A4_AkshayParate_SeqtoSeq

April 23, 2024

1 CS 584 Assignment 4 – Sequence to Sequence Models

Name: Akshay Parate

Stevens ID: 20032008

[2]:
```
# from google.colab import drive
# drive.mount('/content/drive')
```

[ ]:
```

```

1.1 In this assignment, you are required to follow the steps below:

1. Review the lecture slides.
2. Implement the seq2seq (translation) model.

Before you start - Please read the code very carefully. - Install these packages using the following command.

`pip install -r requirements.txt`

- It's better to train the Tensorflow model with GPU and CUDA. If they are not available on your local machine, please consider Google CoLab. You can check `CoLab.md` in this assignments.
- You are **NOT** allowed to use other packages unless otherwise specified.
- You are **ONLY** allowed to edit the code between `# Start your code here` and `# End` for each block.

[3]:
```
# pip install -r requirements.txt
```

[4]:
```
import sys
import os


def print_line(*args):
    """ Inline print and go to the begining of line
    """
    args1 = [str(arg) for arg in args]
    str_ = ' '.join(args1)
    print('\r' + str_, end='')
```

[5]:
```
import tensorflow as tf



# If you are going to use GPU, make sure the GPU in in the output
gpus = tf.config.list_physical_devices('GPU')
```

[6]:
```
from typing import List, Tuple, Union, Dict

import numpy as np
```

1.2 1. Data preparation (5 Points)

1.2.1 1.1 Load and describe data

Here, we use the iwslt2017 dataset. More specifically, this translation task is from French to English: fr-en.

[7]:
```
from datasets import load_dataset
# The load_dataset function is provided by the huggingface datasets
# https://huggingface.co/docs/datasets/index



dataset_path = os.path.join('a4-data', 'dataset')
dataset = load_dataset('iwslt2017', 'iwslt2017-en-fr', cache_dir=dataset_path,␣
↪ignore_verifications=True)
```

```
C:\Users\Akshay\anaconda3\envs\tensorflow\lib\site-packages\tqdm\auto.py:21:
TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
C:\Users\Akshay\anaconda3\envs\tensorflow\lib\site-
packages\datasets\load.py:2524: FutureWarning: 'ignore_verifications' was
deprecated in favor of 'verification_mode' in version 2.9.1 and will be removed
in 3.0.0.
You can remove this warning by passing 'verification_mode=no_checks' instead.
  warnings.warn(
```

Let's first print some basic statistics of this dataset

[8]:
```
print(dataset)
print(len(dataset['train']['translation']),␣
↪len(dataset['validation']['translation']), len(dataset['test']['translation']))
```

```
DatasetDict({
    train: Dataset({
        features: ['translation'],
        num_rows: 232825
    })
    test: Dataset({
```

```
        features: ['translation'],
        num_rows: 8597
    })
    validation: Dataset({
        features: ['translation'],
        num_rows: 890
    })
})
232825 890 8597
```

[9]:
```
# dataset['train']['translation'][0]
```

[10]:
```
print(dataset['train']['translation'][0])
```

```
{'en': "Thank you so much, Chris. And it's truly a great honor to have the
opportunity to come to this stage twice; I'm extremely grateful.", 'fr': "Merci
beaucoup, Chris. C'est vraiment un honneur de pouvoir venir sur cette scène une
deuxième fois. Je suis très reconnaissant."}
```

[11]:
```
from tokenizers import Tokenizer
# The tokenizer is provided by the huggingface tokenizers
# https://huggingface.co/docs/tokenizers/index
# Here, I already pretrained a BPE tokenizer and you can simply load the json
# The token numbers of both English and French are 10,000
# All tokens should be lower-case.
# en_tokenizer = Tokenizer.from_file('./drive/MyDrive/a4-data/en_tokenizer.json')
# fr_tokenizer = Tokenizer.from_file('./drive/MyDrive/a4-data/fr_tokenizer.json')
en_tokenizer = Tokenizer.from_file('./a4-data/en_tokenizer.json')
fr_tokenizer = Tokenizer.from_file('./a4-data/fr_tokenizer.json')
```

[12]:
```
encoding = en_tokenizer.encode("i like sports.")
print(encoding.ids)
print(encoding.tokens)
# >>> [0, 122, 279, 4987, 17, 1]
# >>> ['<s>', 'Ġi', 'Ġlike', 'Ġsports', '.', '</s>']
```

```
[0, 122, 279, 4987, 17, 1]
['<s>', 'Ġi', 'Ġlike', 'Ġsports', '.', '</s>']
```

Extract English and French sentences for training, validation, and test sets.

Note: Every sentence is lower-case.

[13]:
```
train_en_sentences, train_fr_sentences = zip(*[(pair['en'].lower(), pair['fr'].
 ↪lower()) for pair in dataset['train']['translation']])
valid_en_sentences, valid_fr_sentences = zip(*[(pair['en'].lower(), pair['fr'].
 ↪lower()) for pair in dataset['validation']['translation']])
test_en_sentences, test_fr_sentences = zip(*[(pair['en'].lower(), pair['fr'].
 ↪lower()) for pair in dataset['test']['translation']])
```

[14]:
```
test_en_sentences[0]
```

[14]:
```
'several years ago here at ted, peter skillman  introduced a design challenge
called the marshmallow challenge.'
```

[15]:
```
test_fr_sentences[0]
```

[15]:
```
"il y a plusieurs années, ici à ted, peter skillman a présenté une épreuve de
conception appelée l'épreuve du marshmallow."
```

1.2.2 1.2 Encode data (5 Points)

