# NMT 实验报告

## 实验要求

**Seq2Seq模型实现**

- 实现utils.py中的pad sends函数，对batch中的example进行padding补全。
- 实现model_embeddings.py中__init__函数，对source和target embeddings进行初始化。
- 实现nmt_model.py中的__init__函数，对NMT模型参数进行初始化，包括embedings(使用model_embeddings中的embedding），LSTM（layer、dropout、projection)。
- 实现nmt_model.py中的encoder函数，将输入句子转换为hidden表示，可以执行：python sanity_check.py 1d进行初步的正确性检查
- 实现nmt_model.py中的decoder函数。该函数通过逐步调用step函数将hidden表示进行解码。可以执行：python sanity_check.py 1e进行初步的正确性检查。
- 实现nmt_model.py中的step函数。该函数对解码过程中的LSTM cell进行计算，包括target word的encoding h\attention encoding e\output encoding o。可以执行：python sanity_check.py 1f进行初步的正确性检查。
- 实现nmt_model.py中的generate sent masks函数，对batch中添加的padding进行mask操作（参考step函数中如何使用mask)
- 运行代码（要求最终BLEU不低于21）
  1. 产生vocab文件：sh run.sh vocab
  2. 训练：sh run.sh train
  3. 测试：sh run.sh test

## 环境

Windows11、Pycharm、python=3.5、numpy、scipy、tqdm、docopt、pytorch、nltk、torchvision

## 实验过程

1. **实现utils.py中的pad_sents函数，对batch中的example进行padding补全**

   为了进行 tensor 计算，必须确保在给出的 batch 中，句子长度相同。于是我们将一个 batch 中最长的句子长度作为基准，对其它句子进行长度补全。

   先找到最长的句子长度，而后用一个 for 循环补全。

```
def pad_sents(sents, pad_token):
    """ Pad list of sentences according to the longest sentence in the
batch.
    @param sents (list[list[str]]): list of sentences, where each sentence
                                    is represented as a list of words
    @param pad_token (str): padding token
    @returns sents_padded (list[list[str]]): list of sentences where
sentences shorter
        than the max length sentence are padded out with the pad_token, such
that
        each sentences in the batch now has equal length.
    """
```

```
    sents_padded = []

    ### YOUR CODE HERE (~6 Lines)
    max_length = max([len(sentence) for sentence in sents])
    for sentence in sents:
        length = len(sentence)
        if length < max_length:
            sentence += (max_length - length) * [pad_token]
        sents_padded.append(sentence)
    ### END YOUR CODE

    return sents_padded
```

2. **实现model_embeddings.py中__init__函数，对source和target embeddings进行初始化**

   看懂注释中各个参数的意思，并且需要参照 pytorch 帮助文档中的 torch.nn.Embedding，就可以实现了。

   torch.nn.Embedding 帮助文档如下，第一个参数是 embedding 字典的大小，第二个是每个embedding 向量的大小，第三个是不参与 BP 等过程的 padding 的序号。

   EMBEDDING

   CLASS `torch.nn.Embedding(num_embeddings, embedding_dim, padding_idx=None, max_norm=None, norm_type=2.0, scale_grad_by_freq=False, sparse=False, _weight=None, device=None, dtype=None)` [SOURCE]

   A simple lookup table that stores embeddings of a fixed dictionary and size.

   This module is often used to store word embeddings and retrieve them using indices. The input to the module is a list of indices, and the output is the corresponding word embeddings.

