



湖南大學

HUNAN UNIVERSITY

人工智能課程設計二報告
中英翻譯模型

目录

一、 实验目的	3
二、 实验描述	3
三、 实验及结果分析	4
(1) 开发语言及运行环境;	4
(2) 实验的具体步骤;	4
① 数据预处理	5
② 训练	8
③ 评价	12
四、 心得	14
① 警告	14
② 下载 nltk 的 punkt 和 jieba	15
③ 远程环境的问题	15
④ 一些版本兼容问题	15
⑤ 内存优化问题	16
五、 程序文件名清单	17

一、实验目的

建议使用python编程实现，并在Mindspore框架下实现。

二、实验描述

序列到序列（Seq2Seq）学习框架通常用于处理序列数据，如自然语言处理中的机器翻译、文本摘要、语音识别等任务。这种框架涉及到两个序列：输入序列和输出序列。

基本思想：

1. 核心思想

Seq2Seq学习框架的核心思想是通过神经网络学习输入序列到输出序列的映射关系。这通常涉及到编码器-解码器（Encoder-Decoder）结构，其中编码器处理输入序列并生成一个固定长度的上下文向量，解码器则基于这个上下文向量生成输出序列。

2. 网络结构

编码器（Encoder）：通常是一个循环神经网络（RNN），如LSTM（长短期记忆网络）或GRU（门控循环单元），它处理输入序列并捕捉其特征。

解码器（Decoder）：也是一个RNN，它使用编码器的输出（上下文向量）来生成输出序列。解码器在生成每个输出元素时，还会考虑之前生成的元素。

3. 前向传播

输入序列通过编码器处理，生成上下文向量。

解码器使用上下文向量和先前生成的输出序列来生成新的输出序列。

4. 损失函数

通常使用序列生成任务的损失函数，如交叉熵损失，计算预测序列和真实序列之间的差异。

5. 反向传播

通过时间反向传播（Backpropagation Through Time, BPTT）算法计算损失函数关于网络权重的梯度。

6. 权重更新

使用梯度下降或其变体来更新网络权重。

7. 训练过程

训练过程包括前向传播、计算损失、反向传播和权重更新，重复进行直到满足停止条件。

8. 特点

Seq2Seq模型能够处理变长的输入和输出序列，适用于复杂的序列转换任务。

需要处理序列数据的特殊问题，如梯度消失或梯度爆炸。

可以使用注意力机制（Attention Mechanism）来增强模型性能，允许模型在生成输出时关注输入序列的不同部分。

三、实验及结果分析

（1）开发语言及运行环境；

与实验一相同，不再赘述。

（2）实验的具体步骤；

任务说明：

（1）基于序列到序列（Seq2Seq）学习框架，在 MindSpore 等框架下设计并训练一个中英文机器翻译模型，完成中译英和英译中翻译任务。具体模型选择可以参考如 LSTM，GRU，Transformer 等，但不做限制；

（2）实验数据集为 WMT18 新闻评论数据集 [News Commentary v13](#)，整个语料库分训练集（约 252,700 条）、验证集和测试集（分别约 2,000 条）三部分，每部分包含中英文平行语料两个文件，全部语料已预分词处理；

（3）根据指定的评价指标和测试集数据评价模型性能。

评价指标：

（1）BLEU1/2/3:

BLEU (Bilingual Evaluation Understudy) 计算同时出现在系统译文和参考译文中的 n 元词重叠程度，实验要求计算 $n=1/2/3$ 。

（2）Perplexity:

Perplexity 是衡量语言模型生成句子能力的评价指标，计算 perplexity 的公式如下：

$$\text{perplexity}(S) = p(w_1, w_2, w_3, \dots, w_m)^{-\frac{1}{m}} = \sqrt[m]{\prod_{i=1}^m \frac{1}{p(w_i | w_1, w_2, w_3, \dots, w_{i-1})}}$$

其中：

S ，参考译文

w_i ，参考译文中的第 i 个单词

m ，参考译文句长，即译文中包含的单词数

① 数据预处理

文件 `create_vocab.py` 和 `process_data.py` 用于数据预处理，处理过后的 `tok` 文件输出在源数据同源目录中，`MindRecord` 文件放在 `nltk_mindreord` 中。