[16]:
```
def encode(tokenizer: 'Tokenizer', sentences: List[str]) -> List[List[int]]:
    """ Encode the sentences with the pretrained tokenizer.
        You can directly call `tokenizer.encode()` to encode the sentences.
        It will automatically add the <s> and </s> token.

        Note: Please be carefull with the return value of the encode function.

    Args:
        tokenizer: A pretrained en/fr tokenizer
        sentences: A list of strings
    Return:
        sent_token_ids: A list of token ids
    """
    sent_token_ids = []
    n = len(sentences)
    for i, sentence in enumerate(sentences):
        if i % 100 == 0 or i == n - 1:
            print_line('Encoding with Tokenizer:', (i + 1), '/', n)
        # Start your code here
        sent_token_ids.append(tokenizer.encode(sentence).ids)
        # End
    print_line('\n')
    return sent_token_ids
```

[17]:
```
print('en')
train_en = encode(en_tokenizer, train_en_sentences)
valid_en = encode(en_tokenizer, valid_en_sentences)
test_en = encode(en_tokenizer, test_en_sentences)
print('fr')
train_fr = encode(fr_tokenizer, train_fr_sentences)
valid_fr = encode(fr_tokenizer, valid_fr_sentences)
test_fr = encode(fr_tokenizer, test_fr_sentences)
```

```
en
Encoding with Tokenizer: 232825 / 232825
Encoding with Tokenizer: 890 / 890
```

```
Encoding with Tokenizer: 8597 / 8597
fr
Encoding with Tokenizer: 232825 / 232825
Encoding with Tokenizer: 890 / 890
Encoding with Tokenizer: 8597 / 8597
```

[ ]:

Check your implementation with an example

[18]:
```
print(dataset['train']['translation'][0])
print(train_en[0], train_fr[0])
print(en_tokenizer.decode(train_en[0]), fr_tokenizer.decode(train_fr[0]))
```

```
{'en': "Thank you so much, Chris. And it's truly a great honor to have the
opportunity to come to this stage twice; I'm extremely grateful.", 'fr': "Merci
beaucoup, Chris. C'est vraiment un honneur de pouvoir venir sur cette scène une
deuxième fois. Je suis très reconnaissant."}
[0, 658, 162, 188, 494, 15, 2843, 17, 138, 165, 178, 2775, 121, 630, 4502, 140,
222, 124, 1930, 140, 625, 140, 185, 2122, 3446, 30, 122, 400, 2576, 5818, 17, 1]
[0, 763, 478, 15, 3016, 17, 145, 10, 178, 487, 169, 8981, 152, 1038, 2055, 266,
323, 2425, 220, 1760, 586, 17, 214, 459, 378, 9952, 17, 1]
 thank you so much, chris. and it's truly a great honor to have the opportunity
to come to this stage twice; i'm extremely grateful.  merci beaucoup, chris.
c'est vraiment un honneur de pouvoir venir sur cette scène une deuxième fois. je
suis très reconnaissant.
```

1.3 2. Sequence to sequence model (40 Points)

1.3.1 2.1 Encoder (10 Points)

[19]:
```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Layer, GRU, Dense, Embedding, Dropout
from tensorflow.keras.initializers import GlorotUniform


class Encoder(Model):
    def __init__(self, vocab_size: int, embedding_size: int, units: int):
        """ The encoder model for the src sentences.
            It contains an embedding part and a GRU part.

        Args:
            vocab_size: The src vocabulary size
            embedding_size: The embedding size for the embedding layer
            units: Number of hidden units in the RNN (GRU) layer
        """
        super().__init__()
        # Start your code here
```

```
        # Note: Please know what the decoder needs from encoder. This determines␣
↪the parameters of the GRU layer
        self.embedding = Embedding(vocab_size, embedding_size)
        self.gru = GRU(units, return_sequences=True, return_state=True,␣
↪recurrent_initializer='glorot_uniform')
        # End

    def call(self, src_ids, src_mask):
        """ Encoder forward
        Args:
            src_ids: Tensor, (batch_size x max_len), the token ids of input␣
↪sentences in a batch
            src_mask: Tensor, (batch_size x max_len), the mask of the src input.␣
↪True value in the mask means this timestep is valid, otherwise this timestep␣
↪is ignored
        Returns:
            enc_output: Tensor, (batch_size x max_len x units), the output of␣
↪GRU for all timesteps
            final_state: Tensor, (batch_size x units), the state of the final␣
↪valid timestep
        """
        # Start your code here
        # Step 1. Retrieve embedding
        #      2. GRU
        # Please refer to the calling arguments of GRU: https://www.tensorflow.
↪org/api_docs/python/tf/keras/layers/GRU#call-arguments
        # print(src_ids)
        # print(src_ids[0])
        # print()
        embedded = self.embedding(src_ids)
        # print(embedded)
        enc_outputs, final_state = self.gru(embedded, mask=src_mask)
        # End
        return enc_outputs, final_state
```

1.3.2 2.2 Decoder (15 Points)

