWriter('../log',
graph=sess.graph)

            for _epoch in range(1, batches_in_epoch + 1):
                for _batch in range(max_batches):
                    X, y = my_utils.input_generator(
                        train_source_id,
                        train_target_id,
                        batch_size)
                    feed_dict = model.make_train_inputs(
                        x=X,
                        y=y)
                    _, l, train_samples, summary_str,__ = sess.run(
                        [model.train_op,
```

```
                        model.loss,
                        model.decoder_predictions_train,
                        model.summary_op,
                        model.decoder_logits_train],
                       feed_dict)
                #summary_writer.add_summary(summary_str, _epoch * _batch)
                if step % 50 == 0:
                    bleu_score = cal_bleu(train_samples,y)
                    print('-' * 50)
                    print('-' * 50)
                    print('step {}'.format(step))
                    print('-' * 50)
                    print('minibatch loss: {}'.format(sess.run(model.loss,
feed_dict)))
                    print('-'*50)
                    print("Bleu Score: " + str(bleu_score))
                    print('-' * 50)
                    for i in range(5):
                        train_sentence = ''
                        for word in train_samples[i]:
                            train_sentence += target_id2word[word] + ' '
                        print('Predict Sen: ',end="")
                        print(train_sentence)
                    print('-' * 50)
                    for i in range(5):
                        result_sentence = ''
                        for word in y[i]:
                            result_sentence += target_id2word[word] + ' '
                        print('Target Sen: ',end="")
                        print(result_sentence)
                    print('-' * 50)
                    print('\n\n\n')
                step += 1

            print(_epoch, 'epoch finished')

            if _epoch % epoch_be_saved == 0:
                saver.save(sess, '../models/' + 'nmt.ckpt',
global_step=step)
                print('model saved.')

        print('finish training')
```

测试

1. 测试的时候，Decoder当前时间步输入为上一时间步的预测，第一个时间步输入为“<eos>”。模型预测出结束符“<eos>”或者到达预设的长度限制时停止生成；
2. 集束搜索Beam search: 贪心搜索只选择了概率最大的一个, 而集束搜索则选择了概率最大的前k个。这个k值也叫做集束宽度（Beam Width）。

3. 评估: 计算预测句子与ground true之间的BLEU值；BLEU是一种对生成语句进行评估的指标。完美匹配的得分为1.0，而完全不匹配则得 分为0.0。可以用来计算待评价译文和一个或多个参考译文间的距离。使用NLTK包可以计算BLEU值。

```
def cal_bleu(predict,result):
    predict = predict.tolist()
    predict_list = []
    for i in range(len(predict)):
        try:
            end_ = predict[i].index(0)
        except:
            end_ = len(predict[i]) - 1
        if end_ < len(result[i]) - 1:
            predict_list.append(predict[i][:len(result[i])])
        else:
            predict_list.append(predict[i][:end_])
    total_score = 0
    for i in range(len(predict_list)):
        predict_sen = []
        result_sen = [[]]
        for j in range(len(predict_list[i])):
            predict_sen.append(target_id2word[predict_list[i][j]])

        for j in range(len(result[i])):
            result_sen[0].append(target_id2word[result[i][j]])
        score = sentence_bleu(result_sen,predict_sen)
        total_score += score
    return total_score / len(predict)
```

实验结果

训练集:
以下为部分预测结果:

```
...
step 2300
-------------------------------------------------------------------------------
----------------------
minibatch loss: 2.5367274284362793
-------------------------------------------------------------------------------
----------------------
Bleu Score: 0.4156887843934715
predict sentence: <bos> so what are the workers unique for ? <eos> <eos> <eos>
<eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos>
<eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos>
<eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos>
```

