Double-click (or enter) to edit

```
import os
import re
import pickle
import string
import unicodedata
from random import randint

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from nltk.corpus import stopwords
from wordcloud import STOPWORDS, WordCloud

from sklearn.model_selection import train_test_split

import tensorflow as tf
from tensorflow.keras import Input, Model
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.layers import LSTM, Bidirectional, Dense, Embedding, TimeDistributed
```

```
!pip install -q contractions==0.0.48
```

```
from contractions import contractions_dict

for key, value in list(contractions_dict.items())[:10]:
    print(f'{key} == {value}')
```

```
I'm == I am
I'm'a == I am about to
I'm'o == I am going to
I've == I have
I'll == I will
I'll've == I will have
I'd == I would
I'd've == I would have
Whatcha == What are you
amn't == am not
```

```
# Using TPU

# detect and init the TPU
tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
tf.config.experimental_connect_to_cluster(tpu)
tf.tpu.experimental.initialize_tpu_system(tpu)

# instantiate a distribution strategy
tpu_strategy = tf.distribute.experimental.TPUStrategy(tpu)
```

## Getting the data

```
filename1 = '../input/news-summary/news_summary.csv'
filename2 = '../input/news-summary/news_summary_more.csv'

df1 = pd.read_csv(filename1, encoding='iso-8859-1').reset_index(drop=True)
df2 = pd.read_csv(filename2, encoding='iso-8859-1').reset_index(drop=True)
```

```
df1.sample(5)
```

| | author | date | headlines | read_more | text | |
|---|---|---|---|---|---|---|
| **1722** | Chhavi Tyagi | 06 Jan 2017,Friday | Modi must go, let Advani, Jaitley or Rajnath l... | http://indiatoday.intoday.in/story/modi-advani... | West Bengal Chief Minister Mamata Banerjee on ... | c |
| | | | Former Minister E | | E Ahamed, the | IU |

```
df2.sample(5)
```

| | headlines | text |
|---|---|---|
| **53041** | Selling sex in Bollywood is dead: Filmmaker Vi... | Filmmaker Vikram Bhatt, who is known for his b... |
| **61744** | Telangana govt declares Urdu as second officia... | Telangana Chief Minister K Chandrashekar Rao o... |
| **4742** | Vanakam Puducherry: Rahul says is PM's answer ... | Taking a dig at PM Narendra Modi for allegedly... |
| **34195** | Let's not explore political mileage in Ram's n... | Talking about the Ayodhya dispute, Vishva Hind... |
| **26730** | Disappointed, not surprised: Cyrus Mistry on N... | Cyrus Mistry has said he was disappointed but ... |

```
df1_columns = df1.columns.tolist()
df1_columns.remove('headlines')
df1_columns.remove('text')
df1.drop(df1_columns, axis='columns', inplace=True)

df = pd.concat([df1, df2], axis='rows')
del df1, df2

# Shuffling the df
df = df.sample(frac=1).reset_index(drop=True)

print(f'Dataset size: {len(df)}')
df.sample(5)
```

```
Dataset size: 102915
```

| | headlines | text |
|---|---|---|
| **45406** | Former Huawei engineer 'marries' robot he buil... | A former Huawei engineer from China has "marri... |
| **101543** | Aus brand creates Harry Potter-inspired clothi... | Australian fashion label Black Milk has create... |
| **24029** | Australia to sell Kesar variety of Indian mang... | Australia will be selling Kesar variety of Ind... |
| **15782** | White House hires outside counsel to deal with... | US Vice President Mike Pence has reportedly hi... |
| **44624** | AT&T updates 4G logo with '5G E', criticised f... | US-based telco AT&T received criticism from it... |

The `headlines` column will be treated as `summary` for the text.

## 🏄 Data preparation

```
def expand_contractions(text, contraction_map=contractions_dict):
    # Using regex for getting all contracted words
    contractions_keys = '|'.join(contraction_map.keys())
    contractions_pattern = re.compile(f'({contractions_keys})', flags=re.DOTALL)

    def expand_match(contraction):
        # Getting entire matched sub-string
        match = contraction.group(0)
        expanded_contraction = contraction_map.get(match)
        if not expand_contractions:
            print(match)
            return match
        return expanded_contraction

    expanded_text = contractions_pattern.sub(expand_match, text)
    expanded_text = re.sub("'", "", expanded_text)
    return expanded_text


expand_contractions("y'all can't expand contractions i'd think")
```

```
'you all can not expand contractions id think'
```

```
# Converting to lowercase
df.text = df.text.apply(str.lower)
df.headlines = df.headlines.apply(str.lower)

df.sample(5)
```

| | headlines | text |
|---|---|---|
| **22697** | 92 witnesses, 100 evidences in mandsaur gangra... | mandsaur police on tuesday filed a 350-page ch... |
| **79342** | pm modi meets fishermen affected by cyclone ockhi | prime minister narendra modi on tuesday met fi... |
| **51795** | can't eat dal-chawal all day: shreyas on doing... | shreyas talpade has said, "one can't eat 'dal-... |
| **33033** | govt restricts operations of 2.09 lakh firms' ... | finance ministry on tuesday said operations of... |
| **51218** | suffering from facial paralysis: simran writer... | apurva asrani, co-writer of kangana ranaut's '... |

```
df.headlines = df.headlines.apply(expand_contractions)
df.text = df.text.apply(expand_contractions)
df.sample(5)
```

| | headlines | text |
|---|---|---|
| **69141** | who are the sentinelese, tribe that killed an ... | the sentinelese are an indigenous tribe who ha... |
| **8469** | internet should be utility available to whole ... | chinas e-commerce giant alibabas founder, jack... |
| **41465** | maoists kill watchman, set ablaze 5 vehicles i... | maoists shot dead a watchman, suspecting him t... |
| **99162** | salmans nephew ahils 1st birthday celebrated i... | salman khans sister arpita khan sharma shared ... |
| **15109** | perhaps rahul plans to win polls in russia, in... | mocking congress vice president rahul gandhi, ... |

```
# Remove puncuation from word
def rm_punc_from_word(word):
    clean_alphabet_list = [
        alphabet for alphabet in word if alphabet not in string.punctuation
    ]
    return ''.join(clean_alphabet_list)

print(rm_punc_from_word('#cool!'))


# Remove puncuation from text
def rm_punc_from_text(text):
    clean_word_list = [rm_punc_from_word(word) for word in text]
    return ''.join(clean_word_list)

print(rm_punc_from_text("Frankly, my dear, I don't give a damn"))
```

```
 cool
 Frankly my dear I dont give a damn
```

```
# Remove numbers from text
def rm_number_from_text(text):
    text = re.sub('[0-9]+', '', text)
    return ' '.join(text.split())  # to rm `extra` white space

print(rm_number_from_text('You are 100times more sexier than me'))
print(rm_number_from_text('If you taught yes then you are 10 times more delusional than me'))
```

```
 You are times more sexier than me
 If you taught yes then you are times more delusional than me
```

```
# Remove stopwords from text
def rm_stopwords_from_text(text):
    _stopwords = stopwords.words('english')
    text = text.split()
    word_list = [word for word in text if word not in _stopwords]
    return ' '.join(word_list)

rm_stopwords_from_text("Love means never having to say you're sorry")
```

```
 'Love means never say sorry'
```

```
# Cleaning text
def clean_text(text):
    text = text.lower()
    text = rm_punc_from_text(text)
    text = rm_number_from_text(text)
    text = rm_stopwords_from_text(text)

    # there are hyphen(–) in many titles, so replacing it with empty str
    # this hyphen(–) is different from normal hyphen(-)
    text = re.sub('–', '', text)
    text = ' '.join(text.split())  # removing `extra` white spaces

    # Removing unnecessary characters from text
    text = re.sub("(\\t)", ' ', str(text)).lower()
    text = re.sub("(\\r)", ' ', str(text)).lower()
    text = re.sub("(\\n)", ' ', str(text)).lower()

    # remove accented chars ('Sómě Áccěntěd těxt' => 'Some Accented text')
    text = unicodedata.normalize('NFKD', text).encode('ascii', 'ignore').decode(
        'utf-8', 'ignore'
    )

    text = re.sub("(__+)", ' ', str(text)).lower()
    text = re.sub("(--+)", ' ', str(text)).lower()
    text = re.sub("(~~+)", ' ', str(text)).lower()
```

```
    text = re.sub("(\+\++)", ' ', str(text)).lower()
    text = re.sub("(\.\.+)", ' ', str(text)).lower()

    text = re.sub(r"[<>()|&©ø\[\]\'\",;?~*!]", ' ', str(text)).lower()

    text = re.sub("(mailto:)", ' ', str(text)).lower()
    text = re.sub(r"(\\x9\d)", ' ', str(text)).lower()
    text = re.sub("([iI][nN][cC]\d+)", 'INC_NUM', str(text)).lower()
    text = re.sub("([cC][mM]\d+)|([cC][hH][gG]\d+)", 'CM_NUM',
                  str(text)).lower()

    text = re.sub("(\.\s+)", ' ', str(text)).lower()
    text = re.sub("(\-\s+)", ' ', str(text)).lower()
    text = re.sub("(\:\s+)", ' ', str(text)).lower()
    text = re.sub("(\s+.\s+)", ' ', str(text)).lower()

    try:
        url = re.search(r'((https*:\/*)([^\/\s]+))(.[^\s]+)', str(text))
        repl_url = url.group(3)
        text = re.sub(r'((https*:\/*)([^\/\s]+))(.[^\s]+)', repl_url, str(text))
    except Exception as e:
        pass

    text = re.sub("(\s+)", ' ', str(text)).lower()
    text = re.sub("(\s+.\s+)", ' ', str(text)).lower()

    return text

clean_text("Mrs. Robinson, you're trying to seduce me, aren't you?")
```

```
'mrs robinson youre trying seduce arent'
```