不知道实验说明里提到的已经预分词处理是不是想说已经把一句话分成了一段，但还是要进行处理的，用的是 `jieba` 分词。处理过后是这样：

test.en

test.zh

train.en

train.zh

valid.en

valid.zh

->

all.en.tok

all.zh.tok

test.en

test.en.tok

test.zh

test.zh.tok

train.en

train.en.tok

train.zh

train.zh.tok

valid.en

valid.en.tok

valid.zh

valid.zh.tok

vocab.en

vocab.zh

7 hours ago

7 hours ago

2 years ago

7 hours ago

2 years ago

7 hours ago

2 years ago

7 hours ago

2 years ago

7 hours ago

2 years ago

7 hours ago

2 years ago

7 hours ago

7 hours ago

7 hours ago

File: mindrecord_128, Size: 48.03 MB

File: mindrecord_192, Size: 48.00 MB

File: mindrecord_256, Size: 48.00 MB

File: mindrecord_64.db, Size: 9.28 MB

File: mindrecord_128.db, Size: 0.10 MB

File: mindrecord_192.db, Size: 0.02 MB

File: mindrecord_64, Size: 129.25 MB

File: mindrecord_256.db, Size: 0.02 MB

mindrecord_128

mindrecord_128.db

mindrecord_192

mindrecord_192.db

mindrecord_256

mindrecord_256.db

mindrecord_64

mindrecord_64.db

3 days ago

3 days ago

3 days ago

3 days ago

3 days ago

3 days ago

3 days ago

3 days ago

vocab.zh

vocab.en

1 <unk>

2 <pad>

3 <sos>

4 <eos>

5 的

6 ,

7 。

8 在

9 和

10 -

11 是

12 了

13 、

14 也

15 但

16 对

17 美国

18 (

19 而

20)

21 将

22 国家

23 “

24 这

25 ”

26 经济

27 中

28 年

29 一个

1 <unk>

2 <pad>

3 <sos>

4 <eos>

5 the

6 ,

7 .

8 of

9 to

10 and

11 in

12 a

13 that

14 is

15 '

16 s

17 for

18 -

19 it

20 as

21 be

22 on

23 with

24 are

25 not

26 but

27 by

28 has

29 have

```

def create_tokenized_sentences(input_files, language):
    sentences = []
    total_lines = open(input_files, "r", encoding="utf-8").read().splitlines()
    for line in total_lines:
        line = line.strip("\r\n")
        line = line.lower()
        if language == "chinese":
            tokenize_sentence = list(jieba.cut(line))
        else:
            tokenize_sentence = word_tokenize(line)
        str_sentence = " ".join(tokenize_sentence)
        sentences.append(str_sentence)

    tokenize_file = input_files + ".tok"
    f = open(tokenize_file, "w", encoding="utf-8")
    for line in sentences:
        f.write(line)
        f.write("\n")
    f.close()

```

`create_tokenized_sentences` 对输入文件中的句子进行分词处理。对于中文文本，使用 `jieba` 进行分词；对于英文文本，使用 `nltk` 的 `word_tokenize` 分词。分词后的句子会被保存到一个新的文件中，文件名以 `.tok` 结尾。

```

def get_dataset_vocab(text_file, vocab_file):
    counter = Counter()
    text_lines = open(text_file, "r", encoding="utf-8").read().splitlines()
    for line in text_lines:
        for word in line.strip("\r\n").split(" "):
            if word:
                counter[word] += 1

    vocab = open(vocab_file, "w", encoding="utf-8")
    basic_label = ["<unk>", "<pad>", "<sos>", "<eos>"]
    for label in basic_label:
        vocab.write(label + "\n")
    for key, f in sorted(counter.items(), key=lambda x: x[1], reverse=True):
        if f < 2:
            continue
        vocab.write(key + "\n")
    vocab.close()

```

`get_dataset_vocab` 函数用于从分词后的文本中生成词汇表。它统计每个词的出现频率，将标志符写入词频表、按照词语频率将频率大于等于 2 的词写入词汇表文件中。

`merge_text` 将多个分词后的文本文件合并成同一文件，便于后续处理。

```

19  def get_instance_features(instance, tokenizer_src, tokenizer_trg, max_seq_length, bucket):
20      def _find_bucket_length(source_tokens, target_tokens):
21          source_ids = tokenizer_src.convert_tokens_to_ids(source_tokens)
22          target_ids = tokenizer_trg.convert_tokens_to_ids(target_tokens)
23          num = max(len(source_ids), len(target_ids))
24          if num > bucket[-1]:
25              return None
26          for index in range(1, len(bucket)):
27              if bucket[index - 1] < num <= bucket[index]:
28                  return bucket[index]
29          return bucket[0]
30
31      def _convert_ids_and_mask(tokenizer, input_tokens, seq_max_bucket_length):
32          input_ids = tokenizer.convert_tokens_to_ids(input_tokens)
33          input_mask = [1] * len(input_ids)
34
35          while len(input_ids) < seq_max_bucket_length:
36              input_ids.append(1) # Use <pad> token id
37              input_mask.append(0)
38
39          return input_ids[:seq_max_bucket_length], input_mask[:seq_max_bucket_length]

```

get_instance_features() → _find_bucket_length() → for index in range(1, len(bucket)) → if bucket[index - 1] < num <= b...