   Parameters

   - **num_embeddings** (*int*) – size of the dictionary of embeddings
   - **embedding_dim** (*int*) – the size of each embedding vector
   - **padding_idx** (*int, optional*) – If specified, the entries at `padding_idx` do not contribute to the gradient; therefore, the embedding vector at `padding_idx` is not updated during training, i.e. it remains as a fixed "pad". For a newly constructed Embedding, the embedding vector at `padding_idx` will default to all zeros, but can be updated to another value to be used as the padding vector.
   - **max_norm** (*float, optional*) – If given, each embedding vector with norm larger than `max_norm` is renormalized to have norm `max_norm`.
   - **norm_type** (*float, optional*) – The p of the p-norm to compute for the `max_norm` option. Default `2`.
   - **scale_grad_by_freq** (*boolean, optional*) – If given, this will scale gradients by the inverse of frequency of the words in the mini-batch. Default `False`.
   - **sparse** (*bool, optional*) – If `True`, gradient w.r.t. `weight` matrix will be a sparse tensor. See Notes for more details regarding sparse gradients.

   可得实现如下:

```
class ModelEmbeddings(nn.Module):
    """
    Class that converts input words to their embeddings.
    """

    def __init__(self, embed_size, vocab):
        """
        Init the Embedding layers.

        @param embed_size (int): Embedding size (dimensionality)
        @param vocab (Vocab): Vocabulary object containing src and tgt
languages
                              See vocab.py for documentation.
```

```
        """
        super(ModelEmbeddings, self).__init__()
        self.embed_size = embed_size

        # default values
        self.source = None
        self.target = None

        src_pad_token_idx = vocab.src['<pad>']
        tgt_pad_token_idx = vocab.tgt['<pad>']

        ### YOUR CODE HERE (~2 Lines)
        ### TODO - Initialize the following variables:
        ###     self.source (Embedding Layer for source language)
        ###     self.target (Embedding Layer for target langauge)
        ###
        ### Note:
        ###     1. `vocab` object contains two vocabularies:
        ###            `vocab.src` for source
        ###            `vocab.tgt` for target
        ###     2. You can get the length of a specific vocabulary by
running:
        ###             `len(vocab.<specific_vocabulary>)`
        ###     3. Remember to include the padding token for the specific
vocabulary
        ###        when creating your Embedding.
        ###
        ### Use the following docs to properly initialize these variables:
        ###     Embedding Layer:
        ###
https://pytorch.org/docs/stable/nn.html#torch.nn.Embedding
        self.source = nn.Embedding(len(vocab.src), embed_size,
src_pad_token_idx)
        self.target = nn.Embedding(len(vocab.tgt), embed_size,
tgt_pad_token_idx)
        ### END YOUR CODE
```

3. **实现nmt_model.py中的__init__函数，对NMT模型参数进行初始化，包括embedings(使用model_embeddings中的embedding），LSTM（layer、dropout、projection)**