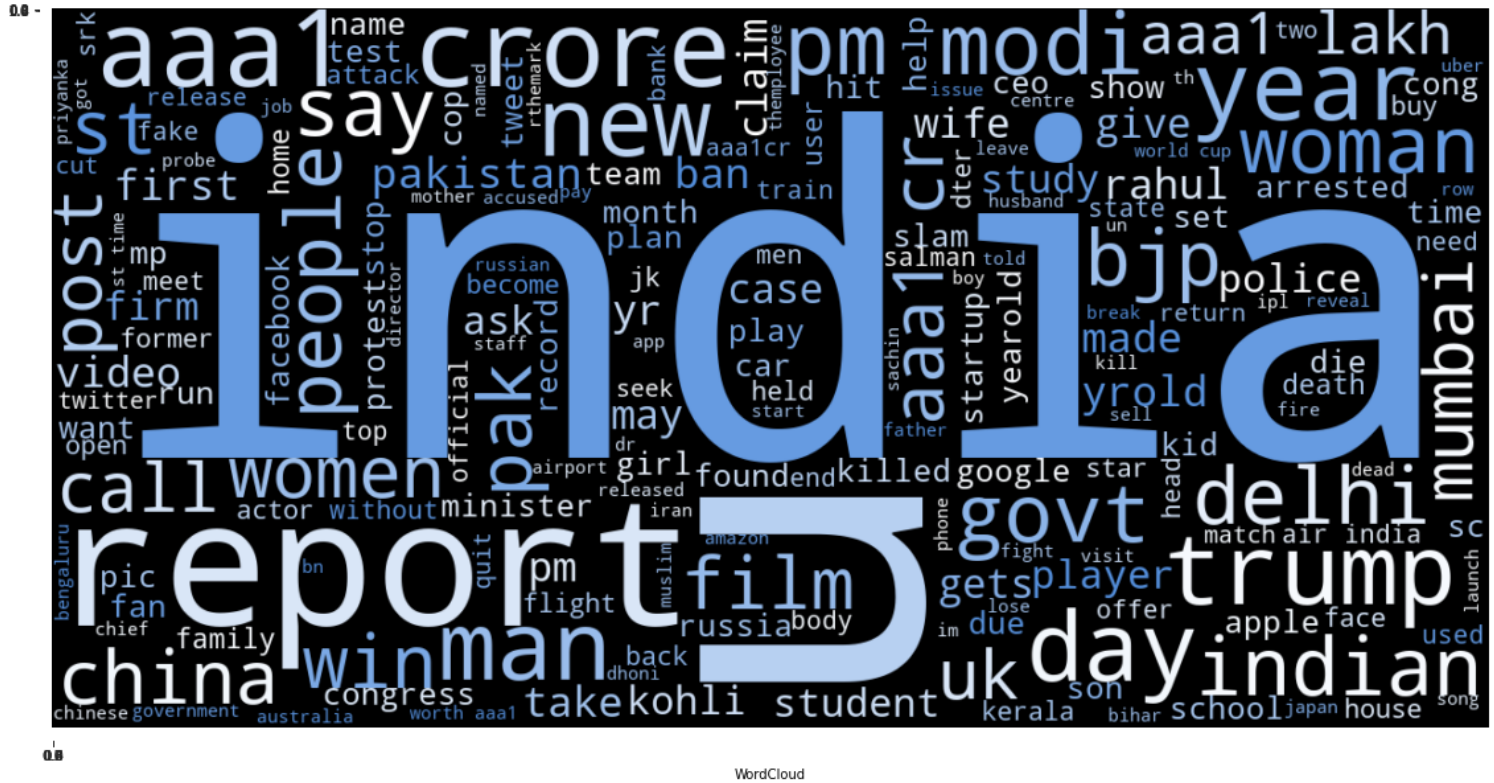
```
df.text = df.text.apply(clean_text)
df.headlines = df.headlines.apply(clean_text)
df.sample(5)
```

| | headlines | text |
|---|---|---|
| **54141** | jpmorgan made bn months record us bank | jpmorgan chase biggest us bank made billion pa... |
| **10135** | google like cult says engineer fired mthemo | former google engineer james damore fired page... |
| **12851** | ban plying atm cash vans pm mha suggests centre | ministry home affairs mha proposed atms replen... |
| **39281** | ongc gives inprinciple nod acquire govts stake... | board staterun oil natural gas corporation ong... |
| **75117** | pakistan allocates aaa1 crore defence | pakistan finance minister miftah ismail friday... |

