tkseem Documentation

Zaid Alyafeai, Maged Saeed

Aug 28, 2020

CONTENTS

CHAPTER

ONE

INSTALLATION

```
pip install tkseem
```

CHAPTER

TWO

TOKENIZERS

2.1 tkseem Package

2.1.1 Classes

CharacterTokenizer([unk_token, pad_token, ...])	Character based tokenization
DisjointLetterTokenizer([unk_token, ...])	Disjoint Letters based tokenization
MorphologicalTokenizer([unk_token, ...])	Auto tokenization using a saved dictionary
RandomTokenizer([unk_token, pad_token, ...])	Randomized based tokenization
SentencePieceTokenizer([unk_token, ...])	Sentencepiece based tokenization.
WordTokenizer([unk_token, pad_token, ...])	White space based tokenization

CharacterTokenizer

class tkseem.**CharacterTokenizer**(*unk_token='<UNK>'*, *pad_token='<PAD>'*, *vocab_size=10000*, *special_tokens=[]*)

Bases: tkseem._base.BaseTokenizer

Character based tokenization

Methods Summary

decode(encoded)	Decode ids
detokenize(tokens)	Convert tokens to a string
encode(text)	Convert string to a list of ids
encode_sentences(sentences[, boundries, ...])	Encode a list of sentences using the trained model
id_to_token(id)	convert id to token
load_model(file_path)	Load a saved model as a frequency dictionary
save_model(file_path)	Save a model as a freqency dictionary
token_to_id(piece)	Get tokens list
tokenize(text)	Tokenize using the frequency dictionary
train(file_path)	Train data using characters

Methods Documentation

decode(*encoded*)
Decode ids

Args: encoded (list): list of ids to decode

Returns: list: tokens

detokenize(*tokens*)
Convert tokens to a string

Args: tokens (list): list of tokens

Returns: str: detokenized string

encode(*text*)
Convert string to a list of ids

Args: text (str): input string

Returns: list: list of ids

encode_sentences(*sentences*, *boundries=''*, *''*, *out_length=None*)
Encode a list of sentences using the trained model

Args: sentences (list): list of sentences boundries (tuple): boundries for each sentence. out_length (int, optional): specify the max length of encodings. Defaults to 100.

Returns: [np.array]: numpy array of encodings

id_to_token(*id*)
convert id to token

Args: id (int): input id

Returns: str: token

load_model(*file_path*)
Load a saved model as a frequency dictionary

Args: file_path (str): file path of the dictionary

save_model(*file_path*)
Save a model as a freqency dictionary

Args: file_path (str): file path to save the model

token_to_id(*piece*)
Get tokens list

Returns: list: tokens

tokenize(*text*)
Tokenize using the frequency dictionary

Args: text (str): input string

Returns: list: generated tokens

train(*file_path*)
Train data using characters

Args: file_path (str): file to train

DisjointLetterTokenizer

class tkseem.**DisjointLetterTokenizer**(*unk_token='<UNK>'*, *pad_token='<PAD>'*, *vocab_size=10000*, *special_tokens=[]*)

Bases: tkseem._base.BaseTokenizer

Disjoint Letters based tokenization

Methods Summary

decode(encoded)	Decode ids
detokenize(tokens)	Convert tokens to a string
encode(text)	Convert string to a list of ids
encode_sentences(sentences[, boundries, ...])	Encode a list of sentences using the trained model
id_to_token(id)	convert id to token
load_model(file_path)	Load a saved model as a frequency dictionary
save_model(file_path)	Save a model as a freqency dictionary
token_to_id(piece)	Get tokens list
tokenize(text[, use_cache, max_cache_size])	Args:
train(file_path)	Train data using disjoint letters

Methods Documentation

decode(*encoded*)

Decode ids

Args: encoded (list): list of ids to decode

Returns: list: tokens

detokenize(*tokens*)

Convert tokens to a string

Args: tokens (list): list of tokens

Returns: str: detokenized string

encode(*text*)

Convert string to a list of ids

Args: text (str): input string

Returns: list: list of ids

encode_sentences(*sentences*, *boundries=''*, *''*, *out_length=None*)

Encode a list of sentences using the trained model

Args: sentences (list): list of sentences boundries (tuple): boundries for each sentence. out_length (int, optional): specify the max length of encodings. Defaults to 100.

Returns: [np.array]: numpy array of encodings

id_to_token(*id*)

convert id to token

Args: id (int): input id

Returns: str: token

load_model(*file_path*)
Load a saved model as a frequency dictionary

Args: file_path (str): file path of the dictionary

save_model(*file_path*)
Save a model as a freqency dictionary

Args: file_path (str): file path to save the model

token_to_id(*piece*)
Get tokens list

Returns: list: tokens

tokenize(*text*, *use_cache=False*, *max_cache_size=1000*)

Args: text (str): input text use_cache (bool, optional): speed up using caching. Defaults to False. max_cache_size (int, optional): max cacne size. Defaults to 1000.

Returns: list: output list of tokens

train(*file_path*)
Train data using disjoint letters

Args: file_path (str): file to train

MorphologicalTokenizer

class tkseem.**MorphologicalTokenizer**(*unk_token='<UNK>'*, *pad_token='<PAD>'*, *vocab_size=10000*, *special_tokens=[]*)
Bases: tkseem._base.BaseTokenizer

Auto tokenization using a saved dictionary

Methods Summary

decode(encoded)	Decode ids
detokenize(tokens)	Convert tokens to a string
encode(text)	Convert string to a list of ids
encode_sentences(sentences[, boundries, ...])	Encode a list of sentences using the trained model
id_to_token(id)	convert id to token
load_model(file_path)	Load a saved model as a frequency dictionary
save_model(file_path)	Save a model as a freqency dictionary
token_to_id(piece)	Get tokens list
tokenize(text[, use_cache, max_cache_size])	Args:
train()	Use a default dictionary for training

Methods Documentation

decode(*encoded*)
Decode ids

Args: encoded (list): list of ids to decode

Returns: list: tokens

detokenize(*tokens*)
Convert tokens to a string

Args: tokens (list): list of tokens

Returns: str: detokenized string

encode(*text*)
Convert string to a list of ids

Args: text (str): input string

Returns: list: list of ids

encode_sentences(*sentences*, *boundries=''*, *''*, *out_length=None*)
Encode a list of sentences using the trained model

Args: sentences (list): list of sentences boundries (tuple): boundries for each sentence. out_length (int, optional): specify the max length of encodings. Defaults to 100.

Returns: [np.array]: numpy array of encodings

id_to_token(*id*)
convert id to token

Args: id (int): input id

Returns: str: token

load_model(*file_path*)
Load a saved model as a frequency dictionary

Args: file_path (str): file path of the dictionary

save_model(*file_path*)
Save a model as a freqency dictionary

Args: file_path (str): file path to save the model

token_to_id(*piece*)
Get tokens list

Returns: list: tokens

tokenize(*text*, *use_cache=False*, *max_cache_size=1000*)

Args: text (str): input text use_cache (bool, optional): speed up using caching. Defaults to False. max_cache_size (int, optional): max cacne size. Defaults to 1000.