   根据每层的维度要求，参阅 pytorch 的帮助文档完成，具体见 CS224N 2019 的 a4.pdf 公式推导部分。

```
def __init__(self, embed_size, hidden_size, vocab, dropout_rate=0.2):
    """ Init NMT Model.

    @param embed_size (int): Embedding size (dimensionality)
    @param hidden_size (int): Hidden Size (dimensionality)
    @param vocab (Vocab): Vocabulary object containing src and tgt languages
                          See vocab.py for documentation.
    @param dropout_rate (float): Dropout probability, for attention
    """
    super(NMT, self).__init__()
    self.model_embeddings = ModelEmbeddings(embed_size, vocab)
    self.hidden_size = hidden_size
    self.dropout_rate = dropout_rate
    self.vocab = vocab
```

```
    # default values
    self.encoder = None
    self.decoder = None
    self.h_projection = None
    self.c_projection = None
    self.att_projection = None
    self.combined_output_projection = None
    self.target_vocab_projection = None
    self.dropout = None

    ### YOUR CODE HERE (~8 Lines)
    ### TODO - Initialize the following variables:
    ###     self.encoder (Bidirectional LSTM with bias)
    ###     self.decoder (LSTM Cell with bias)
    ###     self.h_projection (Linear Layer with no bias), called W_{h} in
the PDF.
    ###     self.c_projection (Linear Layer with no bias), called W_{c} in
the PDF.
    ###     self.att_projection (Linear Layer with no bias), called
W_{attProj} in the PDF.
    ###     self.combined_output_projection (Linear Layer with no bias),
called W_{u} in the PDF.
    ###     self.target_vocab_projection (Linear Layer with no bias), called
W_{vocab} in the PDF.
    ###     self.dropout (Dropout Layer)
    ###
    ### Use the following docs to properly initialize these variables:
    ###     LSTM:
    ###         https://pytorch.org/docs/stable/nn.html#torch.nn.LSTM
    ###     LSTM Cell:
    ###         https://pytorch.org/docs/stable/nn.html#torch.nn.LSTMCell
    ###     Linear Layer:
    ###         https://pytorch.org/docs/stable/nn.html#torch.nn.Linear
    ###     Dropout Layer:
    ###         https://pytorch.org/docs/stable/nn.html#torch.nn.Dropout

    # PDF上都是 h x 2h，在这里都要转置一下，变成 2h x h 等，因为一个 embedding 在实现
中是横着的，比如 1 x h 其实是 h x 1
    self.encoder = nn.LSTM(
        input_size=embed_size,
        hidden_size=self.hidden_size,
        bias=True,
        bidirectional=True)
    self.decoder = nn.LSTMCell(
        input_size=embed_size + self.hidden_size,  # 此处要注意
        hidden_size=self.hidden_size,
        bias=True
    )
    self.h_projection = nn.Linear(
        in_features=self.hidden_size * 2,
        out_features=self.hidden_size,
        bias=False
    )
    self.c_projection = nn.Linear(
        in_features=self.hidden_size * 2,
        out_features=self.hidden_size,
        bias=False
```

```
    )
    self.att_projection = nn.Linear(
        in_features=self.hidden_size * 2,
        out_features=self.hidden_size,
        bias=False
    )
    self.combined_output_projection = nn.Linear(
        in_features=self.hidden_size * 3,
        out_features=self.hidden_size,
        bias=False
    )
    self.target_vocab_projection = nn.Linear(
        in_features=self.hidden_size,
        out_features=len(self.vocab.tgt),
        bias=False
    )
    self.dropout = nn.Dropout(
        p=self.dropout_rate
    )
    ### END YOUR CODE
```