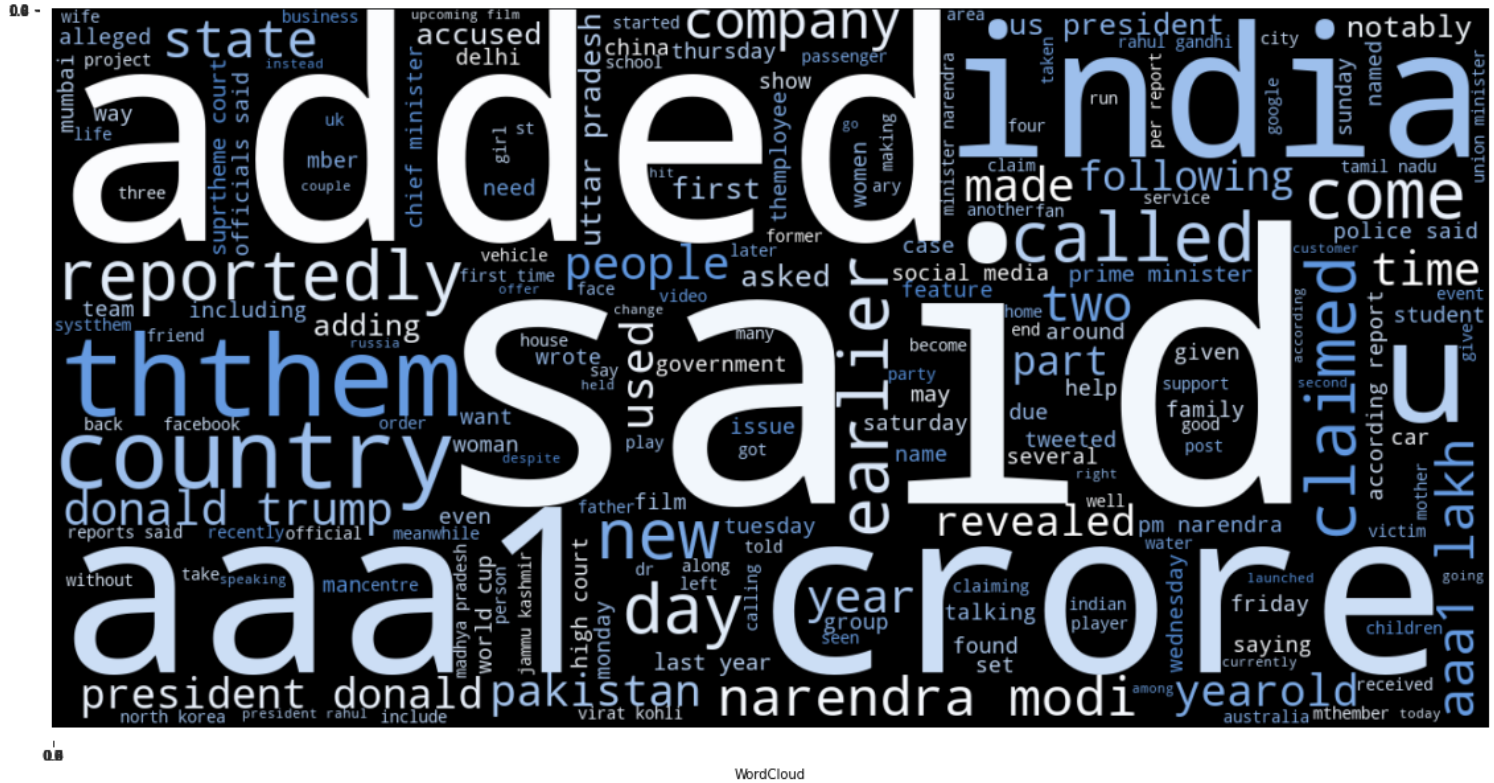
```
# saving the cleaned data
df.to_csv('cleaned_data.csv')
```

```
# To customize colours of wordcloud texts
def wc_blue_color_func(word, font_size, position, orientation, random_state=None, **kwargs):
    return "hsl(214, 67%%, %d%%)" % randint(60, 100)


# stopwords for wordcloud
def get_wc_stopwords():
    wc_stopwords = set(STOPWORDS)

    # Adding words to stopwords
    # these words showed up while plotting wordcloud for text
    wc_stopwords.add('s')
    wc_stopwords.add('one')
    wc_stopwords.add('using')
    wc_stopwords.add('example')
    wc_stopwords.add('work')
    wc_stopwords.add('use')
    wc_stopwords.add('make')

    return wc_stopwords


# plot wordcloud
def plot_wordcloud(text, color_func):
    wc_stopwords = get_wc_stopwords()
    wc = WordCloud(stopwords=wc_stopwords, width=1200, height=600, random_state=0).generate(text)

    f, axs = plt.subplots(figsize=(20, 10))
    with sns.axes_style("ticks"):
        sns.despine(offset=10, trim=True)
        plt.imshow(wc.recolor(color_func=color_func, random_state=0), interpolation="bilinear")
        plt.xlabel('WordCloud')
```

```
plot_wordcloud(' '.join(df.headlines.values.tolist()), wc_blue_color_func)
```




```
plot_wordcloud(' '.join(df.text.values.tolist()), wc_blue_color_func)
```




Using a `start` and `end` tokens in `headlines(summary)` to let the learning algorithm know from where the headlines start's and end's.

```
df.headlines = df.headlines.apply(lambda x: f'_START_ {x} _END_')
```

Again adding `tokens` ... but different ones.

```
start_token = 'sostok'
end_token = 'eostok'
df.headlines = df.headlines.apply(lambda x: f'{start_token} {x} {end_token}')
```

It's important to use `sostok` and `eostok` as start and end tokens respectively as later while using `tensorflow's Tokenizer` will filter the tokens and covert them to lowercase.

**sostok** & **eostok** tokens are for us to know where to start & stop the summary because using `_START_` & `_END_`, tf's tokenizer with convert them to **start** & **end** respectively.

So while decoding the summary sequences of sentences like **'everything is going to end in 2012'** if use `_START_` & `_END_` tokens (which will make the sentence like **'start everything is going to end in 2012 end'** this) whome tf's tokenizer will convert to start and end then we will stop decoding as we hit first **end**, so this is bad and therefore **sostok** & **eostok** these tokens are used.

So we can just use these **sostok** & **eostok** instead of `_START_` & `_END_`, well you can but I tried both ways and while not using these `_START_` & `_END_` I was getting `undesired results` 🤯 😅 i.e. model's `results weren't good`.

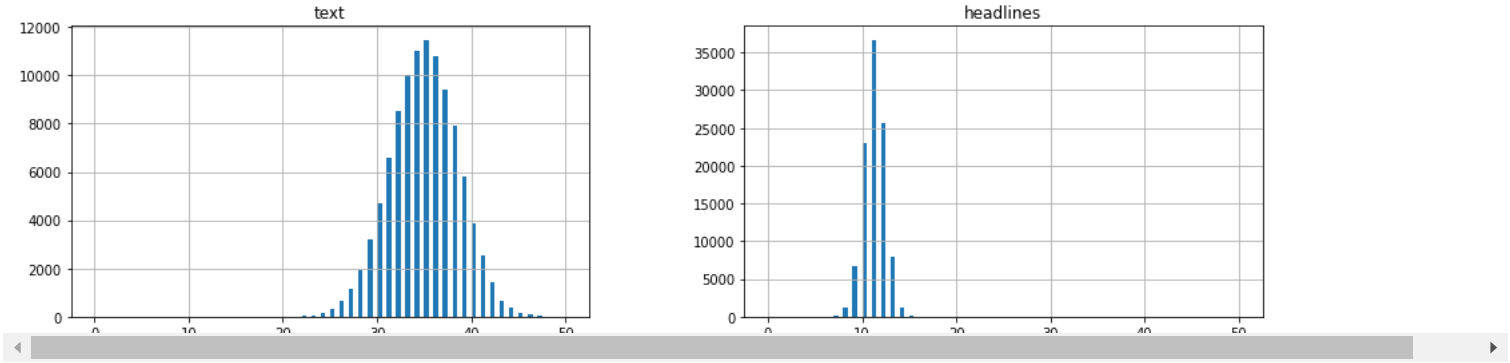
```
df.sample(5)
```

| | headlines | text |
|---|---|---|
| **88989** | sostok _START_ punjab police make second arres... | punjab police saturday arrested amritsar blast... |
| **32928** | sostok _START_ wanted publicity would posed pl... | gilu jo slutshamed breastfeeding baby cover ma... |
| **55234** | sostok _START_ feel small front soha ali khan ... | kareena kapoor speaking occasion sisterinlaw s... |
| **60176** | sostok _START_ want watch baahubali seeing tra... | actress sayani gupta said feel like watching f... |
| **12371** | sostok _START_ bharat special bagged role oppo... | actress disha patani seen salman khans bharat ... |