[20]:
```
class Decoder(Model):
    def __init__(self, vocab_size: int, embedding_size: int, units: int,␣
↪dropout_rate: float):
        """ The decoder model for the tgt sentences.
            It contains an embedding part, a GRU part, a dropout part, and a␣
↪classifier part.

        Args:
            vocab_size: The tgt vocabulary size
```

```
            embedding_size: The embedding size for the embedding layer
            units: Number of hidden units in the RNN (GRU) layer
            dropout_rate: The classifier has a (units x vocab_size) weight. This␣
↪is a large weight matrix. We apply a dropout layer to avoid overfitting.
        """
        super().__init__()
        # Start your code here
        # Note: 1. Please correctly set the parameter of GRU
        #       2. No softmax here because we will need the sequence to sequence␣
↪loss later
        self.embedding = Embedding(vocab_size, embedding_size)
        self.gru = GRU(units, return_sequences=True, return_state=True,␣
↪recurrent_initializer='glorot_uniform')
        self.dropout = Dropout(dropout_rate)
        self.classifier = Dense(vocab_size)
        # End

    def call(self, tgt_ids, initial_state, tgt_mask):
        """ Decoder forward.
            It is called by decoder(tgt_ids=..., initial_state=..., tgt_mask=...)

        Args:
            tgt_ids: Tensor, (batch_size x max_len), the token ids of input␣
↪sentences in a batch
            initial_state: Tensor, (batch_size x units), the state of the final␣
↪valid timestep from the encoder
            tgt_mask: Tensor, (batch_size x max_len), the mask of the tgt input.␣
↪True value in the mask means this timestep is valid, otherwise this timestep␣
↪is ignored
        Return:
            dec_outputs: Tensor, (batch_size x max_len x vocab_size), the output␣
↪of GRU for all timesteps
        """
        # Start your code here
        # Step 1. Retrieve embedding
        #      2. GRU
        #      3. Apply dropout to the GRU output
        #      4. Classifier
        # Note: Please refer to the calling arguments of GRU: https://www.
↪tensorflow.org/api_docs/python/tf/keras/layers/GRU#call-arguments
        embedded = self.embedding(tgt_ids)
        dec_output, final_state = self.gru(embedded,␣
↪initial_state=initial_state, mask=tgt_mask)
        dec_outputs = self.dropout(dec_output)
        dec_outputs = self.classifier(dec_outputs)
        # End
```

```
        return dec_outputs

    def predict(self, tgt_ids, initial_state):
        """ Decoder prediction.
            This is a step in recursive prediction. We use the previous␣
↪prediction and state to predict current token.
            Note that we only need to use the gru_cell instead of GRU becasue we␣
↪only need to calculate one timestep.

        Args:
            tgt_ids: Tensor, (batch_size, ) -> (1, ), the token id of the␣
↪current timestep in the current sentence.
            initial_state: Tensor, (batch_size x units) -> (1 x units), the␣
↪state of the final valid timestep from the encoder or the previous hidden␣
↪state in prediction.
        Return:
            dec_outputs: Tensor, (batch_size x vocab_size) -> (1 x vocab_size),␣
↪the output of GRU for this timestep.
            state: Tensor, (batch_size x units) -> (1 x units), the state of␣
↪this timestep.
        """
        gru_cell = self.gru.cell
        # Start your code here
        # Step 1. Retrieve embedding
        #       2. GRU Cell, see https://www.tensorflow.org/api_docs/python/tf/
↪keras/layers/GRUCell#call-arguments
        #       3. Classifier (No dropout)
        embedded = self.embedding(tgt_ids)
        dec_output, state = self.gru(embedded, initial_state=initial_state)
        dec_outputs = self.classifier(dec_output)
        # End
        return dec_outputs, state
```

1.3.3 2.3 Seq2seq (10 Points)

[46]:
```
class Seq2seq(Model):
    def __init__(self, src_vocab_size: int, tgt_vocab_size: int, embedding_size:␣
↪int, units: int, dropout_rate: float):
        """ The sequence to sequence model.
            It contains an encoder and a decoder.

        Args:
            src_vocab_size: The src vocabulary size
            tgt_vocab_size: The tgt vocabulary size
            embedding_size: The embedding size for the embedding layer
            units: Number of hidden units in the RNN (GRU) layer
```

```
            dropout_rate: The dropout rate used in the decoder.
        """
        super().__init__()
        # Start your code here
        self.encoder = Encoder(src_vocab_size, embedding_size, units)
        self.decoder = Decoder(tgt_vocab_size, embedding_size, units,␣
↪dropout_rate)
        # End

    def call(self, src_ids, src_seq_lens, tgt_ids, tgt_seq_lens):
        """ Seq2seq forward (for the loss calculation in training/validation␣
↪only).
            It is called by model(src_ids=..., src_seq_lens=..., tgt_ids=...,␣
↪tgt_seq_lens=)
            Note: In prediction, we will also need to set `training=False`.

        Args:
            src_ids: Tensor, (batch_size x max_len), the token ids of src␣
↪sentences in a batch
            src_seq_lens: Tensor, (batch_size, ), the length of src sentences in␣
↪a batch
            tgt_ids: Tensor, (batch_size x max_len), the token ids of tgt␣
↪sentences in a batch
            tgt_seq_lens: Tensor, (batch_size, ), the length of src sentences in␣
↪a batch
        Returns:
            dec_outputs: Tensor, (batch_size x max_len x units), the decoder␣
↪predictions
        """
        # Start your code here
        # Step 1. build mask for src and tgt
        #       2. encoder forward
        #       3. decoder forward
        src_mask = tf.sequence_mask(src_seq_lens)
        tgt_mask = tf.sequence_mask(tgt_seq_lens)
        # print(src_ids[0])
        # print()
        enc_output, enc_state = self.encoder(src_ids, src_mask)
        dec_outputs = self.decoder(tgt_ids, enc_state, tgt_mask)
        # End
        return dec_outputs
```

