

人工神经网络 期末项目

20337251 伍建霖

一、算法原理

词嵌入

词嵌入是自然语言处理中一种常用的技术，用于将文本中的单词或短语映射到实数值向量空间中。它是将离散的文本表示转换为连续的向量表示的一种方法。

在传统的文本处理中，单词通常以离散的形式表示，例如使用独热编码将每个单词表示为一个高维稀疏向量，其中只有一个元素为1，其余元素为0。然而，这种表示方式存在一些问题，如维度灾难和无法捕捉词语之间的语义关系。

词嵌入通过将每个单词映射到一个连续的向量空间中的实数向量来解决这些问题。在这个向量空间中，相似的词在几何上更接近，可以通过计算它们之间的距离或相似度来捕捉它们之间的语义关系。这种表示方式能够更好地捕捉单词的上下文信息和语义含义，从而提供更丰富的语义表达。

词嵌入可以通过不同的方法来学习，其中最常用的方法是使用神经网络模型，如Word2Vec、GloVe和BERT等。这些模型会在大规模的文本语料库上进行训练，通过预测单词的上下文或利用语言模型来学习词嵌入表示。学习得到的词嵌入模型可以应用于各种NLP任务，如文本分类、命名实体识别、情感分析等，以提升模型的性能和效果。

LSTM

LSTM是RNN的一个变体，用于处理序列数据，特别是在自然语言处理和时间序列分析领域。

传统的循环神经网络在处理长序列时存在梯度消失或梯度爆炸的问题，导致难以捕捉长期依赖关系。LSTM通过引入一种称为“门控”机制的方式来解决这个问题，使得网络能够更好地处理长期依赖。

LSTM单元由多个门控单元组成，包括遗忘门、输入门和输出门。每个门控单元都包含一个sigmoid激活函数，用于控制信息的流动。通过这些门控机制，LSTM可以选择性地遗忘或记住输入数据的部分信息，并决定哪些信息应该传递到下一个时间步。

在LSTM中，每个单元有一个隐藏状态和一个细胞状态。隐藏状态类似于传统RNN中的输出，而细胞状态则用于传递和存储长期记忆。通过门控机制，LSTM可以根据输入数据和上一个时间步的隐藏状态来更新细胞状态和隐藏状态。

LSTM在处理序列数据时具有较好的记忆能力和长期依赖建模能力，使得它在许多NLP任务中表现出色。例如，它可以用于文本生成、机器翻译、语音识别、情感分析等任务，也可以作为其他更复杂模型的组成部分，如语言模型和序列到序列模型。

注意力机制

注意力机制（Attention Mechanism）是一种用于增强神经网络在处理序列或集合数据时的能力的机制。它允许模型在处理输入序列时集中关注其中的特定部分，从而更好地捕捉关键信息和上下文之间的关系。

在传统的神经网络中，每个输入都会以相同的权重进行处理，无论其在序列中的位置或重要性如何。而注意力机制通过引入可学习的权重，使模型能够动态地对输入序列中的不同部分分配不同的注意力权重。

在使用注意力机制的模型中，一般会有两个主要组件：查询（Query）、键（Key）和值（Value）。查询用于指定模型要关注的部分，键和值分别表示输入序列的信息。通过计算查询和键之间的相似度，可以得到每个键对于查询的注意力权重。然后，通过将值与对应的注意力权重加权求和，得到聚合后的表示，以便模型更关注重要的部分。

注意力机制可以应用于各种神经网络模型中，包括RNN、CNN和自注意力机制。自注意力机制（如Transformer模型中的注意力机制）在自然语言处理领域中得到广泛应用，并在机器翻译、文本摘要、语言建模等任务中取得了显著的性能提升。

总之，注意力机制使得模型能够有选择性地聚焦于输入序列中的不同部分，并通过动态分配权重来捕捉重要的信息，从而提高了模型的表达能力和性能。

对齐函数

1. 点积（Dot Product）对齐函数：

点积是最简单的对齐函数之一，它直接计算查询向量和键向量之间的点积作为对齐分数。点积对齐函数的优点是计算效率高，因为它只涉及向量的点积操作。点积对齐函数在一些序列到序列任务中表现良好，尤其是当查询向量和键向量的维度相对较小且彼此之间的关联性较高时。

2. 乘法（Multiplicative）对齐函数：

乘法对齐函数通过对查询向量和键向量进行逐元素相乘，然后将结果通过线性变换得到对齐分数。乘法对齐函数允许模型学习查询和键之间的非线性关系，相比于点积对齐函数更加灵活。乘法对齐函数的缺点是计算量较大，因为需要进行逐元素的乘法操作。乘法对齐函数在某些任务中表现良好，特别是当查询向量和键向量的维度较高且彼此之间的关联性较复杂时。