4. **实现nmt_model.py中的encode函数，将输入句子转换为hidden表示**

   参阅 pytorch 的帮助文档和函数注释完成，这个 encode 函数的目的是：将输入句子转换为 hidden 表示，并计算 Decoder 的初始 $h_0^{dec}$ 和 $c_0^{dec}$ 。

```
def encode(self, source_padded: torch.Tensor, source_lengths: List[int]) ->
Tuple[
    torch.Tensor, Tuple[torch.Tensor, torch.Tensor]]:
    """ Apply the encoder to source sentences to obtain encoder hidden
states.
        Additionally, take the final states of the encoder and project them
to obtain initial states for decoder.

    @param source_padded (Tensor): Tensor of padded source sentences with
shape (src_len, b), where
                                    b = batch_size, src_len = maximum source
sentence length. Note that
                                   these have already been sorted in order
of longest to shortest sentence.
    @param source_lengths (List[int]): List of actual lengths for each of
the source sentences in the batch
    @returns enc_hiddens (Tensor): Tensor of hidden units with shape (b,
src_len, h*2), where
                                    b = batch size, src_len = maximum source
sentence length, h = hidden size.
    @returns dec_init_state (tuple(Tensor, Tensor)): Tuple of tensors
representing the decoder's initial
                                            hidden state and cell.
    """
    enc_hiddens, dec_init_state = None, None

    ### YOUR CODE HERE (~ 8 Lines)
    ### TODO:
    ###     1. Construct Tensor `X` of source sentences with shape (src_len,
b, e) using the source model embeddings.
```

```
    ###         src_len = maximum source sentence length, b = batch size, e
= embedding size. Note
    ###         that there is no initial hidden state or cell for the
decoder.
    ###     2. Compute `enc_hiddens`, `last_hidden`, `last_cell` by applying
the encoder to `X`.
    ###         - Before you can apply the encoder, you need to apply the
`pack_padded_sequence` function to X.
    ###         - After you apply the encoder, you need to apply the
`pad_packed_sequence` function to enc_hiddens.
    ###         - Note that the shape of the tensor returned by the encoder
is (src_len b, h*2) and we want to
    ###           return a tensor of shape (b, src_len, h*2) as
`enc_hiddens`.
    ###     3. Compute `dec_init_state` = (init_decoder_hidden,
init_decoder_cell):
    ###         - `init_decoder_hidden`:
    ###             `last_hidden` is a tensor shape (2, b, h). The first
dimension corresponds to forwards and backwards.
    ###             Concatenate the forwards and backwards tensors to obtain
a tensor shape (b, 2*h).
    ###             Apply the h_projection layer to this in order to compute
init_decoder_hidden.
    ###             This is h_0^{dec} in the PDF. Here b = batch size, h =
hidden size
    ###         - `init_decoder_cell`:
    ###             `last_cell` is a tensor shape (2, b, h). The first
dimension corresponds to forwards and backwards.
    ###             Concatenate the forwards and backwards tensors to obtain
a tensor shape (b, 2*h).
    ###             Apply the c_projection layer to this in order to compute
init_decoder_cell.
    ###             This is c_0^{dec} in the PDF. Here b = batch size, h =
hidden size
    ###
    ### See the following docs, as you may need to use some of the following
functions in your implementation:
    ###     Pack the padded sequence X before passing to the encoder:
    ###
https://pytorch.org/docs/stable/nn.html#torch.nn.utils.rnn.pack_padded_seque
nce
    ###     Pad the packed sequence, enc_hiddens, returned by the encoder:
    ###
https://pytorch.org/docs/stable/nn.html#torch.nn.utils.rnn.pad_packed_sequen
ce
    ###     Tensor Concatenation:
    ###         https://pytorch.org/docs/stable/torch.html#torch.cat
    ###     Tensor Permute:
    ###
https://pytorch.org/docs/stable/tensors.html#torch.Tensor.permute
    X = self.model_embeddings.source(source_padded)
    enc_hiddens, (last_hidden, last_cell) =
self.encoder(pack_padded_sequence(input=X, lengths=source_lengths))
    enc_hiddens = pad_packed_sequence(sequence=enc_hiddens,
batch_first=True)[0]
    last_hidden = torch.cat(tensors=(last_hidden[0, :], last_hidden[1, :]),
dim=1)
    init_decoder_hidden = self.h_projection(last_hidden)
```

```
    last_cell = torch.cat(tensors=(last_cell[0, :], last_cell[1, :]), dim=1)
    init_decoder_cell = self.c_projection(last_cell)
    dec_init_state = (init_decoder_hidden, init_decoder_cell)

    ### END YOUR CODE

    return enc_hiddens, dec_init_state
```