```
<eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos>
<eos> <eos> <eos> <eos>
target sentence: <bos> so what are the europeans waiting for ? <eos>

predict sentence: <bos> why on gain activities to anticipate the devastating
prolong between nato and the eu – such as used policy toward nato under
president recent critical has been massive in the right direction ? <eos>
<eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos>
<eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos>
<eos> <eos> <eos> <eos> <eos> <eos> <eos>
target sentence: <bos> why not start now to overcome the traditional tension
between nato and the eu – especially as french policy toward nato under
president nicolas sarkozy has been moving in the right direction ? <eos>

predict sentence: <bos> a compounded mutual work of the secretary general of
nato and of the deep of eu foreign policy in the 96 of his mandate care ’ t
require much time and very . <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos>
<eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos>
<eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos> <eos>
target sentence: <bos> a regular mutual presence of the secretary general of
nato and of the head of eu foreign policy in the councils of both
organizations doesn ’ t require much time and effort . <eos>

predict sentence: <bos> why on dismay gold mainstream at a high political
level ( with the stewardship of nato representing in security relief ) – for
instance , by mobilizing the us secretary of state and other back of the
administration , such as the top secretary or the 96 of the bilateral
protection don , to anticipa
...
```

测试集：
以下为部分预测结果：
详见doc文件夹中的result.txt

```
...
990
<bos> strengthening fatalities blow puzzled perhaps perhaps perhaps perhaps
perhaps perhaps perhaps perhaps perhaps perhaps perhaps perhaps perhaps
perhaps perhaps perhaps perhaps perhaps track track by not by not by by by by
by by by
991
<bos> but tayyip , us us also also us us us us us us us us us us us us us us
us us us us us us us us us us us us us
992
<bos> the , personnel personnel s , , , , , , , , , . . . . . . . . . . . .
. . . . . . . .
993
<bos> . , puzzled gun recently an an an an an an an an an an an an an an an
an an years 2008 years 2008 years 2008 years 2008 years 2008 years 2008
994
<bos> should americas , us us us us us us us us us us us us us us us us us us
```

```
us us us us us us us us us us us us us us
995
<bos> . , puzzled personnel an an an an an an an an an an an an by by
not by by by by by by by by by by by by by
996
<bos> there , , us recently recently recently recently recently recently
recently recently recently recently recently recently recently recently
recently recently us also also also also also also also also also also also
also also also
997
<bos> there , years us we we we we we we we we we we we we we we we we we
we we we we we years we years we us years us years
998
<bos> the , the us s by by by by by by by by by by by by by by by by by by by
by by by by by by by by by by by by
999
<bos> there micro-story puzzled us years an years , years , years , years ,
years , years , us years a years a years a years a years a years a years a
years a
1000
<bos> the , , , , recently recently recently recently recently recently
recently recently recently recently recently recently recently recently
recently recently recently recently recently recently recently recently
recently recently recently recently recently recently recently recently

final sentence bleu score:  7.521496624102009e-232
```

可见，测试集预测效果并不是十分理想，这应该是由于跑的轮数太少以及网络模型的问题。另一方面也是由于时间较紧，没来得及多加揣摩，多次地跑来实现更好的效果。

总结与思考

遇到的困难及解决方法

1.新知识的学习问题

这次的实验难度挑战也比较大，较之上次大作业同样需要学习许多之前没有过的新知识。比如说，注意力机制，机器翻译，LSTM搭建等等。而这些东西往往直接上网谷歌难以解决问题，这就迫使我和以往学习不一样地去阅读官方文档或者阅读网上的教程，尽管官方文档较为晦涩，也没有具体的解析，是块难啃的硬骨头，但是还是考验了我的耐心，通过和同学交流以及共同学习，查阅相关文献，解决了新知识的难题。网上的教程也比较关键，我重点参考了这个的实现。

2. TensorFlow使用问题

TensorFlow-GPU的运行速度会快一些，可惜的是在import tensorflow的时候出现了问题，could not find cuda xxx，经查阅相关错误，发现是由于显卡并非英伟达的，而使用不了

TensorFlow-GPU，只能使用TensorFlow-CPU了，导致后来训练模型的时候，一次epoch就要二三十分钟，大大限制训练次数。这个问题也导致了最后模型的效果较差。

3. python用法问题
这一次实验让我对python的一些函数应用地更加自如了。一些同学需要十几行的代码我可以一行实现，相比上次的分词去噪这次的代码写的更加简洁了运用了许多不一样的库和函数，所以说，在觉得这样实现较为复杂的时候，可以先谷歌一下看看有没有更方便的库或者函数可以使用。

心得体会

经过了这一次的实验，对机器翻译的具体实现，模型的构建和对文本的处理都有深入的了解。尽管实验中多多少少存在一些不足诸如硬件上的限制等等，也是学到了许多，在以后的学习中会继续努力，更上一层楼。