Returns: list: output list of tokens

train()
Use a default dictionary for training

RandomTokenizer

class tkseem.**RandomTokenizer**(*unk_token='<UNK>'*, *pad_token='<PAD>'*, *vocab_size=10000*, *special_tokens=[]*)

Bases: tkseem._base.BaseTokenizer

Randomized based tokenization

Methods Summary

decode(encoded)	Decode ids
detokenize(tokens)	Convert tokens to a string
encode(text)	Convert string to a list of ids
encode_sentences(sentences[, boundries, ...])	Encode a list of sentences using the trained model
id_to_token(id)	convert id to token
load_model(file_path)	Load a saved model as a frequency dictionary
save_model(file_path)	Save a model as a freqency dictionary
token_to_id(piece)	Get tokens list
tokenize(text[, use_cache, max_cache_size])	Args:
train(file_path)	Train data using randomly splitted subwords

Methods Documentation

decode(*encoded*)

Decode ids

Args: encoded (list): list of ids to decode

Returns: list: tokens

detokenize(*tokens*)

Convert tokens to a string

Args: tokens (list): list of tokens

Returns: str: detokenized string

encode(*text*)

Convert string to a list of ids

Args: text (str): input string

Returns: list: list of ids

encode_sentences(*sentences*, *boundries=''*, *''*, *out_length=None*)

Encode a list of sentences using the trained model

Args: sentences (list): list of sentences boundries (tuple): boundries for each sentence. out_length (int, optional): specify the max length of encodings. Defaults to 100.

Returns: [np.array]: numpy array of encodings

id_to_token(*id*)

convert id to token

Args: id (int): input id

Returns: str: token

load_model(*file_path*)
Load a saved model as a frequency dictionary

Args: file_path (str): file path of the dictionary

save_model(*file_path*)
Save a model as a freqency dictionary

Args: file_path (str): file path to save the model

token_to_id(*piece*)
Get tokens list

Returns: list: tokens

tokenize(*text*, *use_cache=False*, *max_cache_size=1000*)

Args: text (str): input text use_cache (bool, optional): speed up using caching. Defaults to False. max_cache_size (int, optional): max cacne size. Defaults to 1000.

Returns: list: output list of tokens

train(*file_path*)
Train data using randomly splitted subwords

Args: file_path (str): file to train

SentencePieceTokenizer

class tkseem.**SentencePieceTokenizer**(*unk_token='<UNK>'*, *pad_token='<PAD>'*, *vocab_size=10000*, *special_tokens=[]*)
Bases: tkseem._base.BaseTokenizer

Sentencepiece based tokenization.

Methods Summary

decode(encoded)	Decode ids
detokenize(tokens)	Convert tokens to a string
encode(text)	Convert string to a list of ids
encode_sentences(sentences[, boundries, ...])	Encode a list of sentences using the trained model
id_to_token(id)	convert id to token
load_model(file_path)	Load a saved sp model
save_model(file_path)	Save a model as a freqency dictionary
token_to_id(token)	Get tokens list
tokenize(text)	Tokenize using the frequency dictionary
train(file_path[, model_type])	Train using sentence piece

Methods Documentation

decode(*encoded*)
Decode ids

Args: encoded (list): list of ids to decode

Returns: list: tokens

detokenize(*tokens*)
Convert tokens to a string

Args: tokens (list): list of tokens

Returns: str: detokenized string

encode(*text*)
Convert string to a list of ids

Args: text (str): input string

Returns: list: list of ids

encode_sentences(*sentences*, *boundries='', ''*, *out_length=None*)
Encode a list of sentences using the trained model

Args: sentences (list): list of sentences boundries (tuple): boundries for each sentence. out_length (int, optional): specify the max length of encodings. Defaults to 100.

Returns: [np.array]: numpy array of encodings

id_to_token(*id*)
convert id to token

Args: id (int): input id

Returns: str: token

load_model(*file_path*)
Load a saved sp model

Args: file_path (str): file path of the trained model

save_model(*file_path*)
Save a model as a freqency dictionary

Args: file_path (str): file path to save the model

token_to_id(*token*)
Get tokens list

Returns: list: tokens

tokenize(*text*)
Tokenize using the frequency dictionary

Args: text (str): input string

Returns: list: generated tokens

train(*file_path*, *model_type='bpe'*)
Train using sentence piece

Args: file_path (str): file to train model_type (str, optional): train using sp. Defaults to “bpe”.

WordTokenizer

class tkseem.**WordTokenizer**(*unk_token='<UNK>'*, *pad_token='<PAD>'*, *vocab_size=10000*, *special_tokens=[]*)

Bases: tkseem._base.BaseTokenizer

White space based tokenization

Attributes Summary

tokens_frequency

Methods Summary

decode(encoded)	Decode ids
detokenize(tokens)	Convert tokens to a string
encode(text)	Convert string to a list of ids
encode_sentences(sentences[, boundries, ...])	Encode a list of sentences using the trained model
id_to_token(id)	convert id to token
load_model(file_path)	Load a saved model as a frequency dictionary
save_model(file_path)	Save a model as a freqency dictionary
token_to_id(piece)	Get tokens list
tokenize(text)	Tokenize using the frequency dictionary
train(file_path)	Train using words' frequency

Attributes Documentation

tokens_frequency = None

Methods Documentation

decode(*encoded*)
Decode ids

Args: encoded (list): list of ids to decode

Returns: list: tokens

detokenize(*tokens*)
Convert tokens to a string

Args: tokens (list): list of tokens

Returns: str: detokenized string

encode(*text*)
Convert string to a list of ids

Args: text (str): input string

Returns: list: list of ids

encode_sentences(*sentences*, *boundries=''*, *''*, *out_length=None*)
Encode a list of sentences using the trained model

Args: sentences (list): list of sentences boundries (tuple): boundries for each sentence. out_length (int, optional): specify the max length of encodings. Defaults to 100.