5. **实现nmt_model.py中的decode函数，该函数通过逐步调用step函数将hidden表示进行解码**

   参阅 pytorch 的帮助文档和函数注释完成，这个 decode 函数的目的是：在每个 timestep 中为输入构建 $\bar{y}$ 并运行 step 函数。

```
def decode(self, enc_hiddens: torch.Tensor, enc_masks: torch.Tensor,
           dec_init_state: Tuple[torch.Tensor, torch.Tensor], target_padded:
torch.Tensor) -> torch.Tensor:
    """Compute combined output vectors for a batch.

    @param enc_hiddens (Tensor): Hidden states (b, src_len, h*2), where
                                 b = batch size, src_len = maximum source
sentence length, h = hidden size.
    @param enc_masks (Tensor): Tensor of sentence masks (b, src_len), where
                                 b = batch size, src_len = maximum source
sentence length.
    @param dec_init_state (tuple(Tensor, Tensor)): Initial state and cell
for decoder
    @param target_padded (Tensor): Gold-standard padded target sentences
(tgt_len, b), where
                                   tgt_len = maximum target sentence length,
b = batch size.

    @returns combined_outputs (Tensor): combined output tensor  (tgt_len, b,
h), where
                                    tgt_len = maximum target sentence
length, b = batch_size,  h = hidden size
    """
    # Chop of the <END> token for max length sentences.
    target_padded = target_padded[:-1]

    # Initialize the decoder state (hidden and cell)
    dec_state = dec_init_state

    # Initialize previous combined output vector o_{t-1} as zero
    batch_size = enc_hiddens.size(0)
    o_prev = torch.zeros(batch_size, self.hidden_size, device=self.device)

    # Initialize a list we will use to collect the combined output o_t on
each step
    combined_outputs = []

    ### YOUR CODE HERE (~9 Lines)
    ### TODO:
    ###     1. Apply the attention projection layer to `enc_hiddens` to
obtain `enc_hiddens_proj`,
    ###         which should be shape (b, src_len, h),
    ###         where b = batch size, src_len = maximum source length, h =
hidden size.
```

```
    ###         This is applying W_{attProj} to h^enc, as described in the
PDF.
    ###     2. Construct tensor `Y` of target sentences with shape (tgt_len,
b, e) using the target model embeddings.
    ###         where tgt_len = maximum target sentence length, b = batch
size, e = embedding size.
    ###     3. Use the torch.split function to iterate over the time
dimension of Y.
    ###         Within the loop, this will give you Y_t of shape (1, b, e)
where b = batch size, e = embedding size.
    ###             - Squeeze Y_t into a tensor of dimension (b, e).
    ###             - Construct Ybar_t by concatenating Y_t with o_prev.
    ###             - Use the step function to compute the the Decoder's
next (cell, state) values
    ###               as well as the new combined output o_t.
    ###             - Append o_t to combined_outputs
    ###             - Update o_prev to the new o_t.
    ###     4. Use torch.stack to convert combined_outputs from a list
length tgt_len of
    ###         tensors shape (b, h), to a single tensor shape (tgt_len, b,
h)
    ###         where tgt_len = maximum target sentence length, b = batch
size, h = hidden size.
    ###
    ### Note:
    ###    - When using the squeeze() function make sure to specify the
dimension you want to squeeze
    ###      over. Otherwise, you will remove the batch dimension
accidentally, if batch_size = 1.
    ###
    ### Use the following docs to implement this functionality:
    ###     Zeros Tensor:
    ###         https://pytorch.org/docs/stable/torch.html#torch.zeros
    ###     Tensor Splitting (iteration):
    ###         https://pytorch.org/docs/stable/torch.html#torch.split
    ###     Tensor Dimension Squeezing:
    ###         https://pytorch.org/docs/stable/torch.html#torch.squeeze
    ###     Tensor Concatenation:
    ###         https://pytorch.org/docs/stable/torch.html#torch.cat
    ###     Tensor Stacking:
    ###         https://pytorch.org/docs/stable/torch.html#torch.stack
    enc_hiddens_proj = self.att_projection(enc_hiddens)
    Y = self.model_embeddings.target(target_padded)
    for item in torch.split(tensor=Y, split_size_or_sections=1):
        Y_t = torch.squeeze(input=item)
        Ybar_t = torch.cat(tensors=(o_prev, Y_t), dim=1)
        dec_state, o_t, e_t = self.step(Ybar_t=Ybar_t, dec_state=dec_state,
enc_hiddens=enc_hiddens,
                                        enc_hiddens_proj=enc_hiddens_proj,
enc_masks=enc_masks)
        combined_outputs.append(o_t)
        o_prev = o_t
    combined_outputs = torch.stack(combined_outputs)
    ### END YOUR CODE

    return combined_outputs
```