Finding what should be the `maximum length` of text and headlines that will be feed or accepted by the learning algorithm.

```
text_count = [len(sentence.split()) for sentence in df.text]
headlines_count = [len(sentence.split()) for sentence in df.headlines]

pd.DataFrame({'text': text_count, 'headlines': headlines_count}).hist(bins=100, figsize=(16, 4), range=[0, 50])
plt.show()
```

```
/opt/conda/lib/python3.7/site-packages/pandas/plotting/_matplotlib/tools.py:400: MatplotlibDeprecationWarning:
The is_first_col function was deprecated in Matplotlib 3.4 and will be removed two minor releases later. Use ax.get_subplotspec().is_first_col() i
  if ax.is_first_col():
```




```
# To check how many rows in a column has length (of the text) <= limit
def get_word_percent(column, limit):
    count = 0
    for sentence in column:
        if len(sentence.split()) <= limit:
            count += 1

    return round(count / len(column), 2)


# Check how many % of headlines have 0-13 words
print(get_word_percent(df.headlines, 13))

# Check how many % of summary have 0-42 words
print(get_word_percent(df.text, 42))
```

```
0.99
0.99
```

If the length of headlines or the text is kept large the deep learning model will face issues with performance and also training will slower.

One solution for creating summary for long sentences can be break a paragraph into sentences and then create a summary for them, this way the summary will make sence instead of giving random piece of text and creating summary for it.

```
max_text_len = 42
max_summary_len = 13
```

```
# select the summary and text between their defined max lens respectively
def trim_text_and_summary(df, max_text_len, max_summary_len):
    cleaned_text = np.array(df['text'])
    cleaned_summary = np.array(df['headlines'])

    short_text = []
    short_summary = []

    for i in range(len(cleaned_text)):
        if len(cleaned_text[i].split()) <= max_text_len and len(
            cleaned_summary[i].split()
        ) <= max_summary_len:
            short_text.append(cleaned_text[i])
            short_summary.append(cleaned_summary[i])

    df = pd.DataFrame({'text': short_text, 'summary': short_summary})
    return df


df = trim_text_and_summary(df, max_text_len, max_summary_len)
print(f'Dataset size: {len(df)}')
df.sample(5)
```

```
Dataset size: 100258
```

| | text | summary |
|---|---|---|
| **87248** | france imposed law requiring citizens display ... | sostok _START_ france require clean stickers c... |
| **79580** | indian fast bowler sreesanth revealed consider... | sostok _START_ play country sreesanth ban rest... |
| **92636** | ngo filed written complaint mumbai cyber polic... | sostok _START_ ngo files complaint rishi posti... |
| **4937** | late pm atal bihari vajpayees niece karuna shu... | sostok _START_ chhattisagrah cm nothing rajnan... |
| **67151** | union minister mukhtar abbas naqvi thursday sa... | sostok _START_ opposition stop banging head ev... |

```
# rare word analysis
def get_rare_word_percent(tokenizer, threshold):
    # threshold: if the word's occurrence is less than this then it's rare word

    count = 0
    total_count = 0
    frequency = 0
    total_frequency = 0

    for key, value in tokenizer.word_counts.items():
        total_count += 1
        total_frequency += value
        if value < threshold:
            count += 1
            frequency += value

    return {
        'percent': round((count / total_count) * 100, 2),
        'total_coverage': round(frequency / total_frequency * 100, 2),
        'count': count,
        'total_count': total_count
    }
```

```
# Splitting the training and validation sets
x_train, x_val, y_train, y_val = train_test_split(
    np.array(df['text']),
    np.array(df['summary']),
    test_size=0.1,
    random_state=1,
    shuffle=True
)
```

**Tokenizing text 👉 x**

```
x_tokenizer = Tokenizer()
x_tokenizer.fit_on_texts(list(x_train))

x_tokens_data = get_rare_word_percent(x_tokenizer, 4)
print(x_tokens_data)
```

```
{'percent': 65.36, 'total_coverage': 2.95, 'count': 65213, 'total_count': 99775}
```

🔥 to `increase computation speed` use this

```
x_tokenizer = Tokenizer(num_words=x_tokens_data['total_count'] - x_tokens_data['count'])
```

```
# else use this
x_tokenizer = Tokenizer()
x_tokenizer.fit_on_texts(list(x_train))
```

```
# save tokenizer
with open('x_tokenizer', 'wb') as f:
    pickle.dump(x_tokenizer, f, protocol=pickle.HIGHEST_PROTOCOL)
```

```
# one-hot-encoding
x_train_sequence = x_tokenizer.texts_to_sequences(x_train)
x_val_sequence = x_tokenizer.texts_to_sequences(x_val)

# padding upto max_text_len
x_train_padded = pad_sequences(x_train_sequence, maxlen=max_text_len, padding='post')
x_val_padded = pad_sequences(x_val_sequence, maxlen=max_text_len, padding='post')

# if you're not using num_words parameter in Tokenizer then use this
x_vocab_size = len(x_tokenizer.word_index) + 1

# else use this
# x_vocab_size = x_tokenizer.num_words + 1
```

```
print(x_vocab_size)
```

```
99776
```

**Tokenizing headlines(summary) 👉 y**

```
y_tokenizer = Tokenizer()
y_tokenizer.fit_on_texts(list(y_train))

y_tokens_data = get_rare_word_percent(y_tokenizer, 6)
print(y_tokens_data)
```

```
{'percent': 69.27, 'total_coverage': 4.81, 'count': 25994, 'total_count': 37523}
```

🔥 to `increase computation speed` use this

```
y_tokenizer = Tokenizer(num_words=y_tokens_data['total_count'] - y_tokens_data['count'])
```

```
# else use this
y_tokenizer = Tokenizer()
y_tokenizer.fit_on_texts(list(y_train))
```

```
# save tokenizer
with open('y_tokenizer', 'wb') as f:
    pickle.dump(y_tokenizer, f, protocol=pickle.HIGHEST_PROTOCOL)
```

```
# one-hot-encoding
y_train_sequence = y_tokenizer.texts_to_sequences(y_train)
y_val_sequence = y_tokenizer.texts_to_sequences(y_val)

# padding upto max_summary_len
y_train_padded = pad_sequences(y_train_sequence, maxlen=max_summary_len, padding='post')
y_val_padded = pad_sequences(y_val_sequence, maxlen=max_summary_len, padding='post')

# if you're not using num_words parameter in Tokenizer then use this
y_vocab_size = len(y_tokenizer.word_index) + 1

# else use this
# y_vocab_size = y_tokenizer.num_words + 1

print(y_vocab_size)
```

```
37524
```

```
# removing summary which only has sostok & eostok
def remove_indexes(summary_array):
    remove_indexes = []
    for i in range(len(summary_array)):
        count = 0
        for j in summary_array[i]:
            if j != 0:
                count += 1
        if count == 2:
            remove_indexes.append(i)
    return remove_indexes


remove_train_indexes = remove_indexes(y_train_padded)
remove_val_indexes = remove_indexes(y_val_padded)

y_train_padded = np.delete(y_train_padded, remove_train_indexes, axis=0)
x_train_padded = np.delete(x_train_padded, remove_train_indexes, axis=0)

y_val_padded = np.delete(y_val_padded, remove_val_indexes, axis=0)
x_val_padded = np.delete(x_val_padded, remove_val_indexes, axis=0)
```