`get_instance_features` 将训练实例转换为模型可以使用的特征。

桶策略：桶策略用于根据句子长度选择合适的序列长度，允许将长度相近的句子放在同一个桶中进行处理。**填充和掩码：**填充用于将句子长度统一到桶长度，掩码用于在模型中区分有效词和填充词。**特殊标记：**在分词转换为 ID 时，假设 `<pad>` 标记的 ID 为 1。这需要在词汇表中预先定义 `<pad>` 标记及其 ID。

```

57  def create_training_instance(source_words, target_words, max_seq_length, clip_to_max_len):
58      # Increase buffer for special tokens
59      effective_max_length = max_seq_length - 2 # Reserve space for <sos> and <eos>
60
61      if len(source_words) > effective_max_length or len(target_words) > effective_max_length:
62          if clip_to_max_len:
63              source_words = source_words[:effective_max_length]
64              target_words = target_words[:effective_max_length]
65          else:
66              return None
67
68      source_tokens = ["<sos>"] + source_words + ["<eos>"]
69      target_tokens = ["<sos>"] + target_words + ["<eos>"]
70
71      return SampleInstance(source_tokens=source_tokens, target_tokens=target_tokens)

```

`create_training_instance` 创建训练实例，根据最大序列长度对句子进行截断，并在句子的开头和结尾添加特殊标记（如 `<sos>` 和 `<eos>`）。

首先读取源语言和目标语言的文本文件，对每个句子进行分词处理。然后构

建词汇表，统计单词频率并排序，添加特殊标记。利用词汇表将分词转换为词 ID 序列，并生成掩码以区分有效单词和填充位置。根据模型要求对句子进行截断或填充，使其长度一致。最后将处理好的数据特征保存为 **MindRecord** 文件。

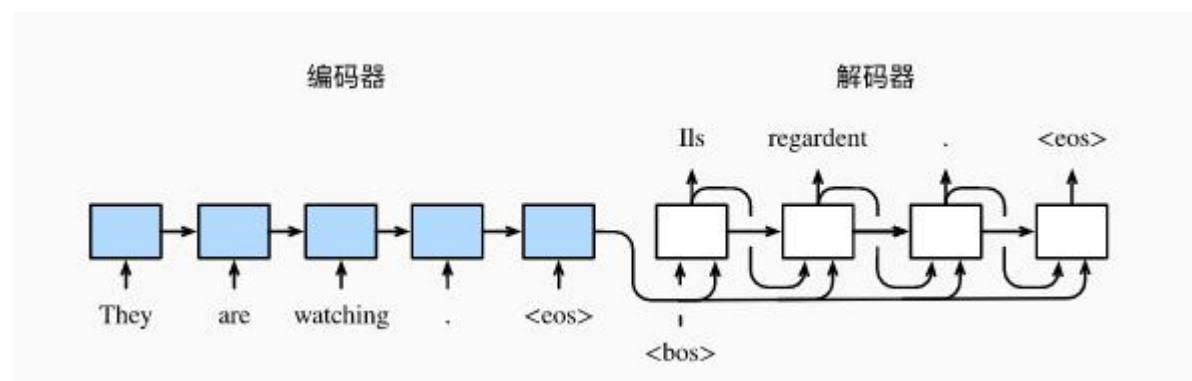
需要注意的是，在德英转换的版本中，最后输出的 **MindRecord** 文件序列长度为 32，但是中英转换中不仅更长，还有更多的长度状态，64, 128, 192, 256。这增加了后续训练模型的复杂度和训练时间。

② 训练

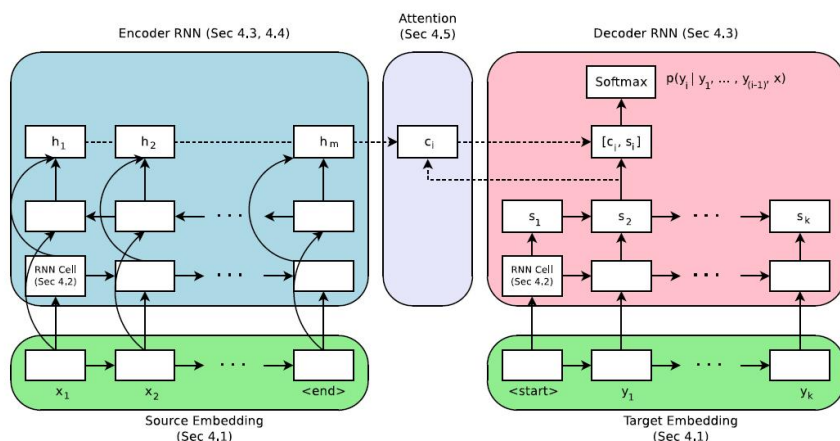
最核心的部分是 `seq2seq.py` 文件和 `gru.py`、`grn_for_train.py` 这三个，修改的时候基本也是对应修改这些文件比较多。`train.py` 指示了训练的过程，训练的时候运行该文件。