Returns: [np.array]: numpy array of encodings

id_to_token(*id*)
convert id to token

Args: id (int): input id

Returns: str: token

load_model(*file_path*)
Load a saved model as a frequency dictionary

Args: file_path (str): file path of the dictionary

save_model(*file_path*)
Save a model as a freqency dictionary

Args: file_path (str): file path to save the model

token_to_id(*piece*)
Get tokens list

Returns: list: tokens

tokenize(*text*)
Tokenize using the frequency dictionary

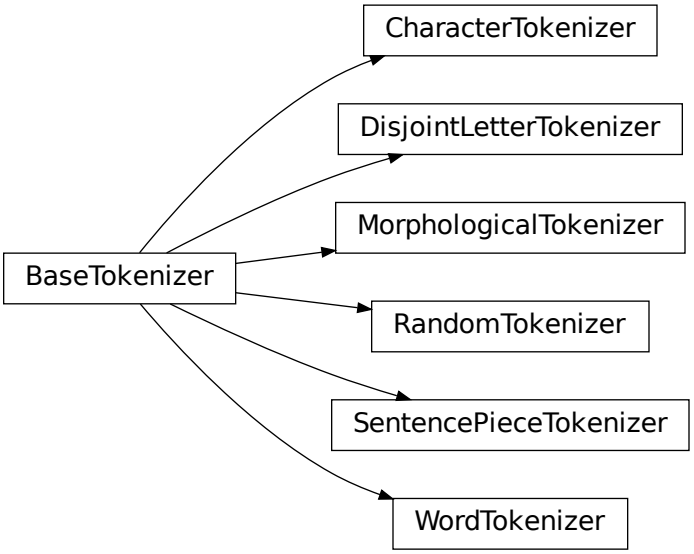
Args: text (str): input string

Returns: list: generated tokens

train(*file_path*)
Train using words' frequency

Args: file_path (str): file to train

2.1.2 Class Inheritance Diagram



CHAPTER

THREE

DOCS

[1]:
```
#!pip3 install tkseem
```

3.1 Frequency Tokenizer

[2]:
```
import tkseem as tk
```

Read, preprocess then train

[3]:
```
tokenizer = tk.WordTokenizer()
tokenizer.train('samples/data.txt')
```

```
Training WordTokenizer ...
```

[4]:
```
print(tokenizer)
```

```
WordTokenizer
```

Tokenize

[5]:
```
tokenizer.tokenize(" ")
```

[5]:
```
['', '']
```

Encode as ids

[6]:
```
encoded = tokenizer.encode(" ")
print(encoded)
```

```
[557, 798]
```

Decode back to tokens

[7]:
```
decoded = tokenizer.decode(encoded)
print(decoded)
```

```
['', '']
```

[8]:
```
detokenized = tokenizer.detokenize(decoded)
print(detokenized)
```

3.2 SentencePiece Tokenizer

Read, preprocess then train

[9]:
```
tokenizer = tk.SentencePieceTokenizer()
tokenizer.train('samples/data.txt')
```

```
Training SentencePiece ...
```

Tokenize

[10]:
```
tokenizer.tokenize("   ")
```

[10]:
```
['', '', '', '', '', '']
```

Encode as ids

[11]:
```
encoded = tokenizer.encode(" ")
print(encoded)
```

```
[1799, 2741]
```

Decode back to tokens

[12]:
```
decoded = tokenizer.decode(encoded)
print(decoded)
```

```
['', '']
```

[13]:
```
detokenized = tokenizer.detokenize(decoded)
print(detokenized)
```

```

```

3.3 Morphological Tokenizer

Read, preprocess then train

[14]:
```
tokenizer = tk.MorphologicalTokenizer()
tokenizer.train()
```

```
Training MorphologicalTokenizer ...
```

Tokenize

[15]:
```
tokenizer.tokenize(" ")
```

[15]:
```
['', '##', '', '##']
```

Encode as ids

[16]:
```
encoded = tokenizer.encode(" ")
print(encoded)
```

```
[2, 367, 764, 184]
```

Decode back to tokens

[17]:
```
decoded = tokenizer.decode(encoded)
print(decoded)
```

```
['', '##', '', '##']
```

3.4 Random Tokenizer

[18]:
```
tokenizer = tk.RandomTokenizer()
tokenizer.train('samples/data.txt')
```

```
Training RandomTokenizer ...
```

[19]:
```
tokenizer.tokenize("   ")
```

[19]:
```
['', '##', '', '##', '', '##', '', '##', '##']
```

3.5 Disjoint Letter Tokenizer

[20]:
```
tokenizer = tk.DisjointLetterTokenizer()
tokenizer.train('samples/data.txt')
```

```
Training DisjointLetterTokenizer ...
```

[21]:
```
print(tokenizer.tokenize("   "))
```

```
['', '##', '##', '', '', '##', '', '##', '##', '##', '##']
```

3.6 Character Tokenizer

[22]:
```
tokenizer = tk.CharacterTokenizer()
tokenizer.train('samples/data.txt')
```

```
Training CharacterTokenizer ...
```

[23]:
```
tokenizer.tokenize(" ")
```

[23]:
```
['', '##', '##', '##', '##', '##', '', '##', '##', '##', '##']
```

3.7 Export Models

Models can be saved for deployment and reloading.

[24]:
```
tokenizer = tk.WordTokenizer()
tokenizer.train('samples/data.txt')
tokenizer.save_model('freq.pl')
```

```
Training WordTokenizer ...
Saving as pickle file ...
```

load model without pretraining

[25]:
```
tokenizer = tk.WordTokenizer()
tokenizer.load_model('freq.pl')
```

```
Loading as pickle file ...
```

[26]:
```
tokenizer.tokenize(' ')
```

[26]:
```
['', '']
```

3.8 Benchmarking

Comparing tokenizers in terms of training time

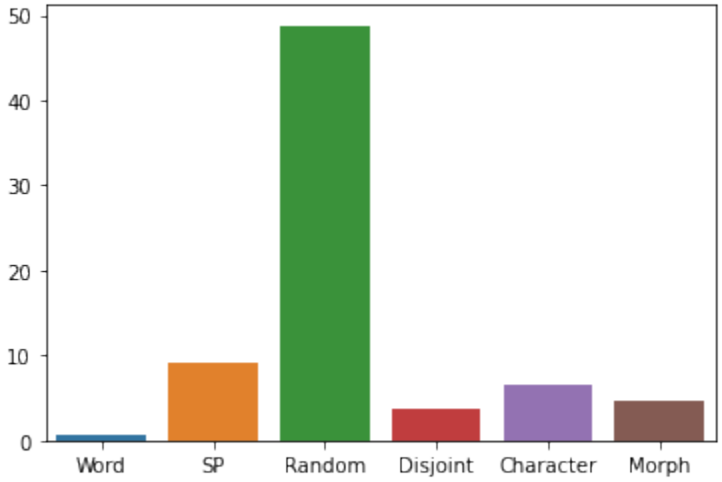
[27]:
```
import seaborn as sns
import pandas as pd
import time

def calc_time(fun):
    tokenizer = fun()
    start_time = time.time()
    # morph tokenizer doesn't take arguments
    if str(tokenizer) == 'MorphologicalTokenizer':
        tokenizer.train()
    else:
        tokenizer.train('samples/data.txt')
    return time.time() - start_time

running_times = {}

running_times['Word'] = calc_time(tk.WordTokenizer)
running_times['SP'] = calc_time(tk.SentencePieceTokenizer)
running_times['Random'] = calc_time(tk.RandomTokenizer)
running_times['Disjoint'] = calc_time(tk.DisjointLetterTokenizer)
running_times['Character'] = calc_time(tk.CharacterTokenizer)
running_times['Morph'] = calc_time(tk.MorphologicalTokenizer)
plt = sns.barplot(data = pd.DataFrame.from_dict([running_times]))
```

```
Training WordTokenizer ...
Training SentencePiece ...
Training RandomTokenizer ...
Training DisjointLetterTokenizer ...
Training CharacterTokenizer ...
Training MorphologicalTokenizer ...
```