3. 加法（Additive）对齐函数：

加法对齐函数通过将查询向量和键向量进行连接，并通过线性变换和激活函数得到对齐分数。加法对齐函数允许模型学习非线性关系，并且在某些情况下比乘法对齐函数更加有效。加法对齐函数的计算量较大，因为需要进行向量的连接和线性变换操作。加法对齐函数在一些序列到序列任务中表现良好，特别是当查询向量和键向量的维度较低且彼此之间的关联性较简单时。

二、实验代码

数据预处理

中文分词部分，这里我使用jieba进行分词

```
import jieba

input_file = "./data/test_ch.txt"
output_file = "./data/test_ch_seg.txt"

with open(input_file, "r", encoding="utf-8") as file:
    lines = file.readlines()

seg_lines = []
for line in lines:
    seg_words = jieba.cut(line.strip())
    seg_line = " ".join(seg_words)
    seg_lines.append(seg_line+'\n')

with open(output_file, "w", encoding="utf-8") as file:
    file.writelines(seg_lines)
```

```
print("中文分词已完成，结果已保存到train_ch_seg.txt文件中。")
```

英文分词部分，这里我使用nltk进行分词

```
import nltk

# 读取文件并逐行分词
def tokenize_file(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        lines = file.readlines()

    tokenized_lines = []
    for line in lines:
        line = line.strip() # 去除开头和结尾的空白字符
        tokens = nltk.word_tokenize(line)
        tokenized_line = ' '.join(tokens) # 使用空格连接分词后的tokens
        tokenized_lines.append(tokenized_line)

    return tokenized_lines

# 使用示例
input_file_path = './data/test_en.txt'
output_file_path = './data/test_en_seg.txt'

# 分词处理
tokenized_lines = tokenize_file(input_file_path)

# 保存分词结果到文件
with open(output_file_path, 'w', encoding='utf-8') as file:
    for line in tokenized_lines:
        file.write(line + '\n')

print(f"分词后的结果已保存到文件: {output_file_path}")
```

分完词后使用库中的程序生成词表

```
from typing import List
from collections import Counter
from itertools import chain
from docopt import docopt
import json
import torch

from utils import read_corpus, input_transpose

class VocabEntry(object):
    def __init__(self, word2id=None):
        if word2id:
            self.word2id = word2id
        else:
            self.word2id = dict()
            self.word2id['<pad>'] = 0
```

```

        self.word2id['<s>'] = 1
        self.word2id['</s>'] = 2
        self.word2id['<unk>'] = 3

    self.unk_id = self.word2id['<unk>']

    self.id2word = {v: k for k, v in self.word2id.items()}

    def __getitem__(self, word):
        return self.word2id.get(word, self.unk_id)

    def __contains__(self, word):
        return word in self.word2id

    def __setitem__(self, key, value):
        raise ValueError('vocabulary is readonly')

    def __len__(self):
        return len(self.word2id)

    def __repr__(self):
        return 'Vocabulary[size=%d]' % len(self)

    def id2word(self, wid):
        return self.id2word[wid]

    def add(self, word):
        if word not in self:
            wid = self.word2id[word] = len(self)
            self.id2word[wid] = word
            return wid
        else:
            return self[word]

    def words2indices(self, sents):
        if type(sents[0]) == list:
            return [[self[w] for w in s] for s in sents]
        else:
            return [self[w] for w in sents]

    def indices2words(self, word_ids):
        return [self.id2word[w_id] for w_id in word_ids]

    def to_input_tensor(self, sents: List[List[str]], device: torch.device) ->
    torch.Tensor:
        word_ids = self.words2indices(sents)
        sents_t = input_transpose(word_ids, self['<pad>'])

        sents_var = torch.tensor(sents_t, dtype=torch.long, device=device)

        return sents_var

    @staticmethod
    def from_corpus(corpus, size, freq_cutoff=2):
        vocab_entry = VocabEntry()