除了 `seq2seq`，训练过程还结合了 GRU、Attention 机制和 Teacher Forcing 机制。

`seq2seq` 模型由编码器(Encoder)和解码器(Decoder)两部分组成，模型的输入和输出都是一个不定长文本序列。Encoder 将原始文本序列编码，在每个时间步中输出隐藏状态，在最终时间步隐藏状态可转换为固定长度的向量，这就是上下文向量。Decoder 会将向量再次转换为目标文本序列，从而实现文本翻译的目的。Encoder 和 Decoder 一般都为 RNN 网络。

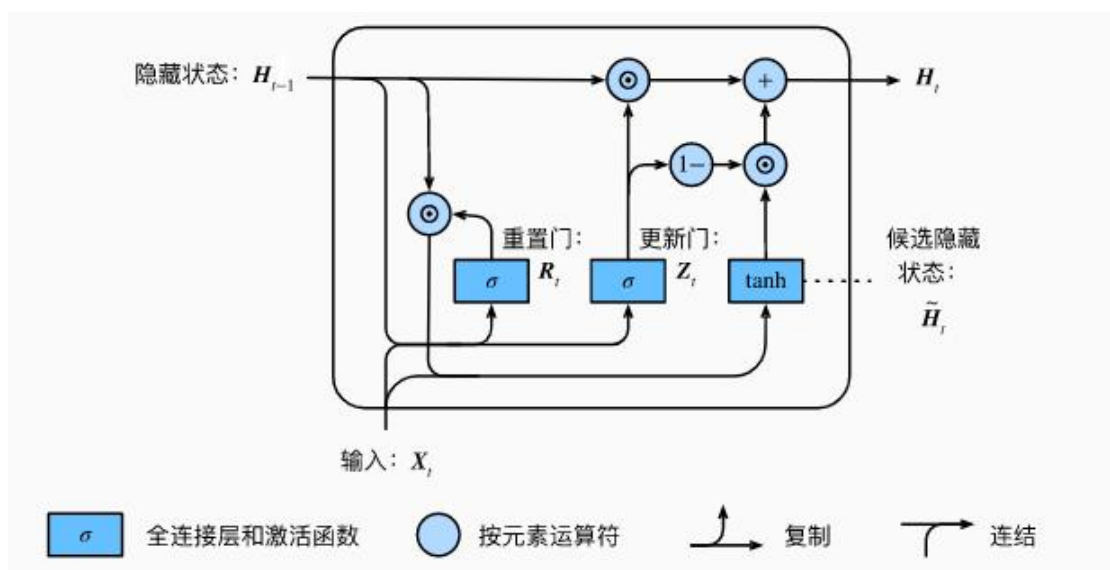


`seq2seq` 模型中，Encoder 与 Decoder 之间只有定长向量联系，这样一来会造成信息的丢失，影响模型效果。为了解决这一问题，采用注意力机制(Attention)来做动态处理。Attention 机制下，Encoder 会对每一个时间步的隐藏状态做加权平均得到上下文向量。Decoder 在每一时间步中调整注意力权重，关注输入序列中的不同部分，最后得到输出。



GRU，即门控循环单元(Gated Recurrent Unit)，在 RNN 网络中用于捕捉时间序列中时间步距离较大的依赖关系。重置门和更新门控制了 RNN 网络中隐藏状态的计算方式。两者的输入元素为当前时间步的输入与上一时间步的隐藏状态，输出通过全连接层完成。

本案例中采用了 GRU 来构成 Encoder 和 Decoder，GRU 的实现主要是通过 DynamicGRUV2 算子完成的。



RNN 后期的迭代训练效果会受到前期结果的影响，然而前期的结果通常不太精确，这样一来就会影响到整个训练进程。为了解决这一问题，引入了 Teacher Forcing 机制。Teacher Forcing 下，模型训练的输入不一定来自于上一次输出，也可能来自于训练集的真实输出。