[47]:
```
class Seq2seqWithAttention(Model):
    def __init__(self, src_vocab_size: int, tgt_vocab_size: int, embedding_size:␣
↪int, units: int, dropout_rate: float):
        """ The sequence to sequence model.
```

```
            It contains an encoder and a decoder.

        Args:
            src_vocab_size: The src vocabulary size
            tgt_vocab_size: The tgt vocabulary size
            embedding_size: The embedding size for the embedding layer
            units: Number of hidden units in the RNN (GRU) layer
            dropout_rate: The dropout rate used in the decoder.
        """
        super().__init__()
        # Start your code here
        self.encoder = Encoder(src_vocab_size, embedding_size, units)
        self.attention = Attention(units)
        self.decoder = DecoderWithAttention(tgt_vocab_size, embedding_size,␣
↪units, dropout_rate)
        # End

    def call(self, src_ids, src_seq_lens, tgt_ids, tgt_seq_lens):
        """ Seq2seq forward (for the loss calculation in training/validation␣
↪only).
            It is called by model(src_ids=..., src_seq_lens=..., tgt_ids=...,␣
↪tgt_seq_lens=)
            Note: In prediction, we will also need to set `training=False`.

        Args:
            src_ids: Tensor, (batch_size x max_len), the token ids of src␣
↪sentences in a batch
            src_seq_lens: Tensor, (batch_size, ), the length of src sentences in␣
↪a batch
            tgt_ids: Tensor, (batch_size x max_len), the token ids of tgt␣
↪sentences in a batch
            tgt_seq_lens: Tensor, (batch_size, ), the length of src sentences in␣
↪a batch
        Returns:
            dec_outputs: Tensor, (batch_size x max_len x units), the decoder␣
↪predictions
        """
        # Start your code here
        # Step 1. build mask for src and tgt
        #      2. encoder forward
        #      3. decoder forward
        src_mask = tf.sequence_mask(src_seq_lens)
        tgt_mask = tf.sequence_mask(tgt_seq_lens)
        # print(src_ids[0])
        # print()
        enc_output, enc_state = self.encoder(src_ids, src_mask)
```

```
        # Initialize the decoder state with the encoder state
        dec_state = enc_state

        # Initialize the attention weights
        attention_weights = None

        # Initialize the decoder outputs
        dec_outputs = []

        # Iterate over each timestep in the target sequence
        for t in range(tgt_ids.shape[1]):
            # Apply attention mechanism to compute context vector
            context_vector, attention_weights = self.attention(enc_output,␣
↪dec_state)

            # Pass context vector and previous decoder state to the decoder
            dec_output, dec_state = self.decoder(tgt_ids[:, t:t+1], dec_state,␣
↪context_vector)

            # Store decoder output
            dec_outputs.append(dec_output)

        # Stack decoder outputs along the timestep axis
        dec_outputs = tf.stack(dec_outputs, axis=1)

        return dec_outputs
```

1.3.4 2.4 Seq2seq loss (5 Points)

[22]:
```
from tensorflow_addons.seq2seq import sequence_loss


def seq2seq_loss(logits, target, seq_lens):
    """ Calculate the sequence to sequence loss using the sequence_loss from␣
↪tensorflow

    Args:
        logits: Tensor (batch_size x max_seq_len x vocab_size). The output of␣
↪the RNN model.
        target: Tensor (batch_size x max_seq_len). The groud-truth of words.
        seq_lens: Tensor (batch_size, ). The real sequence length before padding.
    """
    loss = 0
    # Start your code here
    # 1. make a sequence mask (batch_size x max_seq_len) using tf.sequence_mask.␣
↪This is to build a mask with 1 and 0.
```

```
    mask = tf.sequence_mask(seq_lens, maxlen=tf.shape(target)[1], dtype=tf.
↪float32)
    #    Entry with 1 is the valid time step without padding. Entry with 0 is␣
↪the time step with padding. We need to exclude this time step.
    # 2. calculate the loss with sequence_loss. Carefully read the documentation␣
↪of each parameter
    loss = sequence_loss(logits, target,mask,average_across_timesteps =␣
↪True,average_across_batch = True)
    # End
    return loss
```

```
C:\Users\Akshay\anaconda3\envs\tensorflow\lib\site-
packages\tensorflow_addons\utils\tfa_eol_msg.py:23: UserWarning:

TensorFlow Addons (TFA) has ended development and introduction of new features.
TFA has entered a minimal maintenance and release mode until a planned end of
life in May 2024.
Please modify downstream libraries to take dependencies from other repositories
in our TensorFlow community (e.g. Keras, Keras-CV, and Keras-NLP).

For more information see: https://github.com/tensorflow/addons/issues/2807

  warnings.warn(
C:\Users\Akshay\anaconda3\envs\tensorflow\lib\site-
packages\tensorflow_addons\utils\ensure_tf_install.py:53: UserWarning:
Tensorflow Addons supports using Python ops for all Tensorflow versions above or
equal to 2.12.0 and strictly below 2.15.0 (nightly versions are not supported).
 The versions of TensorFlow you are currently using is 2.10.1 and is not
supported.
Some things might work, some things might not.
If you were to encounter a bug, do not file an issue.
If you want to make sure you're using a tested and supported configuration,
either change the TensorFlow version or the TensorFlow Addons's version.
You can find the compatibility matrix in TensorFlow Addon's readme:
https://github.com/tensorflow/addons
  warnings.warn(
```

1.4 3. Training (50 Points)

1.4.1 3.1 Pad batch (15 Points)

`pad_src_batch`: 5 Points `pad_tgt_batch`: 10 Points

Pad the batch to the equal length and make tensors.

[23]:
```
def pad_src_batch(src_batch: List[List[int]], src_seq_lens: List[int], pad_val:␣
↪int):
    """ Pad the batch for src sentences.
```

```
        Note: Do not use append/extend that can modify the input inplace.

    Args:
        src_batch: A list of src token ids
        src_seq_lens: A list of src lens
        pad_val: The padding value

    Returns:
        src_batch: Tensor, (batch_size x max_len)
        src_seq_lens_batch: Tensor, (batch_size, )
    """
    max_src_len = max(src_seq_lens)
    # Start your code here
    num_sent = len(src_seq_lens)
    # Please refer to tf.convert_to_tensor. The dtype should be tf.int64
    # Padding
    src_batch_padded = [sentence + [pad_val] * (max_src_len - len(sentence)) for␣
↪sentence in src_batch]
    # Convert to tensor
    src_batch = tf.convert_to_tensor(src_batch_padded, dtype=tf.int64)
    src_seq_lens_batch = tf.convert_to_tensor(src_seq_lens, dtype=tf.int64)
    # End
    return src_batch, src_seq_lens_batch
```