```

```

word_freq = Counter(chain(*corpus))
valid_words = [w for w, v in word_freq.items() if v >= freq_cutoff]
print(f'number of word types: {len(word_freq)}, number of word types w/
frequency >= {freq_cutoff}: {len(valid_words)}')

top_k_words = sorted(valid_words, key=lambda w: word_freq[w],
reverse=True)[:size]
for word in top_k_words:
    vocab_entry.add(word)

return vocab_entry

class Vocab(object):
    def __init__(self, src_vocab: VocabEntry, tgt_vocab: VocabEntry):
        self.src = src_vocab
        self.tgt = tgt_vocab

    @staticmethod
    def build(src_sents, tgt_sents, vocab_size, freq_cutoff) -> 'vocab':
        assert len(src_sents) == len(tgt_sents)

        print('initialize source vocabulary ..')
        src = VocabEntry.from_corpus(src_sents, vocab_size, freq_cutoff)

        print('initialize target vocabulary ..')
        tgt = VocabEntry.from_corpus(tgt_sents, vocab_size, freq_cutoff)

        return Vocab(src, tgt)

    def save(self, file_path):
        json.dump(dict(src_word2id=self.src.word2id,
tgt_word2id=self.tgt.word2id), open(file_path, 'w'), indent=2)

    @staticmethod
    def load(file_path):
        entry = json.load(open(file_path, 'r'))
        src_word2id = entry['src_word2id']
        tgt_word2id = entry['tgt_word2id']

        return Vocab(VocabEntry(src_word2id), VocabEntry(tgt_word2id))

    def __repr__(self):
        return 'Vocab(source %d words, target %d words)' % (len(self.src),
len(self.tgt))

if __name__ == '__main__':
    args = docopt(__doc__)

    print('read in source sentences: %s' % args['--train-src'])
    print('read in target sentences: %s' % args['--train-tgt'])

    src_sents = read_corpus(args['--train-src'], source='src')

```

```

tgt_sents = read_corpus(args['--train-tgt'], source='tgt')

vocab = Vocab.build(src_sents, tgt_sents, int(args['--size']), int(args['--
freq-cutoff']))
print('generated vocabulary, source %d words, target %d words' %
(len(vocab.src), len(vocab.tgt)))

vocab.save(args['VOCAB_FILE'])
print('vocabulary saved to %s' % args['VOCAB_FILE'])

```

模型结构

模型初始化部分

```

def __init__(self, embed_size, hidden_size, vocab, dropout_rate=0.2,
input_feed=True, label_smoothing=0.):
    super(NMT, self).__init__()

    self.embed_size = embed_size
    self.hidden_size = hidden_size
    self.dropout_rate = dropout_rate
    self.vocab = vocab
    self.input_feed = input_feed

    # initialize neural network layers...
    # initialize embedding layers
    self.src_embed = nn.Embedding(len(vocab.src), embed_size,
padding_idx=vocab.src['<pad>'])
    self.tgt_embed = nn.Embedding(len(vocab.tgt), embed_size,
padding_idx=vocab.tgt['<pad>'])
    # initialize encoder and decoder
    self.encoder_lstm = nn.LSTM(embed_size, hidden_size, bidirectional=True)
    decoder_lstm_input = embed_size + hidden_size if self.input_feed else
embed_size
    self.decoder_lstm = nn.LSTMCell(decoder_lstm_input, hidden_size)

    # attention: dot product attention
    # project source encoding to decoder rnn's state space
    self.att_src_linear = nn.Linear(hidden_size * 2, hidden_size,
bias=False)

    # transformation of decoder hidden states and context vectors before
reading out target words
    # this produces the `attentional vector` in Luong
    self.att_vec_linear = nn.Linear(hidden_size * 2 + hidden_size,
hidden_size, bias=False)

    # prediction layer of the target vocabulary
    self.readout = nn.Linear(hidden_size, len(vocab.tgt), bias=False)

    # dropout layer
    self.dropout = nn.Dropout(self.dropout_rate)

    # initialize the decoder's state and cells with encoder hidden states
    self.decoder_cell_init = nn.Linear(hidden_size * 2, hidden_size)

```

```

self.label_smoothing = label_smoothing
if label_smoothing > 0.:
    self.label_smoothing_loss = LabelSmoothingLoss(label_smoothing,
tgt_vocab_size=len(vocab.tgt), padding_idx=vocab.tgt['<pad>'])

```

编码器部分

```

def encode(self, src_sents_var: torch.Tensor, src_sent_lens: List[int]) ->
Tuple[torch.Tensor, Tuple[torch.Tensor, torch.Tensor]]:
    # (src_sent_len, batch_size, embed_size)
    src_word_embeds = self.src_embed(src_sents_var)
    packed_src_embed = pack_padded_sequence(src_word_embeds, src_sent_lens)

    # src_encodings: (src_sent_len, batch_size, hidden_size * 2)
    src_encodings, (last_state, last_cell) =
self.encoder_lstm(packed_src_embed)
    src_encodings, _ = pad_packed_sequence(src_encodings)

    # (batch_size, src_sent_len, hidden_size * 2)
    src_encodings = src_encodings.permute(1, 0, 2)

    dec_init_cell = self.decoder_cell_init(torch.cat([last_cell[0],
last_cell[1]], dim=1))
    dec_init_state = torch.tanh(dec_init_cell)

    return src_encodings, (dec_init_state, dec_init_cell)