## 🤼 Modelling

```
latent_dim = 240
embedding_dim = 300
num_epochs = 50
```

```
def get_embedding_matrix(tokenizer, embedding_dim, vocab_size=None):
    word_index = tokenizer.word_index
    voc = list(word_index.keys())

    path_to_glove_file = '../input/glove6b/glove.6B.300d.txt'

    embeddings_index = {}
    with open(path_to_glove_file) as f:
        for line in f:
            word, coefs = line.split(maxsplit=1)
            coefs = np.fromstring(coefs, "f", sep=" ")
            embeddings_index[word] = coefs

    print("Found %s word vectors." % len(embeddings_index))

    num_tokens = len(voc) + 2 if not vocab_size else vocab_size
    hits = 0
    misses = 0

    # Prepare embedding matrix
    embedding_matrix = np.zeros((num_tokens, embedding_dim))
    for word, i in word_index.items():
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            # Words not found in embedding index will be all-zeros.
            # This includes the representation for "padding" and "OOV"
            embedding_matrix[i] = embedding_vector
            hits += 1
        else:
            misses += 1
    print("Converted %d words (%d misses)" % (hits, misses))

    return embedding_matrix


x_embedding_matrix = get_embedding_matrix(x_tokenizer, embedding_dim, x_vocab_size)
y_embedding_matrix = get_embedding_matrix(y_tokenizer, embedding_dim, y_vocab_size)
```

```
Found 400000 word vectors.
Converted 55319 words (44456 misses)
Found 400000 word vectors.
Converted 27095 words (10428 misses)
```

```
print(x_embedding_matrix.shape)
print(y_embedding_matrix.shape)
```

```
(99776, 300)
(37524, 300)
```

Using `pre-trained` embeddings and keeping the `Embedding` layer `non-trainable` we get increase in computation speed as don't need to compute the embedding matrix.

**Here there 3 different training models**

- `build_seq2seq_model_with_just_lstm` - **Seq2Seq model with just LSTMs**. Both `encoder` and `decoder` have just `LSTM`s.

- `build_seq2seq_model_with_bidirectional_lstm` - **Seq2Seq model with Bidirectional LSTMs**. Both `encoder` and `decoder` have `Bidirectional LSTM`s.
- `build_hybrid_seq2seq_model` - **Seq2Seq model with hybrid architecture**. Here `encoder` has `Bidirectional LSTM`s while `decoder` has just `LSTM`s.

**Inference methods for the 3 different learning models - just add `_inference` as `prefix`**

- `build_seq2seq_model_with_just_lstm_inference`
- `build_seq2seq_model_with_bidirectional_lstm_inference`
- `build_hybrid_seq2seq_model_inference`

**Decoding sequence for the 3 different learning models - just add `decode_sequence_` as `suffix`**

- `decode_sequence_build_seq2seq_model_with_just_lstm`
- `decode_sequence_build_seq2seq_model_with_bidirectional_lstm`
- `decode_sequence_build_hybrid_seq2seq_model`

**Seq2Seq model with just LSTMs**. Both `encoder` and `decoder` have just `LSTM`s.

```
def build_seq2seq_model_with_just_lstm(
    embedding_dim, latent_dim, max_text_len,
    x_vocab_size, y_vocab_size,
    x_embedding_matrix, y_embedding_matrix
):
    # instantiating the model in the strategy scope creates the model on the TPU
    with tpu_strategy.scope():

        # =====================
        # 🔥 Encoder
        # =====================
        encoder_input = Input(shape=(max_text_len, ))

        # encoder embedding layer
        encoder_embedding = Embedding(
            x_vocab_size,
            embedding_dim,
            embeddings_initializer=tf.keras.initializers.Constant(x_embedding_matrix),
            trainable=False
        )(encoder_input)

        # encoder lstm 1
        encoder_lstm1 = LSTM(
            latent_dim,
            return_sequences=True,
            return_state=True,
            dropout=0.4,
            recurrent_dropout=0.4
        )
        encoder_output1, state_h1, state_c1 = encoder_lstm1(encoder_embedding)

        # encoder lstm 2
        encoder_lstm2 = LSTM(
            latent_dim,
            return_sequences=True,
            return_state=True,
            dropout=0.4,
            recurrent_dropout=0.4
        )
        encoder_output, *encoder_final_states = encoder_lstm2(encoder_output1)

        # =====================
        # 🌈 Decoder
        # =====================

        # Set up the decoder, using `encoder_states` as initial state.

        decoder_input = Input(shape=(None, ))

        # decoder embedding layer
        decoder_embedding_layer = Embedding(
            y_vocab_size,
            embedding_dim,
            embeddings_initializer=tf.keras.initializers.Constant(y_embedding_matrix),
            trainable=True
        )
        decoder_embedding = decoder_embedding_layer(decoder_input)

        # decoder lstm 1
        decoder_lstm = LSTM(
            latent_dim,
            return_sequences=True,
            return_state=True,
```

```
            dropout=0.4,
            recurrent_dropout=0.4
        )
        decoder_output, *decoder_final_states = decoder_lstm(
            decoder_embedding, initial_state=encoder_final_states
        )

        # dense layer
        decoder_dense = TimeDistributed(
            Dense(y_vocab_size, activation='softmax')
        )
        decoder_output = decoder_dense(decoder_output)

        # ====================
        # ⚡ Model
        # ====================
        model = Model([encoder_input, decoder_input], decoder_output)
        model.summary()

        optimizer = tf.keras.optimizers.RMSprop(learning_rate=0.001)
        model.compile(
            optimizer=optimizer,
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy']
        )

        return {
            'model': model,
            'inputs': {
                'encoder': encoder_input,
                'decoder': decoder_input
            },
            'outputs': {
                'encoder': encoder_output,
                'decoder': decoder_output
            },
            'states': {
                'encoder': encoder_final_states,
                'decoder': decoder_final_states
            },
            'layers': {
                'decoder': {
                    'embedding': decoder_embedding_layer,
                    'last_decoder_lstm': decoder_lstm,
                    'dense': decoder_dense
                }
            }
        }
```