在本案例的实现中，在数据处理环节定义了 random_teacher_force 函数来完成 Teacher Forcing 机制，并在 Decoder 时引入。

dataset.py 中 create_gru_dataset 创建一个经过预处理和批处理的数据集，用于 GRU 模型的训练或评估，添加了强制的随机教师标志。

loss.py 中计算了训练损失。

lr_schedule.py 用于生成动态学习率。

weight_init 初始化 GRU 单元和全连接层的权重和偏置。

seq2seq.py:

```
12  class Attention(nn.Cell):
13      def __init__(self):
14          super(Attention, self).__init__()
15          self.text_len = 64 # 从32改为64
16          self.attn = nn.Dense(in_channels=512 * 3,
17                               out_channels=512).to_float(mstype.float16)
18          self.fc = nn.Dense(512, 1, has_bias=False).to_float(mstype.float16)
19          self.expand_dims = P.ExpandDims()
20          self.tanh = P.Tanh()
21          self.softmax = P.Softmax()
22          self.tile = P.Tile()
23          self.transpose = P.Transpose()
24          self.concat = P.Concat(axis=2)
25          self.squeeze = P.Squeeze(axis=2)
26          self.cast = P.Cast()
27
28      def construct(self, hidden, encoder_outputs):
29          hidden = self.expand_dims(hidden, 1)
30          hidden = self.tile(hidden, (1, self.text_len, 1)) # 使用更新后的text_len
31          encoder_outputs = self.transpose(encoder_outputs, (1, 0, 2))
```

Attention 类实现注意力机制，在解码过程中动态关注编码器输出的不同部分。

输入 hidden 和 encoder_outputs，计算注意力权重。ExpandDims 和 Tile 将隐藏状态扩展到与编码器输出相同的长度。将隐藏状态与编码器输出拼接，通过全连接层和激活函数计算能量值，再通过 Softmax 计算注意力权重。

```

43 class Encoder(nn.Cell):
44     def __init__(self, is_training=True):
45         super(Encoder, self).__init__()
46         self.hidden_size = 512
47         self.vocab_size = 54359
48         self.embedding_size = 256
49         self.embedding = nn.Embedding(self.vocab_size, self.embedding_size)
50         self.rnn = BidirectionGRU(is_training=is_training).to_float(mstype.float16)
51         self.fc = nn.Dense(2 * self.hidden_size, self.hidden_size).to_float(mstype.float16)
52         self.shape = P.Shape()
53         self.transpose = P.Transpose()
54         self.p = P.Print()
55         self.cast = P.Cast()
56         self.text_len = 64 # 从32改为64
57         self.squeeze = P.Squeeze(axis=0)
58         self.tanh = P.Tanh()
59
60     def construct(self, src):
61         '''

```

Encoder 类实现编码器，将源语言句子转换为固定长度的上下文向量。

初始化编码器的参数，包括词汇表大小、嵌入层、双向 GRU 等。

输入源语言句子 `src`，通过嵌入层将词 ID 转换为词向量。使用双向 GRU 处理词向量，输出编码器的隐藏状态和输出。

这里的参数 `vocab_size` 修改成七倍大的时候我就预感到不对劲了，果然跑了两个小时什么提示都没有，下面的 Decoder 也是。

```

80 class Decoder(nn.Cell):
81     def __init__(self, is_training=True):
82         super(Decoder, self).__init__()
83         self.hidden_size = 512
84         self.vocab_size = 43527
85         self.embedding_size = 256
86         self.embedding = nn.Embedding(self.vocab_size, self.embedding_size)
87         self.rnn = GRU(is_training=is_training).to_float(mstype.float16)
88         self.text_len = 64 # 从32改为64
89         self.shape = P.Shape()
90         self.transpose = P.Transpose()
91         self.p = P.Print()
92         self.cast = P.Cast()
93         self.concat = P.Concat(axis=2)
94         self.squeeze = P.Squeeze(axis=0)
95         self.expand_dims = P.ExpandDims()
96         self.log_softmax = P.LogSoftmax(axis=1)
97         weight, bias = dense_default_state(self.embedding_size + self.hidden_size * 3, self.vocab_size)
98         self.fc = nn.Dense(self.embedding_size + self.hidden_size * 3, self.vocab_size,
99                             weight_init=weight, bias_init=bias).to_float(mstype.float16)
100         self.attention = Attention()
101         self.bmm = P.BatchMatMul()

```

Decoder 类实现解码器，将编码器的上下文向量逐步转换为目标语言句子。

初始化解码器的参数，包括词汇表大小、嵌入层、GRU 等。

输入解码器的输入 `inputs`、隐藏状态 `hidden` 和编码器输出 `encoder_outputs`。使用嵌入层将输入词 ID 转换为词向量,通过注意力机制获取加权平均的上下文向量。将词向量和上下文向量拼接,输入到 GRU 中,输出解码器的预测概率和隐藏状态。

```
140 class Seq2Seq(nn.Cell):
141     def __init__(self, is_training=True):
142         super(Seq2Seq, self).__init__()
143         if is_training:
144             self.batch_size = 8
145         else:
146             self.batch_size = 1
147         self.encoder = Encoder(is_training=is_training)
148         self.decoder = Decoder(is_training=is_training)
149         self.expanddims = P.ExpandDims()
150         self.dropout = nn.Dropout()
151         self.shape = P.Shape()
152         self.concat = P.Concat(axis=0)
153         self.argmax = P.ArgMaxWithValue(axis=1, keep_dims=True)
154         self.squeeze = P.Squeeze(axis=0)
155         self.sos = Tensor(np.ones((self.batch_size, 1)) * 2, mstype.int32)
156         self.select = P.Select()
157         self.text_len = 64 # 从32改为64
158
159     def construct(self, encoder_inputs, decoder_inputs, teacher_force):
```