```

解码器部分

```

def decode(self, src_encodings: torch.Tensor, src_sent_masks: torch.Tensor,
decoder_init_vec: Tuple[torch.Tensor, torch.Tensor],
tgt_sents_var: torch.Tensor) -> torch.Tensor:
    # (batch_size, src_sent_len, hidden_size)
    src_encoding_att_linear = self.att_src_linear(src_encodings)

    batch_size = src_encodings.size(0)

    # initialize the attentional vector
    att_tm1 = torch.zeros(batch_size, self.hidden_size, device=self.device)

    # (tgt_sent_len, batch_size, embed_size)
    # here we omit the last word, which is always </s>.
    # Note that the embedding of </s> is not used in decoding
    tgt_word_embeds = self.tgt_embed(tgt_sents_var)

    h_tm1 = decoder_init_vec

    att_ves = []

    # start from y_0='<s>', iterate until y_{T-1}
    for y_tm1_embed in tgt_word_embeds.split(split_size=1):
        y_tm1_embed = y_tm1_embed.squeeze(0)

```

```

        if self.input_feed:
            # input feeding: concate y_tm1 and previous attentional vector
            # (batch_size, hidden_size + embed_size)

            x = torch.cat([y_tm1_embed, att_tm1], dim=-1)
        else:
            x = y_tm1_embed

        (h_t, cell_t), att_t, alpha_t = self.step(x, h_tm1, src_encodings,
src_encoding_att_linear, src_sent_masks)

        att_tm1 = att_t
        h_tm1 = h_t, cell_t
        att_ves.append(att_t)

        # (tgt_sent_len - 1, batch_size, tgt_vocab_size)
        att_ves = torch.stack(att_ves)

    return att_ves

```

注意力机制，这里我使用的对齐函数为点积函数

```

def dot_prod_attention(self, h_t: torch.Tensor, src_encoding: torch.Tensor,
src_encoding_att_linear: torch.Tensor,
                        mask: torch.Tensor=None) -> Tuple[torch.Tensor,
torch.Tensor]:
    # (batch_size, src_sent_len)
    att_weight = torch.bmm(src_encoding_att_linear,
h_t.unsqueeze(2)).squeeze(2)

    if mask is not None:
        att_weight.data.masked_fill_(mask.bool(), -float('inf'))

    softmaxed_att_weight = F.softmax(att_weight, dim=-1)

    att_view = (att_weight.size(0), 1, att_weight.size(1))
    # (batch_size, hidden_size)
    ctx_vec = torch.bmm(softmaxed_att_weight.view(*att_view),
src_encoding).squeeze(1)

    return ctx_vec, softmaxed_att_weight

```

训练阶段

读取数据集并初始化参数

```

train_data_src = read_corpus(args['--train-src'], source='src')
train_data_tgt = read_corpus(args['--train-tgt'], source='tgt')

dev_data_src = read_corpus(args['--dev-src'], source='src')
dev_data_tgt = read_corpus(args['--dev-tgt'], source='tgt')

train_data = list(zip(train_data_src, train_data_tgt))
dev_data = list(zip(dev_data_src, dev_data_tgt))

```



```

train_batch_size = int(args['--batch-size'])
clip_grad = float(args['--clip-grad'])
valid_niter = int(args['--valid-niter'])
log_every = int(args['--log-every'])
model_save_path = args['--save-to']

vocab = vocab.load(args['--vocab'])

model = NMT(embed_size=int(args['--embed-size']),
            hidden_size=int(args['--hidden-size']),
            dropout_rate=float(args['--dropout']),
            input_feed=args['--input-feed'],
            label_smoothing=float(args['--label-smoothing']),
            vocab=vocab)
model.train()

uniform_init = float(args['--uniform-init'])
if np.abs(uniform_init) > 0.:
    print('uniformly initialize parameters [-%f, +%f]' % (uniform_init,
uniform_init), file=sys.stderr)
    for p in model.parameters():
        p.data.uniform_(-uniform_init, uniform_init)

vocab_mask = torch.ones(len(vocab.tgt))
vocab_mask[vocab.tgt['<pad>']] = 0

device = torch.device("cuda:0" if args['--cuda'] else "cpu")
print('use device: %s' % device, file=sys.stderr)

model = model.to(device)

optimizer = torch.optim.Adam(model.parameters(), lr=float(args['--lr']))

num_trial = 0
train_iter = patience = cum_loss = report_loss = cum_tgt_words =
report_tgt_words = 0
cum_examples = report_examples = epoch = valid_num = 0
hist_valid_scores = []
train_time = begin_time = time.time()
print('begin Maximum Likelihood training')

```

开始训练

```

while True:
    epoch += 1

    for src_sents, tgt_sents in batch_iter(train_data,
batch_size=train_batch_size, shuffle=True):
        train_iter += 1

        optimizer.zero_grad()

        batch_size = len(src_sents)