6. **实现nmt_model.py中的step函数。该函数对解码过程中的LSTM cell进行计算**

参阅 pytorch 的帮助文档和函数注释完成，这个 step 函数的目的是：在每个 timstep 中为目标单词进行编码得到 $h_t^{dec}$ ，并计算注意力分数 $e_t$ ，注意力分布 $\alpha_t$ ，注意力输出 $a_t$ ，还有最终合成的输出 $o_t$ 。

```
def step(self, Ybar_t: torch.Tensor,
         dec_state: Tuple[torch.Tensor, torch.Tensor],
         enc_hiddens: torch.Tensor,
         enc_hiddens_proj: torch.Tensor,
         enc_masks: torch.Tensor) -> Tuple[Tuple, torch.Tensor,
torch.Tensor]:
    """ Compute one forward step of the LSTM decoder, including the
attention computation.

    @param Ybar_t (Tensor): Concatenated Tensor of [Y_t o_prev], with shape
(b, e + h). The input for the decoder,
                            where b = batch size, e = embedding size, h =
hidden size.
    @param dec_state (tuple(Tensor, Tensor)): Tuple of tensors both with
shape (b, h), where b = batch size, h = hidden size.
            First tensor is decoder's prev hidden state, second tensor is
decoder's prev cell.
    @param enc_hiddens (Tensor): Encoder hidden states Tensor, with shape
(b, src_len, h * 2), where b = batch size,
                                src_len = maximum source length, h = hidden
size.
    @param enc_hiddens_proj (Tensor): Encoder hidden states Tensor,
projected from (h * 2) to h. Tensor is with shape (b, src_len, h),
                                where b = batch size, src_len = maximum
source length, h = hidden size.
    @param enc_masks (Tensor): Tensor of sentence masks shape (b, src_len),
                                where b = batch size, src_len is maximum
source length.

    @returns dec_state (tuple (Tensor, Tensor)): Tuple of tensors both shape
(b, h), where b = batch size, h = hidden size.
            First tensor is decoder's new hidden state, second tensor is
decoder's new cell.
    @returns combined_output (Tensor): Combined output Tensor at timestep t,
shape (b, h), where b = batch size, h = hidden size.
    @returns e_t (Tensor): Tensor of shape (b, src_len). It is attention
scores distribution.
                            Note: You will not use this outside of this
function.
                                  We are simply returning this value so that
we can sanity check
                                  your implementation.
    """

    combined_output = None

    ### YOUR CODE HERE (~3 Lines)
    ### TODO:
    ###     1. Apply the decoder to `Ybar_t` and `dec_state`to obtain the
new dec_state.
    ###     2. Split dec_state into its two parts (dec_hidden, dec_cell)
```

```
    ###     3. Compute the attention scores e_t, a Tensor shape (b,
src_len).
    ###        Note: b = batch_size, src_len = maximum source length, h =
hidden size.
    ###
    ###       Hints:
    ###         - dec_hidden is shape (b, h) and corresponds to h^dec_t in
the PDF (batched)
    ###         - enc_hiddens_proj is shape (b, src_len, h) and corresponds
to W_{attProj} h^enc (batched).
    ###         - Use batched matrix multiplication (torch.bmm) to compute
e_t.
    ###         - To get the tensors into the right shapes for bmm, you will
need to do some squeezing and unsqueezing.
    ###         - When using the squeeze() function make sure to specify the
dimension you want to squeeze
    ###             over. Otherwise, you will remove the batch dimension
accidentally, if batch_size = 1.
    ###
    ### Use the following docs to implement this functionality:
    ###     Batch Multiplication:
    ###        https://pytorch.org/docs/stable/torch.html#torch.bmm
    ###     Tensor Unsqueeze:
    ###         https://pytorch.org/docs/stable/torch.html#torch.unsqueeze
    ###     Tensor Squeeze:
    ###         https://pytorch.org/docs/stable/torch.html#torch.squeeze
    dec_state = self.decoder(Ybar_t, dec_state)
    dec_hidden, dec_cell = dec_state
    e_t = torch.squeeze(input=torch.bmm(input=enc_hiddens_proj,
mat2=torch.unsqueeze(dec_hidden, 2)), dim=2)
    ### END YOUR CODE

    # Set e_t to -inf where enc_masks has 1
    if enc_masks is not None:
        e_t.data.masked_fill_(enc_masks.byte(), -float('inf'))

    ### YOUR CODE HERE (~6 Lines)
    ### TODO:
    ###     1. Apply softmax to e_t to yield alpha_t
    ###     2. Use batched matrix multiplication between alpha_t and
enc_hiddens to obtain the
    ###         attention output vector, a_t.
    # $$     Hints:
    ###           - alpha_t is shape (b, src_len)
    ###           - enc_hiddens is shape (b, src_len, 2h)
    ###           - a_t should be shape (b, 2h)
    ###           - You will need to do some squeezing and unsqueezing.
    ###     Note: b = batch size, src_len = maximum source length, h =
hidden size.
    ###
    ###     3. Concatenate dec_hidden with a_t to compute tensor U_t
    ###     4. Apply the combined output projection layer to U_t to compute
tensor V_t
    ###     5. Compute tensor O_t by first applying the Tanh function and
then the dropout layer.
    ###
    ### Use the following docs to implement this functionality:
    ###     Softmax:
```

```
    ###
https://pytorch.org/docs/stable/nn.html#torch.nn.functional.softmax
    ###     Batch Multiplication:
    ###        https://pytorch.org/docs/stable/torch.html#torch.bmm
    ###     Tensor View:
    ###
https://pytorch.org/docs/stable/tensors.html#torch.Tensor.view
    ###     Tensor Concatenation:
    ###         https://pytorch.org/docs/stable/torch.html#torch.cat
    ###     Tanh:
    ###         https://pytorch.org/docs/stable/torch.html#torch.tanh
    alpha_t = F.softmax(input=e_t, dim=1)
    a_t = torch.squeeze(input=torch.bmm(input=torch.unsqueeze(alpha_t, 1),
mat2=enc_hiddens), dim=1)
    U_t = torch.cat(tensors=(a_t, dec_hidden), dim=1)
    V_t = self.combined_output_projection(U_t)
    O_t = self.dropout(torch.tanh(V_t))
    ### END YOUR CODE

    combined_output = O_t
    return dec_state, combined_output, e_t
```