Seq2Seq 类组合编码器、解码器和注意力机制,形成完整的序列到序列模型。

输入编码器输入 `encoder_inputs`、解码器输入 `decoder_inputs` 和教师强制标志 `teacher_force`。使用编码器处理源语言句子,获取编码器输出和隐藏状态。在解码过程中,逐步生成目标语言句子,使用教师强制策略在训练时选择输入。

③ 评价

(1) BLEU1/2/3:

BLEU (Bilingual Evaluation Understudy) 计算同时出现在系统译文和参考译文中的 n 元词重叠程度,实验要求计算 $n=1/2/3$ 。

(2) Perplexity:

Perplexity 是衡量语言模型生成句子能力的评价指标,计算 perplexity 的公式如下:

$$\text{perplexity}(S) = p(w_1, w_2, w_3, \dots, w_m)^{-\frac{1}{m}} = \sqrt[m]{\prod_{i=1}^m \frac{1}{p(w_i | w_1, w_2, w_3, \dots, w_{i-1})}}$$

其中:

S , 参考译文

w_i , 参考译文中的第 i 个单词

m , 参考译文句长,即译文中包含的单词数

`calculate_bleu` 用于计算 BLEU 分数, `calculate_perplexity` 用于计算困惑度。

```
532 def calculate_ngram_counts(tokens: List[str], n: int) -> Counter:
533     """计算n-gram出现次数"""
534     return Counter(tuple(tokens[i:i + n]) for i in range(len(tokens) - n + 1))
```

通过遍历 `tokens` 列表, 提取长度为 `n` 的子序列 (`n-gram`), 并使用 `Counter` 统计每个 `n-gram` 的出现次数。

```
537 def calculate_bleu(references: List[List[str]], candidates: List[List[str]], n: int) -> float:
538     """
539     计算n阶BLEU分数
540
541     Args:
542         references: 参考翻译列表
543         candidates: 候选翻译列表
544         n: n-gram的值
545
546     Returns:
547         float: BLEU分数
548     """
549     clipped_counts = 0
550     total_counts = 0
551
552     for ref, cand in zip(references, candidates):
553         ref_counts = calculate_ngram_counts(ref, n)
554         cand_counts = calculate_ngram_counts(cand, n)
555
556         for ngram, count in cand_counts.items():
557             total_counts += count
558             clipped_counts += min(count, ref_counts[ngram])
```

对于每个参考翻译和候选翻译, 计算它们的 `n-gram` 出现次数。对于每个候选翻译中的 `n-gram`, 计算其在参考翻译中的最小出现次数 (`clipped count`)。计算所有候选翻译的总 `n-gram` 出现次数和 `clipped count`, 最终的 BLEU 分数为 `clipped count` 与总 `n-gram` 出现次数的比值。

```
566 def calculate_perplexity(model, test_data: List, config: Config) -> float:
567     """
568     计算困惑度
569
570     Args:
571         model: 训练好的模型
572         test_data: 测试数据
573         config: 配置对象
574
575     Returns:
576         float: 困惑度值
577     """
578     model.set_train(False)
579     total_loss = 0
580     total_words = 0
581     criterion = nn.CrossEntropyLoss(ignore_index=config.PAD_token, reduction='sum')
582
583
584     for src_seq, tgt_seq in test_data:
585         # 转换为tensor
586         src_tensor = Tensor([src_seq], ms.int32)
587         tgt_tensor = Tensor([tgt_seq], ms.int32)
```


使用交叉熵损失函数计算模型在测试数据上的总损失，统计测试数据中非填充词的数量。将总损失除以非填充词的数量，然后取指数得到困惑度。如果非填充词的数量为零，则返回无穷大。

四、心得

[这个实验我是最后一个做的，我自己申请的代金券就剩下 100 多块了，写到这个实验的时候我时间实在是不太够了，马上还要考计网。我觉得老师可能是想让我们仿照这个来做 [MindSpore](#)，毕竟在群里分享的教程就包含这个，数据集提供的格式也差不多。但是这个其实还挺老的，是 21 年写的，我做了两步，发现他提供的教程和给的文件也对不上，我实在没时间去慢慢改了，就没再用。



另一个是写的一个完整的 mindspore 版，改了很久，但是不知道是代码的哪写错了还是性能没有优化好，训练的时候没有问题，但是测试精度非常低，最后也没有改成功。]

以上[]内的部分是之前写的实验报告部分内容。还好老师给延迟了一些时间，不然只能按照精度低的那版交上去了。延迟到 1.10，还管老师又要了一张 500 的代金券（我自己申请的那张 401 的也用完了），因为自己写的精度低的那版没有改成功，所以继续研究的前面有教程的那版代码，依然还是做了非常久。虽然精度低的那版没有改成功，我还是想把两个都交上来，毕竟前面那个也花了很长很长时间。

① 警告

LSTM总是报warning · Issue #I6BTY0 · MindSpore/mindspore ...