```

```

        # (batch_size)
        example_losses = -model(src_sents, tgt_sents)
        batch_loss = example_losses.sum()
        loss = batch_loss / batch_size

    loss.backward()

    # clip gradient
    grad_norm = torch.nn.utils.clip_grad_norm(model.parameters(),
clip_grad)

    optimizer.step()

    batch_losses_val = batch_loss.item()
    report_loss += batch_losses_val
    cum_loss += batch_losses_val

    tgt_words_num_to_predict = sum(len(s[1:]) for s in tgt_sents) #
    omitting leading '<s>'
    report_tgt_words += tgt_words_num_to_predict
    cum_tgt_words += tgt_words_num_to_predict
    report_examples += batch_size
    cum_examples += batch_size

    if train_iter % log_every == 0:
        print('epoch %d, iter %d, avg. loss %.2f, avg. ppl %.2f ' \
            'cum. examples %d, speed %.2f words/sec, time elapsed %.2f
sec' % (epoch, train_iter,

        report_loss / report_examples,

        math.exp(report_loss / report_tgt_words),

        cum_examples,

        report_tgt_words / (time.time() - train_time),

        time.time() - begin_time), file=sys.stderr)

        train_time = time.time()
        report_loss = report_tgt_words = report_examples = 0.

    # perform validation
    if train_iter % valid_niter == 0:
        print('epoch %d, iter %d, cum. loss %.2f, cum. ppl %.2f cum.
examples %d' % (epoch, train_iter,

        cum_loss / cum_examples,

        np.exp(cum_loss / cum_tgt_words),

        cum_examples), file=sys.stderr)

        cum_loss = cum_examples = cum_tgt_words = 0.
        valid_num += 1

```

```

        print('begin validation ...', file=sys.stderr)

        # compute dev. ppl and bleu
        dev_ppl = evaluate_ppl(model, dev_data, batch_size=128) # dev
        batch size can be a bit larger
        valid_metric = -dev_ppl

        print('validation: iter %d, dev. ppl %f' % (train_iter,
dev_ppl), file=sys.stderr)

        is_better = len(hist_valid_scores) == 0 or valid_metric >
max(hist_valid_scores)
        hist_valid_scores.append(valid_metric)

        if is_better:
            patience = 0
            print('save currently the best model to [%s]' %
model_save_path, file=sys.stderr)
            model.save(model_save_path)

            # also save the optimizers' state
            torch.save(optimizer.state_dict(), model_save_path +
'.optim')

        elif patience < int(args['--patience']):
            patience += 1
            print('hit patience %d' % patience, file=sys.stderr)

        if patience == int(args['--patience']):
            num_trial += 1
            print('hit #%d trial' % num_trial, file=sys.stderr)
            if num_trial == int(args['--max-num-trial']):
                print('early stop!', file=sys.stderr)
                exit(0)

            # decay lr, and restore from previously best checkpoint
            lr = optimizer.param_groups[0]['lr'] * float(args['--lr-
decay'])

            print('load previously best model and decay learning
rate to %f' % lr, file=sys.stderr)

            # load model
            params = torch.load(model_save_path, map_location=lambda
storage, loc: storage)
            model.load_state_dict(params['state_dict'])
            model = model.to(device)

            print('restore parameters of the optimizers',
file=sys.stderr)

            optimizer.load_state_dict(torch.load(model_save_path +
'.optim'))

            # set new lr
            for param_group in optimizer.param_groups:
                param_group['lr'] = lr

```

```

        # reset patience
        patience = 0

    if epoch == int(args['--max-epoch']):
        print('reached maximum number of epochs!', file=sys.stderr)
        exit(0)

```

解码阶段

beam search部分

```

class NMT(nn.Module):
    def beam_search(self, src_sent: List[str], beam_size: int=5,
max_decoding_time_step: int=70) -> List[Hypothesis]:
        src_sents_var = self.vocab.src.to_input_tensor([src_sent], self.device)

        src_encodings, dec_init_vec = self.encode(src_sents_var,
[ len(src_sent)])
        src_encodings_att_linear = self.att_src_linear(src_encodings)

        h_tm1 = dec_init_vec
        att_tm1 = torch.zeros(1, self.hidden_size, device=self.device)

        eos_id = self.vocab.tgt['</s>']

        hypotheses = [['<s>']]
        hyp_scores = torch.zeros(len(hypotheses), dtype=torch.float,
device=self.device)
        completed_hypotheses = []

        t = 0
        while len(completed_hypotheses) < beam_size and t <
max_decoding_time_step:
            t += 1
            hyp_num = len(hypotheses)

            exp_src_encodings = src_encodings.expand(hyp_num,
                                                        src_encodings.size(1),
                                                        src_encodings.size(2))

            exp_src_encodings_att_linear =
src_encodings_att_linear.expand(hyp_num,
src_encodings_att_linear.size(1),
src_encodings_att_linear.size(2))

            y_tm1 = torch.tensor([self.vocab.tgt[hyp[-1]] for hyp in
hypotheses], dtype=torch.long, device=self.device)
            y_tm1_embed = self.tgt_embed(y_tm1)

            if self.input_feed:
                x = torch.cat([y_tm1_embed, att_tm1], dim=-1)
            else:

```

```

x = y_tm1_embed

(h_t, cell_t), att_t, alpha_t = self.step(x, h_tm1,
                                           exp_src_encodings,
                                           exp_src_encodings_att_linear, src_sent_masks=None)

# log probabilities over target words
log_p_t = F.log_softmax(self.readout(att_t), dim=-1)

live_hyp_num = beam_size - len(completed_hypotheses)
continuing_hyp_scores = (hyp_scores.unsqueeze(1).expand_as(log_p_t)
+ log_p_t).view(-1)
top_cand_hyp_scores, top_cand_hyp_pos =
torch.topk(continuing_hyp_scores, k=live_hyp_num)

prev_hyp_ids = top_cand_hyp_pos / len(self.vocab.tgt)
hyp_word_ids = top_cand_hyp_pos % len(self.vocab.tgt)

new_hypotheses = []
live_hyp_ids = []
new_hyp_scores = []

for prev_hyp_id, hyp_word_id, cand_new_hyp_score in
zip(prev_hyp_ids, hyp_word_ids, top_cand_hyp_scores):
    prev_hyp_id = prev_hyp_id.item()
    hyp_word_id = hyp_word_id.item()
    cand_new_hyp_score = cand_new_hyp_score.item()

    hyp_word = self.vocab.tgt.id2word[hyp_word_id]
    new_hyp_sent = hypotheses[prev_hyp_id] + [hyp_word]
    if hyp_word == '</s>':

        completed_hypotheses.append(Hypothesis(value=new_hyp_sent[1:-1],
score=cand_new_hyp_score))
    else:
        new_hypotheses.append(new_hyp_sent)
        live_hyp_ids.append(prev_hyp_id)
        new_hyp_scores.append(cand_new_hyp_score)

if len(completed_hypotheses) == beam_size:
    break

live_hyp_ids = torch.tensor(live_hyp_ids, dtype=torch.long,
device=self.device)
h_tm1 = (h_t[live_hyp_ids], cell_t[live_hyp_ids])
att_tm1 = att_t[live_hyp_ids]

hypotheses = new_hypotheses
hyp_scores = torch.tensor(new_hyp_scores, dtype=torch.float,
device=self.device)

if len(completed_hypotheses) == 0:
    completed_hypotheses.append(Hypothesis(value=hypotheses[0][1:],
score=hyp_scores[0].item()))

```

```

        completed_hypotheses.sort(key=lambda hyp: hyp.score, reverse=True)

    return completed_hypotheses

def beam_search(model: NMT, test_data_src: List[List[str]], beam_size: int,
max_decoding_time_step: int) -> List[List[Hypothesis]]:
    was_training = model.training
    model.eval()

    hypotheses = []
    with torch.no_grad():
        for src_sent in tqdm(test_data_src, desc='Decoding', file=sys.stdout):
            example_hyps = model.beam_search(src_sent, beam_size=beam_size,
max_decoding_time_step=max_decoding_time_step)

            hypotheses.append(example_hyps)

    if was_training: model.train(was_training)

    return hypotheses

```

计算bleu得分

```

def compute_corpus_level_bleu_score(references: List[List[str]], hypotheses:
List[Hypothesis]) -> float:
    if references[0][0] == '<s>':
        references = [ref[1:-1] for ref in references]

    bleu_score = corpus_bleu([[ref] for ref in references],
                             [hyp.value for hyp in hypotheses])

    return bleu_score

```

开始解码

```

def decode(args: Dict[str, str]):
    """
    performs decoding on a test set, and save the best-scoring decoding results.
    If the target gold-standard sentences are given, the function also computes
    corpus-level BLEU score.
    """

    print(f"load test source sentences from [{args['TEST_SOURCE_FILE']}]",
file=sys.stderr)
    test_data_src = read_corpus(args['TEST_SOURCE_FILE'], source='src')
    if args['TEST_TARGET_FILE']:
        print(f"load test target sentences from [{args['TEST_TARGET_FILE']}]",
file=sys.stderr)
        test_data_tgt = read_corpus(args['TEST_TARGET_FILE'], source='tgt')

    print(f"load model from {args['MODEL_PATH']}", file=sys.stderr)
    model = NMT.load(args['MODEL_PATH'])