7. **实现nmt_model.py中的generate_sent_masks函数，对batch中添加的padding进行mask操作（参考step函数中如何使用mask）**

   可得实现如下：

```
def generate_sent_masks(self, enc_hiddens: torch.Tensor, source_lengths:
List[int]) -> torch.Tensor:
    """ Generate sentence masks for encoder hidden states.

    @param enc_hiddens (Tensor): encodings of shape (b, src_len, 2*h), where
b = batch size,
                                 src_len = max source length, h = hidden
size.
    @param source_lengths (List[int]): List of actual lengths for each of
the sentences in the batch.

    @returns enc_masks (Tensor): Tensor of sentence masks of shape (b,
src_len),
                                where src_len = max source length, h =
hidden size.
    """
    enc_masks = torch.zeros(enc_hiddens.size(0), enc_hiddens.size(1),
dtype=torch.float)
    for e_id, src_len in enumerate(source_lengths):
        enc_masks[e_id, src_len:] = 1
    return enc_masks.to(self.device)
```

   在 Attention 计算中的 softmax 等对于 padding token 也会参与计算，因为padding token 只是用于实现 mini-batch，没有语义信息，于是需要 generate_sent_masks 函数将这部分给 mask，避免干扰计算。

# 实验结果

模型训练默认运行 30 个epoch。

由于本机（RTX 2060 MAX-Q）显存较小，只有 6G ，在第 7 个 epoch 时数据溢出，无法完成训练；colab 训练需要大约 8 小时，而连续免费使用时间并没有 8 小时，无法完成训练。

不过可以看到，在 epoch 为 10 时，avg-loss 已经可以达到最低 27.75，而开始训练时的 avg-loss 为 160 左右。