2023年1月30日 · 同学你好，该WARNING的含义是在constexpr修饰的函数中输入了变量，该warning的出现并不会对程序的正确性产生影响，我们将在后续版本中修改该warning，谢谢~ @洪zl

这个警告没有影响就没管。

还有一些关于版本问题的警告，但只是语法上有一点区别，不会造成影响，比如：

```
/usr/local/Ascend/ascend-toolkit/latest/python/site-packages/tbe/dsl/unify_schedule/extract_image_patches_without_cbuf_schedule.py:317: SyntaxWarning: "is not" with a literal. Did you mean "!="? if _ is not 1:
```

② 下载 nltk 的 punkt 和 jieba

因为使用远程环境，所以如果是在 github 上下载 punkt 然后自己配的话会出现问题。因为后来我使用 pip 和 nltk.download 下成功了（这个很玄学），就没再进一步研究。

③ 远程环境的问题

做这个实验的时候又发现了使用远程环境的一个新问题，使用远程环境的时候生成是远程环境中生成的，不会自动在本地更新，我尝试着手动更新，没有反应。而且传到远程环境的时候，空的文件夹不会被传输，直接运行存储到该空文件夹之类的操作会报错。

④ 一些版本兼容问题

这个教程已经是几年前写的了，使用的是 MindSpore1.2.0，初次运行的时候有很多报错需要改，有些 API 已经被弃用了。

比如：

dataset 的格式问题；

F.scalar_to_array 需要使用 ops.scalar_to_tensor 来替代；

if-else 分支的返回值需一致；

参数不匹配，16 和 32 位的 float；

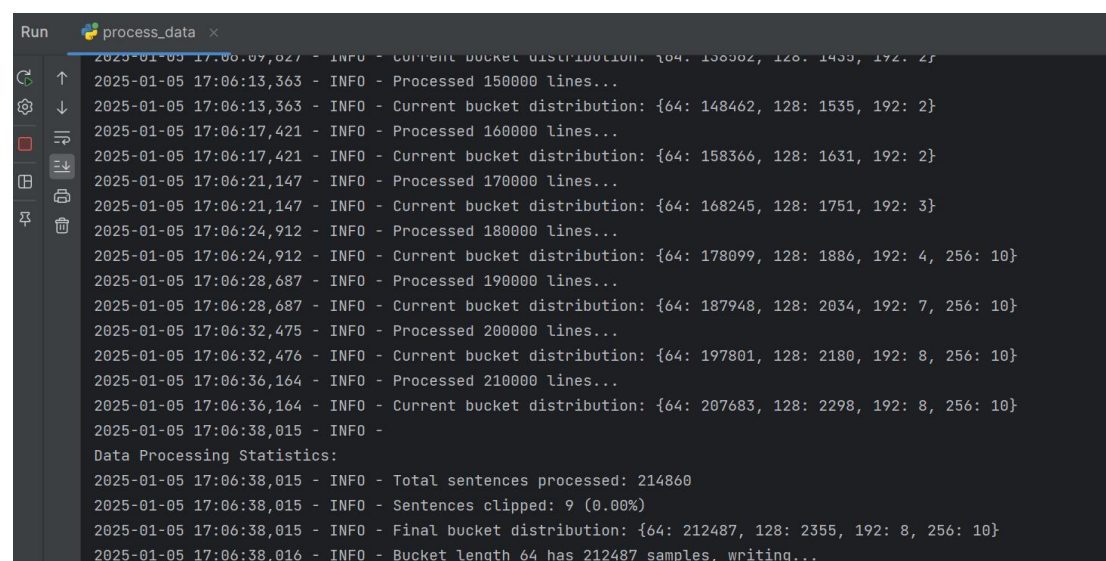
训练时 init_h 的维度不对，改了很多次一直显示张量大小不匹配。

还有设备状态的问题，设备 2 不可用，我使用的设备 0。

原教程就是德英转换的我跑成功了，改了大概两三个小时的报错就差不多，算得上顺利，训练也很快，12.5 分钟左右就能跑完。但是由于是德语和英语转换，

我想改成中英的时候会有很多问题，比如分词要用 `jieba`，还有中文的数据集输入句子会过长，原版本是 32 的，要改成 64 以上的才可以，所以 `seq2seq` 整个文件都要改，相当于改了模型的结构大小，改的时候很多报错问题，我改了很久。