```

```

if args['--cuda']:
    model = model.to(torch.device("cuda:0"))

    hypotheses = beam_search(model, test_data_src,
                             beam_size=int(args['--beam-size']),
                             max_decoding_time_step=int(args['--max-decoding-
time-step']))

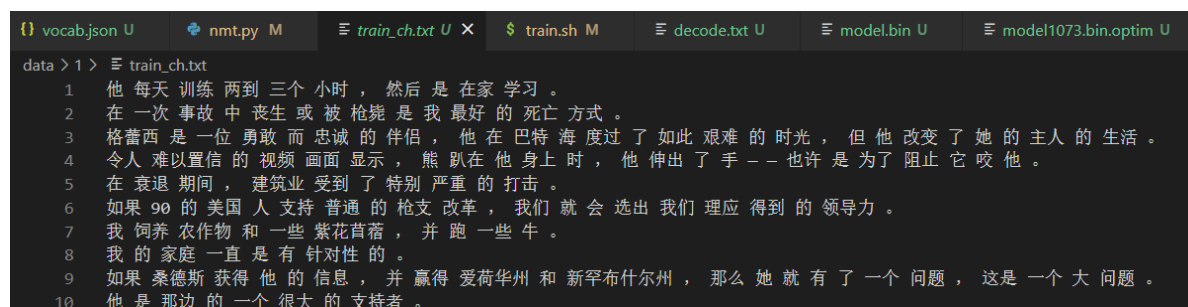
if args['TEST_TARGET_FILE']:
    top_hypotheses = [hyps[0] for hyps in hypotheses]
    bleu_score = compute_corpus_level_bleu_score(test_data_tgt,
top_hypotheses)
    print(f'Corpus BLEU: {bleu_score}', file=sys.stderr)

with open(args['OUTPUT_FILE'], 'w') as f:
    for src_sent, hyps in zip(test_data_src, hypotheses):
        top_hyp = hyps[0]
        hyp_sent = ' '.join(top_hyp.value)
        f.write(hyp_sent + '\n')

```

三、实验结果

中文数据集分词后的结果

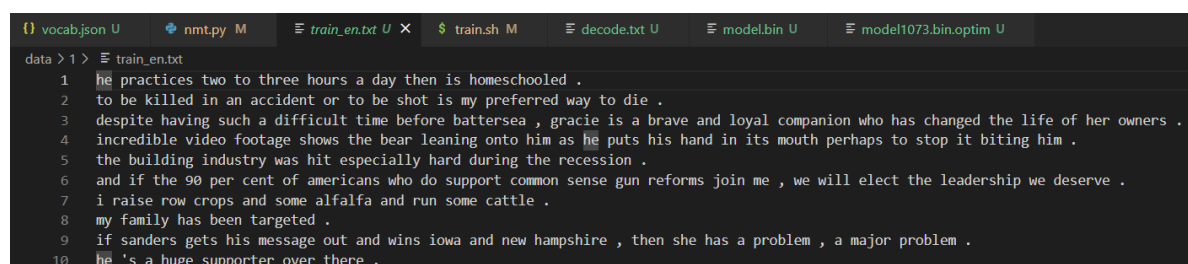


```

data > 1 > train_ch.txt
1 他 每天 训练 两到 三个 小时 ， 然后 是 在家 学习 。
2 在 一次 事故 中 丧生 或 被 枪毙 是 我 最好 的 死亡 方式 。
3 格蕾西 是 一位 勇敢 而 忠诚 的 伴侣 ， 他 在 巴特 海 度过 了 如此 艰难 的 时光 ， 但 他 改变 了 她 的 主人 的 生活 。
4 令人 难以 置信 的 视频 画面 显示 ， 熊 趴 在 他 身上 时 ， 他 伸出 了 手 —— 也许 是 为了 阻止 它 咬 他 。
5 在 衰退 期间 ， 建筑 业 受到 了 特别 严重 的 打击 。
6 如果 90 的 美国 人 支持 普通 的 枪支 改革 ， 我们 就 会 选出 我们 理应 得到 的 领导力 。
7 我 饲养 农作物 和 一些 紫花 苜蓿 ， 并 跑 一些 牛 。
8 我 的 家庭 一直 是 有 针对性 的 。
9 如果 桑德斯 获得 他 的 信息 ， 并 赢得 爱荷华州 和 新罕布什尔州 ， 那么 她 就 有 了 一个 问题 ， 这 是 一个 大 问题 。
10 他 是 那边 的 一个 很大 的 支持者 。

```

英文数据集分词后的结果



```

data > 1 > train_en.txt
1 he practices two to three hours a day then is homeschooled .
2 to be killed in an accident or to be shot is my preferred way to die .
3 despite having such a difficult time before battersea , gracie is a brave and loyal companion who has changed the life of her owners .
4 incredible video footage shows the bear leaning onto him as he puts his hand in its mouth perhaps to stop it biting him .
5 the building industry was hit especially hard during the recession .
6 and if the 90 per cent of americans who do support common sense gun reforms join me , we will elect the leadership we deserve .
7 i raise row crops and some alfalfa and run some cattle .
8 my family has been targeted .
9 if sanders gets his message out and wins iowa and new hampshire , then she has a problem , a major problem .
10 he 's a huge supporter over there .