**Seq2Seq model with Bidirectional LSTMs**. Both `encoder` and `decoder` have `Bidirectional LSTM`s.

```
def build_seq2seq_model_with_bidirectional_lstm(
    embedding_dim, latent_dim, max_text_len,
    x_vocab_size, y_vocab_size,
    x_embedding_matrix, y_embedding_matrix
):
    # instantiating the model in the strategy scope creates the model on the TPU
    with tpu_strategy.scope():

        # ====================
        # 🔥 Encoder
        # ====================
        encoder_input = Input(shape=(max_text_len, ))

        # encoder embedding layer
        encoder_embedding = Embedding(
            x_vocab_size,
            embedding_dim,
            embeddings_initializer=tf.keras.initializers.Constant(x_embedding_matrix),
            trainable=False,
            name='encoder_embedding'
        )(encoder_input)

        # encoder lstm1
        encoder_bi_lstm1 = Bidirectional(
            LSTM(
                latent_dim,
                return_sequences=True,
                return_state=True,
                dropout=0.4,
                recurrent_dropout=0.4,
                name='encoder_lstm_1'
            ),
            name='encoder_bidirectional_lstm_1'
```

```
)
encoder_output1, forward_h1, forward_c1, backward_h1, backward_c1 = encoder_bi_lstm1(
    encoder_embedding
)
encoder_bi_lstm1_output = [
    encoder_output1, forward_h1, forward_c1, backward_h1, backward_c1
]

# encoder lstm 2
encoder_bi_lstm2 = Bidirectional(
    LSTM(
        latent_dim,
        return_sequences=True,
        return_state=True,
        dropout=0.4,
        recurrent_dropout=0.4,
        name='encoder_lstm_2'
    ),
    name='encoder_bidirectional_lstm_2'
)
encoder_output2, forward_h2, forward_c2, backward_h2, backward_c2 = encoder_bi_lstm2(
    encoder_output1
)
encoder_bi_lstm2_output = [
    encoder_output2, forward_h2, forward_c2, backward_h2, backward_c2
]

# encoder lstm 3
encoder_bi_lstm = Bidirectional(
    LSTM(
        latent_dim,
        return_sequences=True,
        return_state=True,
        dropout=0.4,
        recurrent_dropout=0.4,
        name='encoder_lstm_3'
    ),
    name='encoder_bidirectional_lstm_3'
)
encoder_output, *encoder_final_states = encoder_bi_lstm(encoder_output2)

# =====================
# 🌈 Decoder
# =====================

# Set up the decoder, using `encoder_states` as initial state.

decoder_input = Input(shape=(None, ))

# decoder embedding layer
decoder_embedding_layer = Embedding(
    y_vocab_size,
    embedding_dim,
    embeddings_initializer=tf.keras.initializers.Constant(y_embedding_matrix),
    trainable=False,
    name='decoder_embedding'
)
decoder_embedding = decoder_embedding_layer(decoder_input)

decoder_bi_lstm = Bidirectional(
    LSTM(
        latent_dim,
        return_sequences=True,
        return_state=True,
        dropout=0.4,
        recurrent_dropout=0.2,
        name='decoder_lstm_1'
    ),
    name='decoder_bidirectional_lstm_1'
)
decoder_output, *decoder_final_states = decoder_bi_lstm(
    decoder_embedding, initial_state=encoder_final_states
    # decoder_embedding, initial_state=encoder_final_states[:2]
)  # taking only the forward states

# dense layer
decoder_dense = TimeDistributed(
    Dense(y_vocab_size, activation='softmax')
)
decoder_output = decoder_dense(decoder_output)

# =====================
# ⚡ Model
# =====================
model = Model([encoder_input, decoder_input], decoder_output, name='seq2seq_model_with_bidirectional_lstm')
```

```
        model.summary()

        optimizer = tf.keras.optimizers.RMSprop(learning_rate=0.001)
        model.compile(
            optimizer=optimizer,
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy']
        )

        return {
            'model': model,
            'inputs': {
                'encoder': encoder_input,
                'decoder': decoder_input
            },
            'outputs': {
                'encoder': encoder_output,
                'decoder': decoder_output
            },
            'states': {
                'encoder': encoder_final_states,
                'decoder': decoder_final_states
            },
            'layers': {
                'decoder': {
                    'embedding': decoder_embedding_layer,
                    'last_decoder_lstm': decoder_bi_lstm,
                    'dense': decoder_dense
                }
            }
        }
```

**Seq2Seq model with hybrid architecture**. Here `encoder` has `Bidirectional LSTM`s while `decoder` has just `LSTM`s.

```
def build_hybrid_seq2seq_model(
    embedding_dim, latent_dim, max_text_len,
    x_vocab_size, y_vocab_size,
    x_embedding_matrix, y_embedding_matrix
):
    # instantiating the model in the strategy scope creates the model on the TPU
    with tpu_strategy.scope():

        # =====================
        # 🔥 Encoder
        # =====================
        encoder_input = Input(shape=(max_text_len, ))

        # encoder embedding layer
        encoder_embedding = Embedding(
            x_vocab_size,
            embedding_dim,
            embeddings_initializer=tf.keras.initializers.Constant(x_embedding_matrix),
            trainable=False,
            name='encoder_embedding'
        )(encoder_input)

        # encoder lstm1
        encoder_bi_lstm1 = Bidirectional(
            LSTM(
                latent_dim,
                return_sequences=True,
                return_state=True,
                dropout=0.4,
                recurrent_dropout=0.4,
                name='encoder_lstm_1'
            ),
            name='encoder_bidirectional_lstm_1'
        )
        encoder_output1, forward_h1, forward_c1, backward_h1, backward_c1 = encoder_bi_lstm1(
            encoder_embedding
        )
        encoder_bi_lstm1_output = [
            encoder_output1, forward_h1, forward_c1, backward_h1, backward_c1
        ]

        # encoder lstm 2
        encoder_bi_lstm2 = Bidirectional(
            LSTM(
                latent_dim,
                return_sequences=True,
                return_state=True,
                dropout=0.4,
                recurrent_dropout=0.4,
                name='encoder_lstm_2'
```

```
    ),
    name='encoder_bidirectional_lstm_2'
)
encoder_output2, forward_h2, forward_c2, backward_h2, backward_c2 = encoder_bi_lstm2(
    encoder_output1
)
encoder_bi_lstm2_output = [
    encoder_output2, forward_h2, forward_c2, backward_h2, backward_c2
]

# encoder lstm 3
encoder_bi_lstm = Bidirectional(
    LSTM(
        latent_dim,
        return_sequences=True,
        return_state=True,
        dropout=0.4,
        recurrent_dropout=0.4,
        name='encoder_lstm_3'
    ),
    name='encoder_bidirectional_lstm_3'
)
encoder_output, *encoder_final_states = encoder_bi_lstm(encoder_output2)

# ====================
# 🌈 Decoder
# ====================

# Set up the decoder, using `encoder_states` as initial state.

decoder_input = Input(shape=(None, ))

# decoder embedding layer
decoder_embedding_layer = Embedding(
    y_vocab_size,
    embedding_dim,
    embeddings_initializer=tf.keras.initializers.Constant(y_embedding_matrix),
    trainable=False,
    name='decoder_embedding'
)
decoder_embedding = decoder_embedding_layer(decoder_input)

decoder_lstm = LSTM(
    latent_dim,
    return_sequences=True,
    return_state=True,
    dropout=0.4,
    recurrent_dropout=0.2,
    name='decoder_lstm_1'
)
decoder_output, *decoder_final_states = decoder_lstm(
    decoder_embedding, initial_state=encoder_final_states[:2]
)  # taking only the forward states

# dense layer
decoder_dense = TimeDistributed(
    Dense(y_vocab_size, activation='softmax')
)
decoder_output = decoder_dense(decoder_output)

# ====================
# ⚡ Model
# ====================
model = Model([encoder_input, decoder_input], decoder_output, name='seq2seq_model_with_bidirectional_lstm')
model.summary()

optimizer = tf.keras.optimizers.RMSprop(learning_rate=0.001)
model.compile(
    optimizer=optimizer,
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

return {
    'model': model,
    'inputs': {
        'encoder': encoder_input,
        'decoder': decoder_input
    },
    'outputs': {
        'encoder': encoder_output,
        'decoder': decoder_output
    },
    'states': {
        'encoder': encoder_final_states,
```

```
            'decoder': decoder_final_states
        },
        'layers': {
            'decoder': {
                'embedding': decoder_embedding_layer,
                'last_decoder_lstm': decoder_lstm,
                'dense': decoder_dense
            }
        }
    }
```

```
seq2seq = build_seq2seq_model_with_just_lstm(
    embedding_dim, latent_dim, max_text_len,
    x_vocab_size, y_vocab_size,
    x_embedding_matrix, y_embedding_matrix
)
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 42)]         0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 42, 300)      29932800    input_1[0][0]
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, None)]       0
__________________________________________________________________________________________________
lstm (LSTM)                     [(None, 42, 240), (N 519360      embedding[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, None, 300)    11257200    input_2[0][0]
__________________________________________________________________________________________________
lstm_1 (LSTM)                   [(None, 42, 240), (N 461760      lstm[0][0]
__________________________________________________________________________________________________
lstm_2 (LSTM)                   [(None, None, 240),  519360      embedding_1[0][0]
                                                                 lstm_1[0][1]
                                                                 lstm_1[0][2]
__________________________________________________________________________________________________
time_distributed (TimeDistribut (None, None, 37524)  9043284     lstm_2[0][0]
==================================================================================================
Total params: 51,733,764
Trainable params: 21,800,964
Non-trainable params: 29,932,800
__________________________________________________________________________________________________
```