[24]:
```
def pad_tgt_batch(tgt_batch: List[List[int]], tgt_seq_lens: List[int], pad_val:␣
↪int):
    """ Pad the batch for tgt sentences.
        Note: 1. Do not use append/extend that can modify the input inplace.
              2. We need to build the x (feature) and y (label) for tgt␣
↪sentences.
                 Please understand what the feature and label are in translation.

    Args:
        tgt_batch: A list of src token ids
        tgt_seq_lens: A list of src lens
        pad_val: The padding value

    Returns:
        tgt_x_batch: Tensor, (batch_size x max_len)
        tgt_y_batch: Tensor, (batch_size x max_len)
        src_seq_lens_batch: Tensor, (batch_size, )
    """
    tgt_x_batch, tgt_y_batch, tgt_seq_lens_batch = [], [], []
    for sent, seq_len in zip(tgt_batch, tgt_seq_lens):
        # Start your code here
        # Append x, y, and seq_len
        tgt_x_batch.append(sent[:-1])  # Exclude last token for y
```

```
        tgt_y_batch.append(sent[1:])    # Exclude first token for x
        tgt_seq_lens_batch.append(seq_len - 1)  # Reduce sequence length by 1␣
↪for y
        # End

    max_tgt_len = max(tgt_seq_lens_batch)
    # Start your code here
    # Please refer to tf.convert_to_tensor. The dtype should be tf.int64
    # Padding
    tgt_x_batch = [sentence + [pad_val] * (max_tgt_len - len(sentence)) for␣
↪sentence in tgt_x_batch]
    tgt_y_batch = [sentence + [pad_val] * (max_tgt_len - len(sentence)) for␣
↪sentence in tgt_y_batch]

    # Convert to tensor
    tgt_x_batch = tf.convert_to_tensor(tgt_x_batch, dtype=tf.int64)
    tgt_y_batch = tf.convert_to_tensor(tgt_y_batch, dtype=tf.int64)
    tgt_seq_lens_batch = tf.convert_to_tensor(tgt_seq_lens_batch, dtype=tf.int64)
    # End
    return tgt_x_batch, tgt_y_batch, tgt_seq_lens_batch
```

[25]:
```
def pad_batch(src_batch: List[List[int]], src_seq_lens: List[int], tgt_batch:␣
↪List[List[int]], tgt_seq_lens: List[int], pad_val: int):
    src_batch, src_seq_lens_batch = pad_src_batch(src_batch, src_seq_lens,␣
↪pad_val)
    tgt_x_batch, tgt_y_batch, tgt_seq_lens_batch = pad_tgt_batch(tgt_batch,␣
↪tgt_seq_lens, pad_val)
    return src_batch, src_seq_lens_batch, tgt_x_batch, tgt_y_batch,␣
↪tgt_seq_lens_batch
```

1.4.2 3.2 Batch Index Sampler (10 Points)

Create a index sampler to sample data index for each batch.

This is to make the sentences in each batch have similar lengths to speed up training.

Example:

```
Assume the sentence lengths are: [5, 2, 3, 6, 2, 3, 6] and batch_size is 2.
We can make the indices in the batches as follows:
[1, 4] of length 2
[2, 5] of length 3
[0, 3] of lengths 5 and 6
[6] of length 6
```

[26]:
```
class SeqLenBatchSampler:
    def __init__(self, seq_lens: List[int], batch_size: int, seed: int = 6666):
        """ The index sampler.
```

```
            It can be used with iteration:
            ```
            n_batch = len(sampler)
            for indices in sampler:
                ...
            ```

            Args:
                seq_lens: A list training sequence lengths (src)
                batch_size: .
                seed: .
        """
        np.random.seed(seed)
        self.seq_lens = seq_lens
        self.batch_size = batch_size
        self.batches = self._make_batch_index()

        self.n_batch = len(self.batches)
        self.counter = -1

    def _make_batch_index(self) -> List[List[int]]:
        """ Build the indexes in each batch.

            Return:
                batches: A list of indices batch, e.g., [[0, 2, 8], [3, 6, 4],␣
↪[5, 1, 7], ...]
        """
        n = len(self.seq_lens)
        n_batch = int(np.ceil(n / self.batch_size))
        batches = []
        # Start your code here
        # Step 1. Use np.argsort to get all indices with sorted length
        #      2. Split the indices into batches using a for loop: `for i in␣
↪range(n_batch):`
        sorted_indices = np.argsort(self.seq_lens)
        batches = [sorted_indices[i::n_batch] for i in range(n_batch)]
        # End
        return batches

    def __len__(self):
        return self.n_batch

    def __item__(self, index):
        return self.batches[index]

    def __iter__(self):
        np.random.shuffle(self.batches)
```

```
        self.counter = -1
        return self

    def __next__(self):
        self.counter += 1
        if self.counter < self.n_batch:
            return self.batches[self.counter]
        raise StopIteration
```

1.4.3 3.3 Running the model

Generate the length

[27]:
```
np.random.seed(6666)
train_seq_lens_en = [len(en_sent) for en_sent in train_en]
train_seq_lens_fr = [len(fr_sent) for fr_sent in train_fr]
valid_seq_lens_en = [len(en_sent) for en_sent in valid_en]
valid_seq_lens_fr = [len(fr_sent) for fr_sent in valid_fr]
test_seq_lens_en = [len(en_sent) for en_sent in test_en]
test_seq_lens_fr = [len(fr_sent) for fr_sent in test_fr]
```

Create np array

[28]:
```
train_en = np.array(train_en, dtype=object)
train_seq_lens_en = np.array(train_seq_lens_en)
train_fr = np.array(train_fr, dtype=object)
train_seq_lens_fr = np.array(train_seq_lens_fr)
```

Model parameters

[29]:
```
import random


seed = 6666
random.seed(seed)
np.random.seed(seed)
tf.random.set_seed(seed)
```

[30]:
```
src_vocab_size = len(fr_tokenizer.get_vocab())
tgt_vocab_size = len(en_tokenizer.get_vocab())
hidden_units = 256
embedding_dim = 128
dropout_rate = 0.0
```

[31]:
```
model = Seq2seq(src_vocab_size, tgt_vocab_size, embedding_dim, hidden_units,␣
↪dropout_rate)
```

[ ]:
```

```

[32]:
```
num_epoch = 15
# batch_size = 256
# num_epoch = 1
batch_size = 32
learning_rate = 1e-3
```