comparing tokenizers in tokenization time

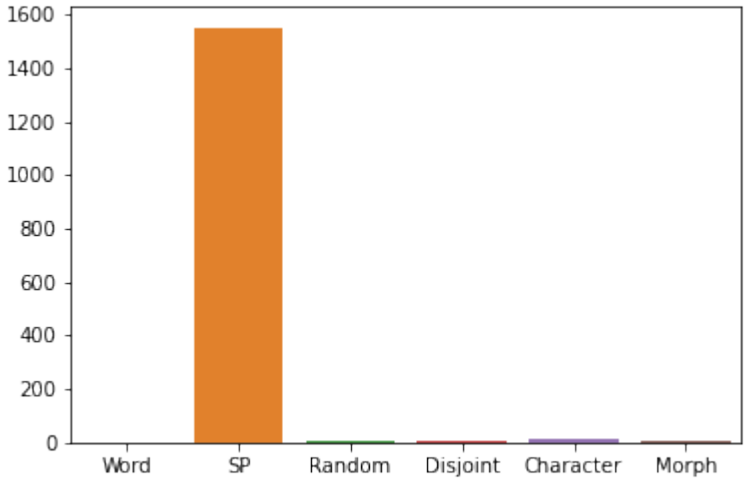
[28]:
```
import seaborn as sns
import pandas as pd
import time

def calc_time(fun):
    tokenizer = fun()
    # morph tokenizer doesn't take arguments
    if str(tokenizer) == 'MorphologicalTokenizer':
        tokenizer.train()
    else:
        tokenizer.train('samples/data.txt')
    start_time = time.time()
    tokenizer.tokenize(open('samples/data.txt', 'r').read())
    return time.time() - start_time

running_times = {}

running_times['Word'] = calc_time(tk.WordTokenizer)
running_times['SP'] = calc_time(tk.SentencePieceTokenizer)
running_times['Random'] = calc_time(tk.RandomTokenizer)
running_times['Disjoint'] = calc_time(tk.DisjointLetterTokenizer)
running_times['Character'] = calc_time(tk.CharacterTokenizer)
running_times['Morph'] = calc_time(tk.MorphologicalTokenizer)
plt = sns.barplot(data = pd.DataFrame.from_dict([running_times]))
```

```
Training WordTokenizer ...
Training SentencePiece ...
Training RandomTokenizer ...
Training DisjointLetterTokenizer ...
Training CharacterTokenizer ...
Training MorphologicalTokenizer ...
```



3.9 Caching

Caching is used for speeding up the tokenization process.

[32]:
```
import tkseem as tk
tokenizer = tk.MorphologicalTokenizer()
tokenizer.train()
```

```
Training MorphologicalTokenizer ...
```

[33]:
```
%%timeit
out = tokenizer.tokenize(open('samples/data.txt', 'r').read(), use_cache = False)
```

```
8.82 s ± 277 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

[34]:
```
%%timeit
out = tokenizer.tokenize(open('samples/data.txt', 'r').read(), use_cache = True, max_
↪cache_size = 10000)
```

```
7.14 s ± 296 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

CHAPTER

FOUR

SENTIMENT ANALYSIS

[ ]:
```
!pip install tkseem
!pip install tnkeeh
```

[ ]:
```
!wget https://raw.githubusercontent.com/ARBML/tkseem/master/tasks/sentiment_analysis/
↪sentiment/data.txt
!wget https://raw.githubusercontent.com/ARBML/tkseem/master/tasks/sentiment_analysis/
↪sentiment/labels.txt
```

4.1 Imports

[3]:
```
import numpy as np
import tkseem as tk
import tnkeeh as tn
from tensorflow.keras.models import Sequential
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import GRU, Embedding, Dense, Input, Dropout,␣
↪Bidirectional
```

4.2 Process data

[4]:
```
tn.clean_data(file_path = 'sentiment/data.txt', save_path = 'sentiment/cleaned_data.
↪txt', remove_diacritics=True,
      excluded_chars=['!', '.', '?'])
tn.split_classification_data('sentiment/cleaned_data.txt', 'sentiment/labels.txt')
train_data, test_data, train_lbls, test_lbls = tn.read_data(mode = 1)
```

```
Remove diacritics
Remove Tatweel
Saving to sentiment/cleaned_data.txt
Split data
Save to data
Read data  ['test_data.txt', 'test_lbls.txt', 'train_data.txt', 'train_lbls.txt']
```

[5]:
```
max_length = max(len(data) for data in train_data)
```

4.3 Tokenize

[6]:
```
tokenizer = tk.SentencePieceTokenizer()
tokenizer.train('data/train_data.txt')
```

```
Training SentencePiece ...
```

4.4 Tokenize data

[7]:
```
def preprocess(tokenizer, data, labels):
    X = tokenizer.encode_sentences(data)
    y = np.array([int(lbl) for lbl in labels])
    return X, y
```

[8]:
```
# process training data
X_train, y_train = preprocess(tokenizer, train_data, train_lbls)

# process test data
X_test, y_test = preprocess(tokenizer, test_data, test_lbls)
```

4.5 Model

[9]:
```
model = Sequential()
model.add(Embedding(tokenizer.vocab_size, 32))
model.add(Bidirectional(GRU(units = 32)))
model.add(Dense(32, activation = 'tanh'))
model.add(Dropout(0.3))
model.add(Dense(1, activation = 'sigmoid'))
model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy
↪'])
```

4.6 Train

[10]:
```
history = model.fit(X_train, y_train, epochs = 12, validation_split = 0.1,  batch_
↪size= 128, shuffle = True)
```

```
Epoch 1/12
6/6 [==============================] - 3s 445ms/step - loss: 0.6936 - accuracy: 0.
↪4986 - val_loss: 0.6990 - val_accuracy: 0.3625
Epoch 2/12
6/6 [==============================] - 2s 324ms/step - loss: 0.6883 - accuracy: 0.
↪5097 - val_loss: 0.6986 - val_accuracy: 0.3625
Epoch 3/12
6/6 [==============================] - 1s 193ms/step - loss: 0.6827 - accuracy: 0.
↪6139 - val_loss: 0.6890 - val_accuracy: 0.5875
Epoch 4/12
6/6 [==============================] - 2s 254ms/step - loss: 0.6706 - accuracy: 0.
↪8222 - val_loss: 0.6814 - val_accuracy: 0.6625
Epoch 5/12
```

(continues on next page)