If you want to change `model` then just change the `function name` above.

```
model = seq2seq['model']

encoder_input = seq2seq['inputs']['encoder']
decoder_input = seq2seq['inputs']['decoder']

encoder_output = seq2seq['outputs']['encoder']
decoder_output = seq2seq['outputs']['decoder']

encoder_final_states = seq2seq['states']['encoder']
decoder_final_states = seq2seq['states']['decoder']

decoder_embedding_layer = seq2seq['layers']['decoder']['embedding']
last_decoder_lstm = seq2seq['layers']['decoder']['last_decoder_lstm']
decoder_dense = seq2seq['layers']['decoder']['dense']
```

```
model.layers[-2].input
```

```
[<KerasTensor: shape=(None, None, 300) dtype=float32 (created by layer 'embedding_1')>,
 <KerasTensor: shape=(None, 240) dtype=float32 (created by layer 'lstm_1')>,
 <KerasTensor: shape=(None, 240) dtype=float32 (created by layer 'lstm_1')>]
```

```
callbacks = [
    EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=2),
    ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=2, min_lr=0.000001, verbose=1),
]
```

Use a `tuple` instead of `list` in `validation_parameter` in `model.fit()`, to know the reason reading this post.

```
history = model.fit(
    [x_train_padded, y_train_padded[:, :-1]],
    y_train_padded.reshape(y_train_padded.shape[0], y_train_padded.shape[1], 1)[:, 1:],
    epochs=num_epochs,
    batch_size=128 * tpu_strategy.num_replicas_in_sync,
    callbacks=callbacks,
    validation_data=(
        [x_val_padded, y_val_padded[:, :-1]],
        y_val_padded.reshape(y_val_padded.shape[0], y_val_padded.shape[1], 1)[:, 1:]
```

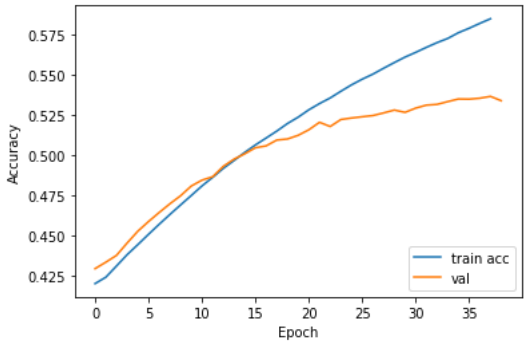
```
    )
)
```

```
Epoch 1/50
89/89 [==============================] - 37s 234ms/step - loss: 7.0099 - accuracy: 0.2499 - val_loss: 5.1364 - val_accuracy: 0.4292
Epoch 2/50
89/89 [==============================] - 5s 52ms/step - loss: 5.2058 - accuracy: 0.4184 - val_loss: 4.9203 - val_accuracy: 0.4332
Epoch 3/50
89/89 [==============================] - 5s 52ms/step - loss: 5.0199 - accuracy: 0.4224 - val_loss: 4.7193 - val_accuracy: 0.4374
Epoch 4/50
89/89 [==============================] - 5s 52ms/step - loss: 4.7725 - accuracy: 0.4284 - val_loss: 4.4864 - val_accuracy: 0.4452
Epoch 5/50
89/89 [==============================] - 5s 52ms/step - loss: 4.5356 - accuracy: 0.4364 - val_loss: 4.3093 - val_accuracy: 0.4525
Epoch 6/50
89/89 [==============================] - 5s 51ms/step - loss: 4.3404 - accuracy: 0.4430 - val_loss: 4.1756 - val_accuracy: 0.4585
Epoch 7/50
89/89 [==============================] - 5s 52ms/step - loss: 4.1772 - accuracy: 0.4489 - val_loss: 4.0440 - val_accuracy: 0.4642
Epoch 8/50
89/89 [==============================] - 5s 52ms/step - loss: 4.0264 - accuracy: 0.4555 - val_loss: 3.9389 - val_accuracy: 0.4696
Epoch 9/50
89/89 [==============================] - 5s 51ms/step - loss: 3.8925 - accuracy: 0.4619 - val_loss: 3.8454 - val_accuracy: 0.4746
Epoch 10/50
89/89 [==============================] - 5s 51ms/step - loss: 3.7701 - accuracy: 0.4679 - val_loss: 3.7627 - val_accuracy: 0.4806
Epoch 11/50
89/89 [==============================] - 4s 51ms/step - loss: 3.6613 - accuracy: 0.4740 - val_loss: 3.6932 - val_accuracy: 0.4843
Epoch 12/50
89/89 [==============================] - 5s 51ms/step - loss: 3.5623 - accuracy: 0.4795 - val_loss: 3.6662 - val_accuracy: 0.4864
Epoch 13/50
89/89 [==============================] - 5s 52ms/step - loss: 3.4690 - accuracy: 0.4854 - val_loss: 3.5769 - val_accuracy: 0.4929
Epoch 14/50
89/89 [==============================] - 5s 51ms/step - loss: 3.3787 - accuracy: 0.4913 - val_loss: 3.5259 - val_accuracy: 0.4974
Epoch 15/50
89/89 [==============================] - 5s 52ms/step - loss: 3.3071 - accuracy: 0.4955 - val_loss: 3.4754 - val_accuracy: 0.5007
Epoch 16/50
89/89 [==============================] - 5s 51ms/step - loss: 3.2269 - accuracy: 0.5016 - val_loss: 3.4391 - val_accuracy: 0.5045
Epoch 17/50
89/89 [==============================] - 5s 51ms/step - loss: 3.1570 - accuracy: 0.5059 - val_loss: 3.4175 - val_accuracy: 0.5056
Epoch 18/50
89/89 [==============================] - 5s 51ms/step - loss: 3.0902 - accuracy: 0.5107 - val_loss: 3.3699 - val_accuracy: 0.5093
Epoch 19/50
89/89 [==============================] - 5s 51ms/step - loss: 3.0331 - accuracy: 0.5145 - val_loss: 3.3580 - val_accuracy: 0.5099
Epoch 20/50
89/89 [==============================] - 5s 52ms/step - loss: 2.9695 - accuracy: 0.5195 - val_loss: 3.3378 - val_accuracy: 0.5122
Epoch 21/50
89/89 [==============================] - 5s 52ms/step - loss: 2.9129 - accuracy: 0.5237 - val_loss: 3.3000 - val_accuracy: 0.5157
Epoch 22/50
89/89 [==============================] - 5s 52ms/step - loss: 2.8612 - accuracy: 0.5279 - val_loss: 3.2586 - val_accuracy: 0.5203
Epoch 23/50
89/89 [==============================] - 5s 51ms/step - loss: 2.8066 - accuracy: 0.5330 - val_loss: 3.2670 - val_accuracy: 0.5177
Epoch 24/50
89/89 [==============================] - 5s 52ms/step - loss: 2.7592 - accuracy: 0.5363 - val_loss: 3.2373 - val_accuracy: 0.5221
Epoch 25/50
89/89 [==============================] - 5s 52ms/step - loss: 2.7154 - accuracy: 0.5400 - val_loss: 3.2215 - val_accuracy: 0.5230
Epoch 26/50
89/89 [==============================] - 5s 52ms/step - loss: 2.6706 - accuracy: 0.5444 - val_loss: 3.2073 - val_accuracy: 0.5238
Epoch 27/50
89/89 [==============================] - 5s 52ms/step - loss: 2.6254 - accuracy: 0.5487 - val_loss: 3.1981 - val_accuracy: 0.5245
Epoch 28/50
89/89 [==============================] - 5s 51ms/step - loss: 2.5876 - accuracy: 0.5516 - val_loss: 3.1890 - val_accuracy: 0.5262
Epoch 29/50
89/89 [==============================] - 5s 52ms/step - loss: 2.5532 - accuracy: 0.5550 - val_loss: 3.1716 - val_accuracy: 0.5280
```