[33]:
```
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
train_batch_sampler = SeqLenBatchSampler(train_seq_lens_fr, batch_size)
```

[ ]:
```
```

[34]:
```
n_training_samples = len(train_fr)
n_valid_batch = int(np.ceil(len(valid_fr) / batch_size))
pad_token_id = fr_tokenizer.token_to_id('<pad>')
train_losses, valid_losses = [], []
for epoch in range(num_epoch):
    epoch_loss = 0.0
    for batch_idx, data_index in enumerate(train_batch_sampler):
        src_batch, src_seq_lens = train_fr[data_index],␣
↪train_seq_lens_fr[data_index]

        tgt_batch, tgt_seq_lens = train_en[data_index],␣
↪train_seq_lens_en[data_index]
        real_batch_size = len(src_batch)
        (src_batch, src_seq_lens_batch,
         tgt_x_batch, tgt_y_batch, tgt_seq_lens_batch) = pad_batch(src_batch,␣
↪src_seq_lens,
                                                                  tgt_batch,␣
↪tgt_seq_lens,
                                                                  ␣
↪pad_val=pad_token_id)
        with tf.GradientTape() as tape:
            # print(src_batch[0])
            # print()
            output = model(src_batch, src_seq_lens_batch, tgt_x_batch,␣
↪tgt_seq_lens_batch)
            loss = seq2seq_loss(output, tgt_y_batch, tgt_seq_lens_batch)

        print_line(f'Epoch {epoch + 1} / {num_epoch} - Step {batch_idx + 1} /␣
↪{len(train_batch_sampler)} - loss: {loss:.4f}')

        trainable_vars = model.trainable_variables
        gradients = tape.gradient(loss, trainable_vars)

        # Update weights
        optimizer.apply_gradients(zip(gradients, trainable_vars))
        epoch_loss += loss * real_batch_size
```

```
    valid_loss = 0.0
    for batch_idx in range(n_valid_batch):
        start = batch_idx * batch_size
        end = start + batch_size
        src_batch, src_seq_lens = valid_fr[start:end], valid_seq_lens_fr[start:
↪end]
        tgt_batch, tgt_seq_lens = valid_en[start:end], valid_seq_lens_en[start:
↪end]
        real_batch_size = len(src_batch)
        (src_batch, src_seq_lens_batch,
         tgt_x_batch, tgt_y_batch, tgt_seq_lens_batch) = pad_batch(src_batch,␣
↪src_seq_lens,
                                                                  tgt_batch,␣
↪tgt_seq_lens,
                                                                  ␣
↪pad_val=pad_token_id)
        output = model(src_batch, src_seq_lens_batch, tgt_x_batch,␣
↪tgt_seq_lens_batch, training=False)
        loss = seq2seq_loss(output, tgt_y_batch, tgt_seq_lens_batch)

        if batch_idx % 1 == 0 or batch_idx == len(valid_en) - 1:
            print_line(f'Epoch {epoch + 1} / {num_epoch} - Step {batch_idx + 1} /
↪ {n_valid_batch} - loss: {loss:.4f}')

        valid_loss += loss * real_batch_size
    train_epoch_loss = epoch_loss / n_training_samples
    valid_epoch_loss = valid_loss / len(valid_en)
    train_losses.append(train_epoch_loss)
    valid_losses.append(valid_epoch_loss)
    print(f'\rEpoch {epoch + 1} / {num_epoch} - Step {len(train_batch_sampler)} /
↪ {len(train_batch_sampler)} - train loss: {train_epoch_loss:.4f} - valid loss:␣
↪{valid_epoch_loss:.4f}')
```

```
Epoch 1 / 15 - Step 7276 / 7276 - train loss: 4.5990 - valid loss: 4.4556
Epoch 2 / 15 - Step 7276 / 7276 - train loss: 3.7599 - valid loss: 4.1394
Epoch 3 / 15 - Step 7276 / 7276 - train loss: 3.4482 - valid loss: 4.0177
Epoch 4 / 15 - Step 7276 / 7276 - train loss: 3.2521 - valid loss: 3.9648
Epoch 5 / 15 - Step 7276 / 7276 - train loss: 3.1115 - valid loss: 3.9091
Epoch 6 / 15 - Step 7276 / 7276 - train loss: 3.0056 - valid loss: 3.9491
Epoch 7 / 15 - Step 7276 / 7276 - train loss: 2.9224 - valid loss: 3.9134
Epoch 8 / 15 - Step 7276 / 7276 - train loss: 2.8532 - valid loss: 3.9331
Epoch 9 / 15 - Step 7276 / 7276 - train loss: 2.7957 - valid loss: 3.9571
Epoch 10 / 15 - Step 7276 / 7276 - train loss: 2.7465 - valid loss: 3.9695
Epoch 11 / 15 - Step 7276 / 7276 - train loss: 2.7041 - valid loss: 3.9746
Epoch 12 / 15 - Step 7276 / 7276 - train loss: 2.6681 - valid loss: 3.9969
Epoch 13 / 15 - Step 7276 / 7276 - train loss: 2.6360 - valid loss: 3.9928
```

```
Epoch 14 / 15 - Step 7276 / 7276 - train loss: 2.6077 - valid loss: 4.0397
Epoch 15 / 15 - Step 7276 / 7276 - train loss: 2.5825 - valid loss: 4.0501
```

If you implement everything correctly, the valid loss will be around 4.