(continued from previous page)

```
6/6 [==============================] - 1s 238ms/step - loss: 0.6473 - accuracy: 0.
↪8861 - val_loss: 0.6730 - val_accuracy: 0.6875
Epoch 6/12
6/6 [==============================] - 1s 214ms/step - loss: 0.6117 - accuracy: 0.
↪9014 - val_loss: 0.6543 - val_accuracy: 0.7125
Epoch 7/12
6/6 [==============================] - 2s 266ms/step - loss: 0.5536 - accuracy: 0.
↪9167 - val_loss: 0.6210 - val_accuracy: 0.7500
Epoch 8/12
6/6 [==============================] - 1s 237ms/step - loss: 0.4579 - accuracy: 0.
↪9347 - val_loss: 0.5906 - val_accuracy: 0.7500
Epoch 9/12
6/6 [==============================] - 1s 197ms/step - loss: 0.3353 - accuracy: 0.
↪9500 - val_loss: 0.5605 - val_accuracy: 0.7375
Epoch 10/12
6/6 [==============================] - 1s 219ms/step - loss: 0.2050 - accuracy: 0.
↪9639 - val_loss: 0.5069 - val_accuracy: 0.7625
Epoch 11/12
6/6 [==============================] - 1s 216ms/step - loss: 0.1315 - accuracy: 0.
↪9694 - val_loss: 0.5215 - val_accuracy: 0.7250
Epoch 12/12
6/6 [==============================] - 1s 166ms/step - loss: 0.1063 - accuracy: 0.
↪9625 - val_loss: 0.5699 - val_accuracy: 0.7125
```

4.7 Test

[11]:
```
def classify(sentence):
  sequence = tokenizer.encode_sentences([sentence], out_length = max_length)[0]
  pred = model.predict(sequence)[0][0]
  print(pred)
```

[12]:
```
classify("  ")
classify(" ")
```

```
0.06951779
0.89656436
```

CHAPTER

FIVE

POETRY CLASSIFICATION

[1]:
```
!pip install tkseem
!pip install tnkeeh
```

```
/bin/bash: pip: command not found
/bin/bash: pip: command not found
```

[ ]:
```
!wget https://raw.githubusercontent.com/ARBML/tkseem/master/tasks/meter_
↪classification/meters/data.txt
!wget https://raw.githubusercontent.com/ARBML/tkseem/master/tasks/meter_
↪classification/meters/labels.txt
```

5.1 Imports

[3]:
```
import tensorflow as tf
import tkseem as tk
import tnkeeh as tn
import numpy as np
from tensorflow.keras.layers import GRU, Embedding, Dense, Input, Dropout,␣
↪Bidirectional, BatchNormalization, Flatten, Reshape
from tensorflow.keras.models import Sequential
from sklearn.model_selection import train_test_split
```

5.2 Process data

[4]:
```
tn.clean_data(file_path = 'meters/data.txt', save_path = 'meters/cleaned_data.txt',␣
↪remove_diacritics=True,
      excluded_chars=['!', '.', '?', '#'])
tn.split_classification_data('meters/cleaned_data.txt', 'meters/labels.txt')
train_data, test_data, train_lbls, test_lbls = tn.read_data(mode = 1)
```

```
Remove diacritics
Remove Tatweel
Saving to meters/cleaned_data.txt
Split data
Save to data
Read data  ['test_data.txt', 'test_lbls.txt', 'train_data.txt', 'train_lbls.txt']
```

5.3 Tokenization

[5]:
```
tokenizer = tk.CharacterTokenizer()
tokenizer.train('data/train_data.txt')
```

```
Training CharacterTokenizer ...
```

5.4 Tokenize data

[6]:
```
def preprocess(tokenizer, data, labels):
    X = tokenizer.encode_sentences(data)
    y = np.array([int(lbl) for lbl in labels])
    return X, y
```

[7]:
```
# process training data
X_train, y_train = preprocess(tokenizer, train_data, train_lbls)

# process test data
X_test, y_test = preprocess(tokenizer, test_data, test_lbls)
```

[8]:
```
max_length = max(len(sent) for sent in X_train)
```

5.5 Model

[9]:
```
model = Sequential()
model.add(Input((max_length,)))
model.add(Embedding(tokenizer.vocab_size, 256))
model.add(Bidirectional(GRU(units = 256, return_sequences=True)))
model.add(Bidirectional(GRU(units = 256, return_sequences=True)))
model.add(Bidirectional(GRU(units = 256)))
model.add(Dense(128, activation = 'relu'))
model.add(Dropout(0.3))
model.add(Dense(14, activation = 'softmax'))
model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', metrics =␣
↪['accuracy'])
```

[10]:
```
model.fit(X_train, y_train, validation_split = 0.1, epochs = 10, batch_size= 256,␣
↪shuffle = True)
```

```
Epoch 1/10
133/133 [==============================] - 465s 3s/step - loss: 2.3899 - accuracy: 0.
↪1572 - val_loss: 1.9431 - val_accuracy: 0.2902
Epoch 2/10
133/133 [==============================] - 452s 3s/step - loss: 1.8384 - accuracy: 0.
↪3214 - val_loss: 1.6722 - val_accuracy: 0.3905
Epoch 3/10
133/133 [==============================] - 436s 3s/step - loss: 1.5614 - accuracy: 0.
↪4314 - val_loss: 1.5018 - val_accuracy: 0.4581
Epoch 4/10
133/133 [==============================] - 381s 3s/step - loss: 1.1860 - accuracy: 0.
↪5879 - val_loss: 0.8718 - val_accuracy: 0.7109
```

(continues on next page)

(continued from previous page)

```
Epoch 5/10
133/133 [==============================] - 370s 3s/step - loss: 0.7501 - accuracy: 0.
↪7595 - val_loss: 0.5991 - val_accuracy: 0.8085
Epoch 6/10
133/133 [==============================] - 360s 3s/step - loss: 0.5233 - accuracy: 0.
↪8410 - val_loss: 0.5352 - val_accuracy: 0.8332
Epoch 7/10
133/133 [==============================] - 361s 3s/step - loss: 0.4070 - accuracy: 0.
↪8807 - val_loss: 0.4281 - val_accuracy: 0.8708
Epoch 8/10
133/133 [==============================] - 355s 3s/step - loss: 0.3229 - accuracy: 0.
↪9074 - val_loss: 0.3947 - val_accuracy: 0.8841
Epoch 9/10
133/133 [==============================] - 356s 3s/step - loss: 0.2724 - accuracy: 0.
↪9241 - val_loss: 0.3725 - val_accuracy: 0.8926
Epoch 10/10
133/133 [==============================] - 355s 3s/step - loss: 0.2301 - accuracy: 0.
↪9352 - val_loss: 0.3540 - val_accuracy: 0.8989
```