在以下输出中可以看出句子长度，64 左右的居多，128 的也有。如果还是 32 的话就会有非常多的警告显示 “`long sentences ignored`”（应该是，当时做的时候忘记截图了），我觉得这样的话精度也不会好，跳过太多内容了。



```
Run process_data x
2025-01-05 17:06:07,027 - INFO - Current bucket distribution: {64: 138302, 128: 1439, 192: 21}
2025-01-05 17:06:13,363 - INFO - Processed 150000 lines...
2025-01-05 17:06:13,363 - INFO - Current bucket distribution: {64: 148462, 128: 1535, 192: 2}
2025-01-05 17:06:17,421 - INFO - Processed 160000 lines...
2025-01-05 17:06:17,421 - INFO - Current bucket distribution: {64: 158366, 128: 1631, 192: 2}
2025-01-05 17:06:21,147 - INFO - Processed 170000 lines...
2025-01-05 17:06:21,147 - INFO - Current bucket distribution: {64: 168245, 128: 1751, 192: 3}
2025-01-05 17:06:24,912 - INFO - Processed 180000 lines...
2025-01-05 17:06:24,912 - INFO - Current bucket distribution: {64: 178099, 128: 1886, 192: 4, 256: 10}
2025-01-05 17:06:28,687 - INFO - Processed 190000 lines...
2025-01-05 17:06:28,687 - INFO - Current bucket distribution: {64: 187948, 128: 2034, 192: 7, 256: 10}
2025-01-05 17:06:32,475 - INFO - Processed 200000 lines...
2025-01-05 17:06:32,475 - INFO - Current bucket distribution: {64: 197801, 128: 2180, 192: 8, 256: 10}
2025-01-05 17:06:36,164 - INFO - Processed 210000 lines...
2025-01-05 17:06:36,164 - INFO - Current bucket distribution: {64: 207683, 128: 2298, 192: 8, 256: 10}
2025-01-05 17:06:38,015 - INFO -
Data Processing Statistics:
2025-01-05 17:06:38,015 - INFO - Total sentences processed: 214860
2025-01-05 17:06:38,015 - INFO - Sentences clipped: 9 (0.00%)
2025-01-05 17:06:38,015 - INFO - Final bucket distribution: {64: 212487, 128: 2355, 192: 8, 256: 10}
2025-01-05 17:06:38,016 - INFO - Bucket length 64 has 212487 samples, writing...
```

⑤ 内存优化问题

`RuntimeError: Sync stream failed:Ascend_0`

运行很久之后出现这个问题，德英转换的时候没有出现过，可能中英转换太复杂了，而且网络结构也更复杂，`vocab_size` 的大小是原本的 7 倍，需要修改代码以优化内存和计算资源的使用。

一模一样的逻辑，原本德英的数据集很快，中英的就不行，我在这改得要崩溃了，尝试优化，减少内存访问，增加内存设置，修改学习率，增加保存间隔，减少硬盘访问，问题是每跑一次到出结果报这个错误的时候都要十五分钟左右，在这里耗了非常多时间，代金券就剩不到十个小时了，不知道能不能把结果跑出来了。

后续是，没有在代金券用完之前成功跑出来第二版代码的结果，训练时间太

长了，没有结果分数应该很不理想吧，前面都做得很认真所以觉得很可惜，但又觉得伪造个截图交上来实在没有意义。

这个实验前前后后做了半个月的期末周时间，前面七个实验加起来也没半个月，做的时候很多天都是等一个结果十几分钟等到一个报错，非常崩溃，最后也没有成功做出好的结果，会不会是我算法选错了。中间做到一半的时候开始肚子痛发烧，一站起来头针扎一样的疼，一直头昏脑涨到考完最后一门核心，现在写剩下的一部分报告的时候也很晕，如果有部分内容让您觉得我在胡言乱语我很抱歉。这版报告解释的是后一版 mindspore 官网教程的思路，因为当时写报告的时候觉得自己可以完成，而且虽然这版代码由于优化的问题没有时间跑完，德英的那版我还是跑通了的。

做实验前后花了华为 1200，老师给的 800+自己申请的 400，由于欠费问题被迫自己充了 30 块，有点心虚有点好笑。明明我有按需使用，不用就关，一大半都是这个实验花的。

整理的时候发现光这个实验就开了五个 project。

■ project8	a minute ago
■ project8p	10 days ago
■ project8p2	10 days ago
■ projects	14 days ago
■ seq2seq	3 days ago
■ seq2seq2	2 days ago

五、程序文件名清单

project8: 精度低版

seq2seq2: mindspore 官方教程版，由于压缩后超过 100MB 的限制了，所以我把预处理的文件夹和日志全都删掉了，不然下不下来。