[35]:
```
model.summary(expand_nested=True)
```

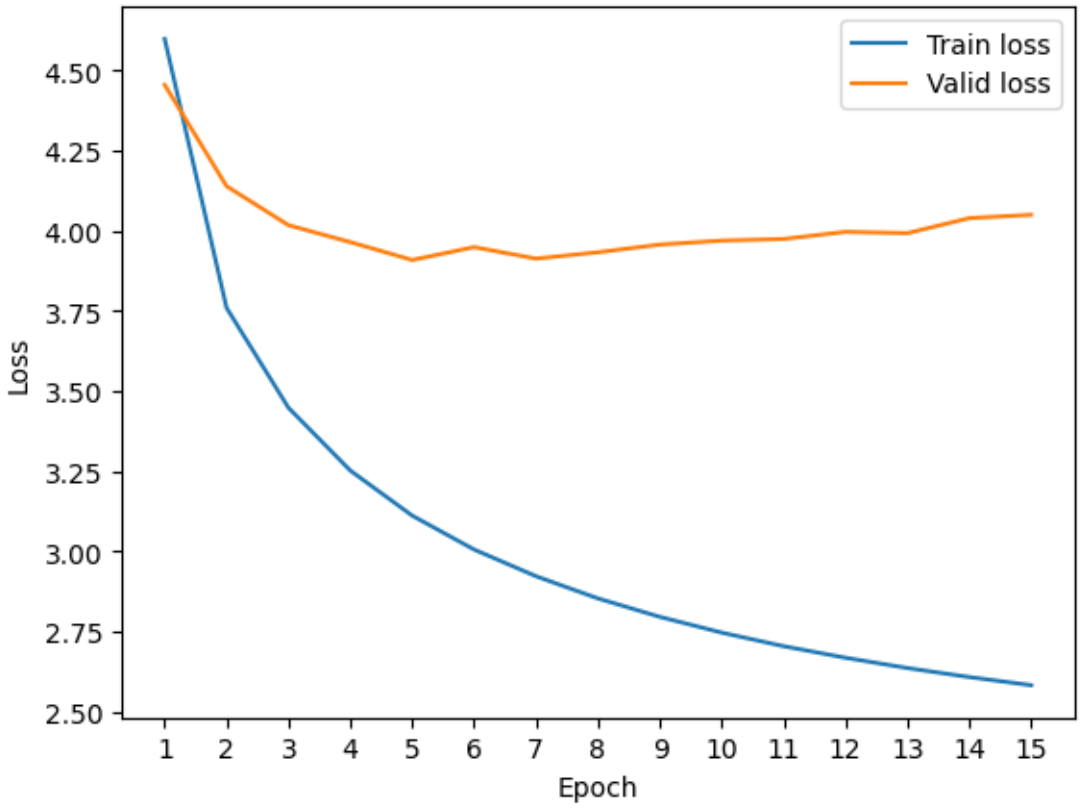
```
Model: "seq2seq"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 encoder (Encoder)            multiple                  1576448
|---------------------------------------------------------------|
| embedding (Embedding)      multiple                  1280000   |
|                                                               |
| gru (GRU)                  multiple                  296448    |
_________________________________________________________________

 decoder (Decoder)            multiple                  4146448
|---------------------------------------------------------------|
| embedding_1 (Embedding)    multiple                  1280000   |
|                                                               |
| gru_1 (GRU)                multiple                  296448    |
|                                                               |
| dropout (Dropout)          multiple                  0         |
|                                                               |
| dense (Dense)              multiple                  2570000   |
_________________________________________________________________

=================================================================
Total params: 5,722,896
Trainable params: 5,722,896
Non-trainable params: 0
_________________________________________________________________
```

[36]:
```
%matplotlib inline
from matplotlib import pyplot as plt


x = np.arange(1, len(train_losses) + 1)
plt.plot(x, train_losses, label='Train loss')
plt.plot(x, valid_losses, label='Valid loss')
plt.legend()
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.xticks(x)
plt.show()
```



1.4.4 3.4 Translate French to English (15 Points)

[37]:
```
sos_token_id = en_tokenizer.token_to_id('<s>')
eos_token_id = en_tokenizer.token_to_id('</s>')
max_pred_len = 200
def translate(encoder: 'Encoder', decoder: 'Decoder', fr_sentences:␣
↪List[List[int]]):
    """ Translate the src (French) sentences to English sentences.
        This is a recursive translation.

    Args:
        encoder: The encoder part in seq2seq
        decoder: The decoder part in seq2seq
        fr_sentences: The src token ids of all sentences
    Returns:
        pred_sentences: The predicted string sentences
    """
    n = len(fr_sentences)
    pred_sentences = []
    for i, src_ids in enumerate(fr_sentences):
        print_line(f'{i + 1} / {n}')
```

```
        # Shape of src_ids: (1 x seq_len)
        src_ids = tf.expand_dims(tf.convert_to_tensor(src_ids, dtype=tf.int64),␣
↪axis=0)
        # pred is the prediction token ids. It starts with <s>
        pred = [sos_token_id]
        # Start your code here
        # Step 1. Calculate the encoder outputs and hidden states (similar to␣
↪seq2seq2 model)
        # Step 2. Run a while loop when the last token in pred is not␣
↪eos_token_id and the length of pred is less than max_pred_len
        # Step 3.     In the while loop, build the input (cur_token) of decoder:␣
↪the last token of pred. Shape (batch_size, ) -> (1, )
        #             For example, if the current pred is [1, 50, 21, 8], the␣
↪cur_token is [8]
        # Step 4.     In the while loop, use decoder.predict to get the decoder␣
↪output
        # Step 5.     In the while loop, find the index with the maximum value.␣
↪Then you can call tf.squeeze and numpy() to get the index
        # Step 6.     In the while loop, append the predicted token to pred
        # Step 7. Use en_tokenizer to decode the id to strings: pred_sentence
        enc_output, enc_hidden = encoder(src_ids, None)
        dec_hidden = enc_hidden
        while pred[-1] != eos_token_id and len(pred) < max_pred_len:
    #         # Prepare input for decoder
            cur_token = np.array([pred[-1]])  # Shape: (1,)
    #         # Get decoder output for the current token
            dec_output, dec_hidden = decoder.predict(tf.expand_dims(cur_token,␣
↪axis=0), dec_hidden)
    #         # Get index of predicted token with maximum probability
            predicted_index = int(tf.argmax(dec_output[0], axis=-1)[-1])
    #         # Append predicted token to pred
            pred.append(predicted_index)
    #     # Decode predicted sentence using English tokenizer
        pred_sentence = en_tokenizer.decode(pred)
    #     # End
        pred_sentences.append(pred_sentence)
    print_line('\n')
    return pred_sentences
```