[10]:
```
<tensorflow.python.keras.callbacks.History at 0x7ff3a7692160>
```

5.6 Test

[11]:
```
label2name = ['', '', '', '', '', '',
              '', '', '', '', '', '', '', '']
```

[14]:
```
def classify(sentence):
    sequence = tokenizer.encode_sentences([sentence], out_length = max_length)
    pred = model.predict(sequence)[0]
    print(label2name[np.argmax(pred, 0).astype('int')], np.max(pred))
```

[15]:
```
classify("     #     ")
classify("    #    ")
classify("       #     ")
classify("      #    ")
classify("    #    ")
classify("   #    ")
classify("   #  ")
classify("    #   ")
classify("     #     ")
classify("     #   ")
```

```
 0.9957462
 0.98703927
 0.9792284
 0.99692947
 0.94578993
 0.3755584
 0.981885
 0.8000305
 0.7176092
 0.99850094
```

```
[ ]: # modified version from https://www.tensorflow.org/tutorials/text/nmt_with_attention
```

CHAPTER

SIX

TRANLSATION

[4]:
```
!wget https://raw.githubusercontent.com/ARBML/tkseem/master/tasks/translation/data/ar_
↪data.txt
!wget https://raw.githubusercontent.com/ARBML/tkseem/master/tasks/translation/data/en_
↪data.txt
```

```
Will not apply HSTS. The HSTS database must be a regular and non-world-writable file.
ERROR: could not open HSTS store at '/home/zaid/.wget-hsts'. HSTS will be disabled.
--2020-08-28 14:49:14--  https://raw.githubusercontent.com/ARBML/tkseem/master/tasks/
↪translation/data/ar_data.txt
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.112.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.112.133|:
↪443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3705050 (3.5M) [text/plain]
Saving to: 'ar_data.txt'

ar_data.txt         100%[===================>]   3.53M   719KB/s    in 5.1s

2020-08-28 14:49:21 (708 KB/s) - 'ar_data.txt' saved [3705050/3705050]

Will not apply HSTS. The HSTS database must be a regular and non-world-writable file.
ERROR: could not open HSTS store at '/home/zaid/.wget-hsts'. HSTS will be disabled.
--2020-08-28 14:49:21--  https://raw.githubusercontent.com/ARBML/tkseem/master/tasks/
↪translation/data/en_data.txt
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.112.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.112.133|:
↪443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2510593 (2.4M) [text/plain]
Saving to: 'en_data.txt'

en_data.txt         100%[===================>]   2.39M   588KB/s    in 4.2s

2020-08-28 14:49:26 (588 KB/s) - 'en_data.txt' saved [2510593/2510593]
```

[ ]:
```
!pip install tkseem
!pip install tnkeeh
```

[1]:
```
import re
import nltk
import time
import numpy as np
```

(continues on next page)

(continued from previous page)

```
import tkseem as tk
import tnkeeh as tn
import tensorflow as tf
import matplotlib.ticker as ticker
import matplotlib.pyplot as plt
```

6.1 Data Preprocessing

[5]:
```
tn.clean_data('ar_data.txt','ar_clean_data.txt', remove_diacritics=True)
tn.clean_data('en_data.txt','en_clean_data.txt')

tn.split_parallel_data('ar_clean_data.txt', 'en_clean_data.txt', split_ratio=0.3)
train_inp_text, train_tar_text, test_inp_text, test_tar_text = tn.read_data(mode = 2)
```

```
Remove diacritics
Remove Tatweel
Saving to ar_clean_data.txt
Remove Tatweel
Saving to en_clean_data.txt
Split data
Save to data
Read data  ['ar_data.txt', 'en_data.txt', 'test_inp_data.txt', 'test_tar_data.txt',
↪'train_inp_data.txt', 'train_tar_data.txt']
```

6.2 Tokenization

[6]:
```
ar_tokenizer = tk.SentencePieceTokenizer(special_tokens=['<s>', '</s>'])
ar_tokenizer.train('data/train_inp_data.txt')

en_tokenizer = tk.SentencePieceTokenizer(special_tokens=['<s>', '</s>'])
en_tokenizer.train('data/train_tar_data.txt')

train_inp_data = ar_tokenizer.encode_sentences(train_inp_text, boundries = ('<s>', '</
↪s>'))
train_tar_data = en_tokenizer.encode_sentences(train_tar_text, boundries = ('<s>', '</
↪s>'))
```

```
Training SentencePiece ...
Training SentencePiece ...
```

6.3 Create Dataset

[7]:
```
BATCH_SIZE = 64
BUFFER_SIZE = len(train_inp_data)

dataset = tf.data.Dataset.from_tensor_slices((train_inp_data, train_tar_data)).
↪shuffle(BUFFER_SIZE)
dataset = dataset.batch(BATCH_SIZE, drop_remainder=True)
```

6.4 Encoder, Decoder

[8]:
```
class Encoder(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, enc_units, batch_sz):
        super(Encoder, self).__init__()
        self.batch_sz = batch_sz
        self.enc_units = enc_units
        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
        self.gru = tf.keras.layers.GRU(self.enc_units,
                                       return_sequences=True,
                                       return_state=True,
                                       recurrent_initializer='glorot_uniform')

    def call(self, x, hidden):
        x = self.embedding(x)
        output, state = self.gru(x, initial_state = hidden)
        return output, state

    def initialize_hidden_state(self):
        return tf.zeros((self.batch_sz, self.enc_units))

class BahdanauAttention(tf.keras.layers.Layer):
    def __init__(self, units):
        super(BahdanauAttention, self).__init__()
        self.W1 = tf.keras.layers.Dense(units)
        self.W2 = tf.keras.layers.Dense(units)
        self.V = tf.keras.layers.Dense(1)

    def call(self, query, values):
        # query hidden state shape == (batch_size, hidden size)
        # query_with_time_axis shape == (batch_size, 1, hidden size)
        # values shape == (batch_size, max_len, hidden size)
        # we are doing this to broadcast addition along the time axis to calculate␣
↪the score
        query_with_time_axis = tf.expand_dims(query, 1)

        # score shape == (batch_size, max_length, 1)
        # we get 1 at the last axis because we are applying score to self.V
        # the shape of the tensor before applying self.V is (batch_size, max_length,␣
↪units)
        score = self.V(tf.nn.tanh(
            self.W1(query_with_time_axis) + self.W2(values)))

        # attention_weights shape == (batch_size, max_length, 1)
        attention_weights = tf.nn.softmax(score, axis=1)

        # context_vector shape after sum == (batch_size, hidden_size)
        context_vector = attention_weights * values
        context_vector = tf.reduce_sum(context_vector, axis=1)

        return context_vector, attention_weights

class Decoder(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, dec_units, batch_sz):
        super(Decoder, self).__init__()
        self.batch_sz = batch_sz
        self.dec_units = dec_units
```

(continues on next page)