```

生成的词表

```
{ } vocab.json ... \2 U  nmt.py M  { } vocab.json ... \1 U X  $ train.sh M  ≡ decode.txt U  ≡ model...

data > 1 > { } vocab.json > { } src_word2id
2      "src_word2id": {
9186      "\u516c\u6c11\u6743\u5229": 9183,
9187      "\u8331\u8389": 9184,
9188      "\u7ebd\u897f\u5170": 9185,
9189      "\u53d8\u8f6f": 9186,
9190      "\u6805\u680f": 9187,
9191      "\u516d\u82f1\u5bfc": 9188,
9192      "\u8003": 9189,
9193      "\u71b5": 9190,
9194      "\u8feb\u5207\u9700\u8981": 9191,
9195      "\u4f24\u75a4": 9192,
9196      "\u9999\u69df": 9193,
9197      "\u5fc5\u7136": 9194,
9198      "\u5168\u76db\u65f6\u671f": 9195,
9199      "\u65af\u5854\u65af": 9196,
9200      "\u5927\u9a6c\u58eb\u9769": 9197,
9201      "\u4e2d\u5e74\u7537\u5b50": 9198,
9202      "\u8fc1\u5f99": 9199
9203      },
9204      "tgt_word2id": {
9205      "<pad>": 0,
9206      "<s>": 1,
9207      "</s>": 2,
9208      "<unk>": 3,
9209      "the": 4,
9210      ".": 5,
```

表现最好的一次，bleu4得分为10.73，命令行参数为 `python nmt.py train --cuda --vocab=./data/1/vocab.json --train-src=./data/1/train_ch.txt --train-tgt=./data/1/train_en.txt --dev-src=./data/1/val_ch.txt --dev-tgt=./data/1/val_en.txt --input-feed --valid-niter=100 --label-smoothing=0.1 --dropout=0.5 --seed=3407 --hidden-size=512 --embed-size=512`

```
{ } vocab.json ... \2 U  nmt.py M  { } vocab.json ... \1 U  $ train.sh M  ≡ decode.txt U X  ≡ model.bin U  ≡ model1073.bin.optim U

≡ decode.txt
1  i was hungover from it , i 'm going to feel a bit of a little bit of <unk> .
2  he did for his own own defence , but would not vote for mr <unk> .
3  the summer has been very well from the summer in the summer for the summer and it could eventually be able to shift how they antonio the se
4  a man has been <unk> by <unk> <unk> and <unk> .
5  `` it 's a incredible decade , '' she said .
6  spokesman david murray said a <unk> was placed on various <unk> and then in the public <unk> .
7  germany police , who played out of their complained
8  from the start of the season , there started a plan to start a plan .
9  the fans of the club 's fans are very hard to have a changes and in the past few seasons in the past few seasons , and the fa fans fans <unk>
10 she and luke kissing , finally , and , finally , and finally have been <unk> once once a few years .
```

四、遇到的问题

1. linux环境导致训练卡住：一开始我在wsl上进行训练，但训练进行后就会卡住，我尝试过debug，发现每次卡住的位置都不一样，有的在`output=model(input)`这里卡住，有的在`loss.backward()`这里卡住。最后我只能在windows下进行训练
2. 层数增加效果变差：库中的模型默认就是一层的，我尝试增加层数，然而效果不如一层的
3. 数据预处理错误：一开始错误地认为英语有天然的分割，就不需要进行分词了。
4. 验证集上表现差，但测试集表现好：一开始dev.ppl会逐渐变小，再一点点增加，我在dev.ppl(50左右)最小的时候计算的bleu4得分(3.6)，不如我在dev.ppl增大后(74左右)计算的bleu4得分(5.22)，有可能是数据没预处理好的原因
5. batch size增加，bleu降低：当我把batch size增加到64后，bleu得分降低；把batch size降低到16后，bleu得分接近batch size为32时的得分
6. 中文数据集按字分词效果不如按词分词

五、参考资料

[从简单RNN到带Attention LSTM机器翻译基本思路 - 知乎 \(zhihu.com\)](https://zhuanlan.zhihu.com/p/10731073)