[38]:
```
model.save('my_model', save_format='tf')
```

```
WARNING:absl:Found untraced functions such as gru_cell_layer_call_fn,
gru_cell_layer_call_and_return_conditional_losses, gru_cell_1_layer_call_fn,
gru_cell_1_layer_call_and_return_conditional_losses while saving (showing 4 of
4). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: my_model\assets
```

```
INFO:tensorflow:Assets written to: my_model\assets
```

[ ]:

[39]:
```
test_pred = translate(model.encoder, model.decoder, fr_sentences=test_fr)
```

```
8597 / 8597
```

1.4.5 3.5 Demonstrate 10 translation examples (5 Points)

1.4.6 3.6 Compute the bleu score (5 Points)

[ ]:

[40]:
```
len(test_pred)
```

[40]:
```
8597
```

[48]:
```
import numpy as np
np.random.seed(6666)
sample_num = 10
# Start your code here
# Use np.random.choice to sample 10 sentence indices. Remember to set correct␣
 ↪replace
# Print format:
# 1.
# French: ...
# True English: ...
# Translated English: ...
# ------------------
sampled_indices = np.random.choice(len(test_pred), size=sample_num,␣
 ↪replace=False)
# Print the format for each sampled sentence
print(sampled_indices)
for idx in sampled_indices:
    print("French:", test_fr_sentences[idx])
    print("Ground Truth English:", test_en_sentences[idx])
    print("Translated English:", test_pred[idx])
    print("------------------")
# End
```

```
[8023 4498 8365 8128 2181 3545 2530 6473 4809 1616]
French: les bonnes équipes envoient ces informations de façon à ce que les
joueurs puissent s'en servir.
Ground Truth English: the good teams stream it in a way that the players can
use.
Translated English:  the teams that are sending information information to the
service of the work they could be.
------------------
```

French: merci.
Ground Truth English: thank you.
Translated English: thank you.

French: il y a eu plusieurs cas où c'était vraiment juste.
Ground Truth English: there have been several close calls.
Translated English: there were many cases where it was very good.

French: mes prières vous accompagnent dans votre combat.
Ground Truth English: my prayers are with you for your fight.
Translated English: your prayers are now beaten up in your head.

French: et la question était : comment la technologie pourrait, les nouvelles technologies, y être ajoutée ?
Ground Truth English: and the question was: how could technology, new technology, be added to that?
Translated English: and the question was, how technology could be technology, new technologies?

French: combien d'entre vous ont vu l'ordinateur watson d'ibm gagner à jeopardy ?
Ground Truth English: i mean, how many of you saw the winning of jeopardy by ibm's watson?
Translated English: how many of you have seen the computer-wazi ratio of a table?

French: j'ai travaillé dans une mine de charbon -- dangereux.
Ground Truth English: i worked in a coal mine -- dangerous.
Translated English: i've been working on a coal mine -- a penguin.

French: n'importe qui d'autre l'aimerait aussi.
Ground Truth English: somebody else would love about this woman.
Translated English: all else would feel like it.

French: c'est tragique que les nord-coréens aient à cacher leurs identités et affronter tant de choses seulement pour survivre.
Ground Truth English: it's tragic that north koreans have to hide their identities and struggle so hard just to survive.
Translated English: it's bad for the prisoners of the syrian refugees and the obstacles to survive and to be able to stay in.

French: la glace que je photographie dans les icebergs est parfois très jeune -- deux milles ans.
Ground Truth English: some of the ice in the icebergs that i photograph is very young -- a couple thousand years old.
Translated English: the ice that i see in the earliest couple of weeks are very young, very, very different from the star.
----

[1]:
```
import evaluate


sacrebleu = evaluate.load('sacrebleu', cache_dir=dataset_path)
# Start your code here
# see https://huggingface.co/spaces/evaluate-metric/sacrebleu
# Note: please understand the format and meaning of references.
results = sacrebleu.compute(predictions=test_pred, references=test_en_sentences)
# End
# print(score)
score = results['score']
print(round(score, 2))
```

```
7.13
```

If you implement everything correctly, the BLEU score will be around 7.

1.5 Conclusion (5 Points)

Including but not limited to: translation example analysis (case study), bleu score analysis, model structure / parameter analysis, etc.

Answer:

Translation from French to English is carried out using a Seq2Seq model, both with and without attention mechanisms. Initially, the sentences are tokenized and encoded into token ids, then padded to ensure uniform length. To expedite training, a batch index sampler ensures batch sentences have similar lengths.

The Seq2Seq model without attention comprises an encoder with an embedding layer and a GRU layer, and a decoder with similar layers plus a dense layer. The decoder outputs probability distributions across the vocabulary, selecting the word with the highest probability per timestep. Padded tokens are ignored, and the loss is computed by comparing decoder outputs to targets.

When handling longer sentences, the Seq2Seq model without attention exhibits a tendency to repeat certain phrases. This stems from its heavy reliance on encoder hidden states for translation generation, lacking the capability to attend to crucial parts of the sentence. Consequently, translations may lack fluency and naturalness. In contrast, the Seq2Seq model with attention empowers the decoder to concentrate on significant segments of the sentence, enhancing the production of fluent and natural translations, particularly for longer sentences. Nonetheless, there remains a possibility of repetitive phrases even with the attention mechanism in place.

In conclusion, while the Seq2Seq model without attention demonstrates proficiency in translating shorter sentences, its limitations become apparent with longer sentences. The model's reliance on encoder hidden states often leads to repetitive phrases, resulting in translations that lack fluency and naturalness. Despite these drawbacks, it serves as a foundation for machine translation research, highlighting the importance of attention mechanisms in improving translation quality and addressing issues such as overfitting.

[ ]:
```

```