(continued from previous page)

```
        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
        self.gru = tf.keras.layers.GRU(self.dec_units,
                                       return_sequences=True,
                                       return_state=True,
                                       recurrent_initializer='glorot_uniform')
        self.fc = tf.keras.layers.Dense(vocab_size)

        # used for attention
        self.attention = BahdanauAttention(self.dec_units)

    def call(self, x, hidden, enc_output):
        # enc_output shape == (batch_size, max_length, hidden_size)
        context_vector, attention_weights = self.attention(hidden, enc_output)

        # x shape after passing through embedding == (batch_size, 1, embedding_dim)
        x = self.embedding(x)

        # x shape after concatenation == (batch_size, 1, embedding_dim + hidden_size)
        x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)

        # passing the concatenated vector to the GRU
        output, state = self.gru(x)

        # output shape == (batch_size * 1, hidden_size)
        output = tf.reshape(output, (-1, output.shape[2]))

        # output shape == (batch_size, vocab)
        x = self.fc(output)

        return x, state, attention_weights



def get_loss_object():
    return  tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True, reduction=
↪'none')

def loss_function(real, pred):
    mask = tf.math.logical_not(tf.math.equal(real, 1))
    loss_ = get_loss_object()(real, pred)

    mask = tf.cast(mask, dtype=loss_.dtype)
    loss_ *= mask

    return tf.reduce_mean(loss_)
```

Initialize models

[9]:
```
units = 1024
embedding_dim = 256
max_length_inp = train_inp_data.shape[1]
max_length_tar = train_tar_data.shape[1]
steps_per_epoch = len(train_inp_data)//BATCH_SIZE
vocab_inp_size = ar_tokenizer.vocab_size
vocab_tar_size = en_tokenizer.vocab_size

encoder = Encoder(vocab_inp_size, embedding_dim, units, BATCH_SIZE)
```

(continues on next page)

(continued from previous page)

```
decoder = Decoder(vocab_tar_size, embedding_dim, units, BATCH_SIZE)
```

6.5 Training Procedure

[10]:
```
@tf.function
def train_step(inp, targ, enc_hidden, encoder, decoder, optimizer, en_tokenizer):
    loss = 0

    with tf.GradientTape() as tape:
        enc_output, enc_hidden = encoder(inp, enc_hidden)

        dec_hidden = enc_hidden

        dec_input = tf.expand_dims([en_tokenizer.token_to_id('<s>')] * BATCH_SIZE, 1)

        # Teacher forcing - feeding the target as the next input
        for t in range(1, targ.shape[1]):
            # passing enc_output to the decoder
            predictions, dec_hidden, _ = decoder(dec_input, dec_hidden, enc_output)

            loss += loss_function(targ[:, t], predictions)

            # using teacher forcing
            dec_input = tf.expand_dims(targ[:, t], 1)

    batch_loss = (loss / int(targ.shape[1]))

    variables = encoder.trainable_variables + decoder.trainable_variables

    gradients = tape.gradient(loss, variables)

    optimizer.apply_gradients(zip(gradients, variables))

    return batch_loss

def train(epochs = 10, verbose = 0 ):
    optimizer = tf.keras.optimizers.Adam()

    for epoch in range(epochs):
        start = time.time()

        enc_hidden = encoder.initialize_hidden_state()
        total_loss = 0

        for (batch, (inp, targ)) in enumerate(dataset.take(steps_per_epoch)):
            batch_loss = train_step(inp, targ, enc_hidden, encoder, decoder,␣
↪optimizer, en_tokenizer)
            total_loss += batch_loss

            if batch % 100 == 0 and verbose:
                print('Epoch {} Batch {} Loss {:.4f}'.format(epoch + 1,
                                                             batch,
                                                             batch_loss.numpy()))
```

(continues on next page)

(continued from previous page)

```
        if verbose:
            print('Epoch {} Loss {:.4f}'.format(epoch + 1,
                                                total_loss / steps_per_epoch))
            print('Time taken for 1 epoch {} sec\n'.format(time.time() - start))
```

Start training

[11]:
```
train(epochs = 10, verbose = 1)
```

```
Epoch 1 Batch 0 Loss 8.7736
Epoch 1 Batch 100 Loss 2.1184
Epoch 1 Batch 200 Loss 1.7768
Epoch 1 Batch 300 Loss 1.7248
Epoch 1 Batch 400 Loss 1.6401
Epoch 1 Loss 2.0055
Time taken for 1 epoch 1444.5116345882416 sec

Epoch 2 Batch 0 Loss 1.6100
Epoch 2 Batch 100 Loss 1.5598
Epoch 2 Batch 200 Loss 1.5922
Epoch 2 Batch 300 Loss 1.5228
Epoch 2 Batch 400 Loss 1.4033
Epoch 2 Loss 1.5530
Time taken for 1 epoch 1424.0314059257507 sec

Epoch 3 Batch 0 Loss 1.2111
Epoch 3 Batch 100 Loss 1.4820
Epoch 3 Batch 200 Loss 1.3912
Epoch 3 Batch 300 Loss 1.4882
Epoch 3 Batch 400 Loss 1.2942
Epoch 3 Loss 1.3888
Time taken for 1 epoch 1441.213187456131 sec

Epoch 4 Batch 0 Loss 1.2663
Epoch 4 Batch 100 Loss 1.3889
Epoch 4 Batch 200 Loss 1.1667
Epoch 4 Batch 300 Loss 1.2853
Epoch 4 Batch 400 Loss 1.2746
Epoch 4 Loss 1.2559
Time taken for 1 epoch 1422.2563009262085 sec

Epoch 5 Batch 0 Loss 1.1258
Epoch 5 Batch 100 Loss 1.1021
Epoch 5 Batch 200 Loss 1.1365
Epoch 5 Batch 300 Loss 1.1450
Epoch 5 Batch 400 Loss 1.3664
Epoch 5 Loss 1.1176
Time taken for 1 epoch 1378.149689912796 sec

Epoch 6 Batch 0 Loss 0.9396
Epoch 6 Batch 100 Loss 1.0216
Epoch 6 Batch 200 Loss 1.1066
Epoch 6 Batch 300 Loss 1.0084
Epoch 6 Batch 400 Loss 1.1767
Epoch 6 Loss 0.9732
Time taken for 1 epoch 1328.8411734104156 sec
```

(continues on next page)

(continued from previous page)

```
Epoch 7 Batch 0 Loss 0.9608
Epoch 7 Batch 100 Loss 0.8912
Epoch 7 Batch 200 Loss 0.8274
Epoch 7 Batch 300 Loss 0.8302
Epoch 7 Batch 400 Loss 0.7896
Epoch 7 Loss 0.8303
Time taken for 1 epoch 1294.177453994751 sec

Epoch 8 Batch 0 Loss 0.6882
Epoch 8 Batch 100 Loss 0.6465
Epoch 8 Batch 200 Loss 0.7108
Epoch 8 Batch 300 Loss 0.7176
Epoch 8 Batch 400 Loss 0.7323
Epoch 8 Loss 0.7000
Time taken for 1 epoch 1367.661788702011 sec

Epoch 9 Batch 0 Loss 0.5313
Epoch 9 Batch 100 Loss 0.4794
Epoch 9 Batch 200 Loss 0.6126
Epoch 9 Batch 300 Loss 0.6033
Epoch 9 Batch 400 Loss 0.5891
Epoch 9 Loss 0.5853
Time taken for 1 epoch 1372.0978388786316 sec

Epoch 10 Batch 0 Loss 0.5009
Epoch 10 Batch 100 Loss 0.5200
Epoch 10 Batch 200 Loss 0.4687
Epoch 10 Batch 300 Loss 0.4556
Epoch 10 Batch 400 Loss 0.4321
Epoch 10 Loss 0.4802
Time taken for 1 epoch 1334.807544708252 sec
```

6.6 Test

[12]:
```
def evaluate(sentence):
    attention_plot = np.zeros((max_length_tar, max_length_inp))

    inputs = ar_tokenizer.encode_sentences([sentence], boundries = ('<s>', '</s>'),
                                   out_length = max_length_inp)
    inputs = tf.convert_to_tensor(inputs)

    result = ''

    hidden = [tf.zeros((1, units))]
    enc_out, enc_hidden = encoder(inputs, hidden)

    dec_hidden = enc_hidden
    dec_input = tf.expand_dims([en_tokenizer.token_to_id('<s>')], 0)

    for t in range(max_length_tar):
        predictions, dec_hidden, attention_weights = decoder(dec_input,
                                                             dec_hidden,
                                                             enc_out)
```

(continues on next page)

(continued from previous page)

```
        # storing the attention weights to plot later on
        attention_weights = tf.reshape(attention_weights, (-1, ))
        attention_plot[t] = attention_weights.numpy()

        predicted_id = tf.argmax(predictions[0]).numpy()

        result += en_tokenizer.id_to_token(predicted_id) + ' '

        if en_tokenizer.id_to_token(predicted_id) == '</s>':
            return result, sentence

        # the predicted ID is fed back into the model
        dec_input = tf.expand_dims([predicted_id], 0)

    return result, sentence

def translate(sentences, translations, verbose = 1):
    inputs = sentences
    outputs = []

    for i, sentence in enumerate(sentences):
        result, sentence = evaluate(sentence)
        result = ar_tokenizer.detokenize(result)
        result = result.replace('<s>', '').replace('</s>', '')
        result = re.sub(' +', ' ', result)
        outputs.append(result)
        if verbose:
            print('inpt: %s' % (sentence))
            print('pred: {}'.format(result))
            print('true: {}'.format(translations[i]))
```

[13]:
```
translate(test_inp_text[:50], test_tar_text[:50], verbose = 1)
```

```
inpt:
pred:  Well there ' s the name for you
true: Well there ' s a bank for you
inpt:
pred:  What happened Dad
true: What happened Father
inpt:
pred:  Well I ' ll be years since
true: Well it ' s almost four years now
inpt:
pred:  That ' s right isn ' t it
true: That ' s right ain ' t it Ma
inpt:      5
pred:  Four months years of the floor
true: Four years Four years 5th June Pa
inpt:
pred:  I couldn ' t steal up for the prisoner ' s jewels
true: I couldn ' t keep up the payments
inpt:
pred:  You remember him
true: You remember him
inpt:
pred:  The Potem oin the toxic ms
```

(continues on next page)

(continued from previous page)

```
true: Randy Dunlap
inpt:
pred:  l journey less
true: Bark
inpt:
pred:  So he died and keep me to stop
true: So I dropped in and he asked me to sit down
inpt:
pred:  George do you know what was
true: George do you know what he was wearing
inpt:
pred:  Kim was
true: A kimono
inpt:
pred:  No
true: No
inpt:
pred:  Yeah
true: Yeah
inpt:
pred:  Oh now ' s silly
true: Oh now Bark
inpt:
pred:  The ve got a big room
true: It must have been a dressing gown
inpt:
pred:  I know the whole bedroom
true: I know a dressing gown when I see it
inpt:
pred:  He was amazing mom
true: It was a kimono George
inpt:
pred:  Are you a rooster with the prisoner glasses
true: Do dressing gowns have flowers on ' em
inpt:
pred:  Oh Merna
true: Oh Bark
inpt:
pred:  Don ' t mind who is my father
true: Never mind that Father
inpt:
pred:  What did he say
true: What did he say
inpt:
pred:  Oh he ' s my mom enough
true: Oh he was nice enough
inpt:
pred:  Oh now ' s silly
true: Oh now Bark
inpt:
pred:  Yes he ' s done
true: Yeah he did
inpt:
pred:  How long time is still Dad
true: How much time did he give you Father
inpt:
pred:  Nine months
```

(continues on next page)

(continued from previous page)

```
true: Six months
inpt:
pred:  Oh Well uh we ' s no coffin
true: Oh Oh well then there ' s no immediate rush
inpt:
pred:  When did you a pot s I inquire
true: When are the six months up
inpt:
pred:  Paper
true: Tuesday
inpt:
pred:  But but why didn ' t you stop them
true: But but why didn ' t you tell us sooner
inpt:
pred:  Paper
true: Tuesday
inpt:
pred:  Don ' t give them a lot of time is it
true: Doesn ' t give us much time does it
inpt:
pred:  Or
true: Or
inpt:
pred:  That ' s right
true: That ' s right
inpt:
pred:  Of course
true: Oh sure
inpt:
pred:  The d put this place is a few weeks
true: Who gave you that dress the Salvation Army
inpt:
pred:  And
true: And uh
inpt:
pred:  Yeah
true: Yeah
inpt:
pred:  Well I can ' t do it on your own
true: Well I can ' t do it alone
inpt:
pred:  No she ' s not your delicate ed and the Israelites
true: No she ' s never even sent us an orange
inpt:
pred:  Yes but what brings about
true: Yes but what about Harvey
inpt:
pred:  Oh we don ' t want to remember my mom
true: Oh we wouldn ' t want to ask Harvey
inpt:
pred:  Oh no I wouldn ' t die
true: Oh no we wouldn ' t ask Harvey
inpt:
pred:  Don ' t you caught my mom did you Rachel
true: No we asked Harvey to marry Nellie
inpt:
pred:  We can ' t we ' ve got a man can do that
```

(continues on next page)

(continued from previous page)

```
true: We can ' t expect the guy to do more than that
inpt:
pred:  Elizabeth stop talking to the way
true: Robert stop talking that way
inpt:
pred:  A spear it Robert
true: Cut it out Robert
inpt:
pred:  I haven ' t the whole world I ' re in
true: I haven ' t room for both of you
inpt:
pred:  There ' s nothing for a big tree in the street
true: There ' s only a small couch in the living room
```

PYTHON MODULE INDEX

t

INDEX

T

W