**Plotting model's performance**

```
# Accuracy
plt.plot(history.history['accuracy'][1:], label='train acc')
plt.plot(history.history['val_accuracy'], label='val')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
```

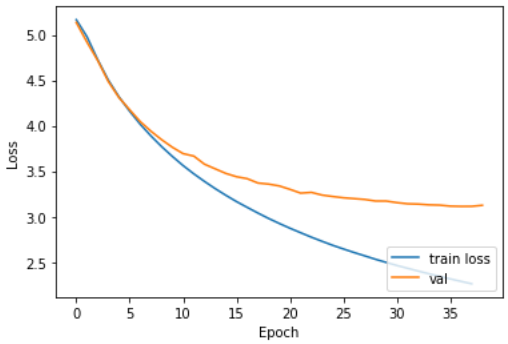
```
<matplotlib.legend.Legend at 0x7f764b240f50>
```




```
# Loss
plt.plot(history.history['loss'][1:], label='train loss')
```

```
plt.plot(history.history['val_loss'], label='val')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='lower right')
```

```
<matplotlib.legend.Legend at 0x7f7647d134d0>
```




## Inference


```
# Next, let's build the dictionary to convert the index to word for target and source vocabulary:
reverse_target_word_index = y_tokenizer.index_word
reverse_source_word_index = x_tokenizer.index_word
target_word_index = y_tokenizer.word_index
```

```
def build_seq2seq_model_with_just_lstm_inference(
    max_text_len, latent_dim, encoder_input, encoder_output,
    encoder_final_states, decoder_input, decoder_output,
    decoder_embedding_layer, decoder_dense, last_decoder_lstm
):
    # Encode the input sequence to get the feature vector
    encoder_model = Model(
        inputs=encoder_input, outputs=[encoder_output] + encoder_final_states
    )

    # Decoder setup
    # Below tensors will hold the states of the previous time step
    decoder_state_input_h = Input(shape=(latent_dim, ))
    decoder_state_input_c = Input(shape=(latent_dim, ))
    decoder_hidden_state_input = Input(shape=(max_text_len, latent_dim))

    # Get the embeddings of the decoder sequence
    decoder_embedding = decoder_embedding_layer(decoder_input)

    # To predict the next word in the sequence, set the initial
    # states to the states from the previous time step
    decoder_output, *decoder_states = last_decoder_lstm(
        decoder_embedding,
        initial_state=[decoder_state_input_h, decoder_state_input_c]
    )

    # A dense softmax layer to generate prob dist. over the target vocabulary
    decoder_output = decoder_dense(decoder_output)

    # Final decoder model
    decoder_model = Model(
        [decoder_input] + [decoder_hidden_state_input, decoder_state_input_h, decoder_state_input_c],
        [decoder_output] + decoder_states
    )

    return (encoder_model, decoder_model)
```

Useful `stackoverflow` post to understand `inference` process when using `bidirectional lstms` in `encoder` and `decoder` in the training model.

```
def build_seq2seq_model_with_bidirectional_lstm_inference(
```

```
    max_text_len, latent_dim, encoder_input, encoder_output,
    encoder_final_states, decoder_input, decoder_output,
    decoder_embedding_layer, decoder_dense, last_decoder_bi_lstm
):

    # Encode the input sequence to get the feature vector
    encoder_model = Model(
        inputs=encoder_input, outputs=[encoder_output] + encoder_final_states
    )

    # Decoder setup
    # Below tensors will hold the states of the previous time step
    decoder_state_forward_input_h = Input(shape=(latent_dim, ))
    decoder_state_forward_input_c = Input(shape=(latent_dim, ))
    decoder_state_backward_input_h = Input(shape=(latent_dim, ))
    decoder_state_backward_input_c = Input(shape=(latent_dim, ))

    # Create the hidden input layer with twice the latent dimension,
    # since we are using bi - directional LSTM's we will get
    # two hidden states and two cell states
    decoder_hidden_state_input = Input(shape=(max_text_len, latent_dim * 2))

    decoder_initial_state = [
        decoder_state_forward_input_h, decoder_state_forward_input_c,
        decoder_state_backward_input_h, decoder_state_backward_input_c
    ]

    # Get the embeddings of the decoder sequence
    decoder_embedding = decoder_embedding_layer(decoder_input)

    # To predict the next word in the sequence, set the initial
    # states to the states from the previous time step
    decoder_output, *decoder_states = last_decoder_bi_lstm(
        decoder_embedding, initial_state=decoder_initial_state
    )

    # A dense softmax layer to generate prob dist. over the target vocabulary
    decoder_output = decoder_dense(decoder_output)

    # Final decoder model
    decoder_model = Model(
        [decoder_input] + [decoder_hidden_state_input] + decoder_initial_state,
        [decoder_output] + decoder_states
    )

    return (encoder_model, decoder_model)
```

```
def build_hybrid_seq2seq_model_inference(
    max_text_len, latent_dim, encoder_input, encoder_output,
    encoder_final_states, decoder_input, decoder_output,
    decoder_embedding_layer, decoder_dense, last_decoder_bi_lstm
):

    # Encode the input sequence to get the feature vector
    encoder_model = Model(
        inputs=encoder_input, outputs=[encoder_output] + encoder_final_states
    )

    # Decoder setup
    # Below tensors will hold the states of the previous time step
    decoder_state_forward_input_h = Input(shape=(latent_dim, ))
    decoder_state_forward_input_c = Input(shape=(latent_dim, ))
    # decoder_state_backward_input_h = Input(shape=(latent_dim, ))
    # decoder_state_backward_input_c = Input(shape=(latent_dim, ))

    # Create the hidden input layer with twice the latent dimension,
    # since we are using bi - directional LSTM's we will get
    # two hidden states and two cell states
    decoder_hidden_state_input = Input(shape=(max_text_len, latent_dim * 2))

    decoder_initial_state = [
        decoder_state_forward_input_h, decoder_state_forward_input_c,
        #decoder_state_backward_input_h, decoder_state_backward_input_c
    ]

    # Get the embeddings of the decoder sequence
    decoder_embedding = decoder_embedding_layer(decoder_input)

    # To predict the next word in the sequence, set the initial
    # states to the states from the previous time step
    decoder_output, *decoder_states = last_decoder_bi_lstm(
        decoder_embedding, initial_state=decoder_initial_state
    )
```

```
    # A dense softmax layer to generate prob dist. over the target vocabulary
    decoder_output = decoder_dense(decoder_output)

    # Final decoder model
    decoder_model = Model(
        [decoder_input] + [decoder_hidden_state_input] + decoder_initial_state,
        [decoder_output] + decoder_states
    )

    return (encoder_model, decoder_model)
```

```
encoder_model, decoder_model = build_seq2seq_model_with_just_lstm_inference(
    max_text_len, latent_dim, encoder_input, encoder_output,
    encoder_final_states, decoder_input, decoder_output,
    decoder_embedding_layer, decoder_dense, last_decoder_lstm
)
```

```
encoder_model.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 42)]              0
_________________________________________________________________
embedding (Embedding)        (None, 42, 300)           29932800
_________________________________________________________________
lstm (LSTM)                  [(None, 42, 240), (None,  519360
_________________________________________________________________
lstm_1 (LSTM)                [(None, 42, 240), (None,  461760
=================================================================
Total params: 30,913,920
Trainable params: 981,120
Non-trainable params: 29,932,800
_________________________________________________________________
```

```
decoder_model.summary()
```

```
Model: "model_2"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_2 (InputLayer)            [(None, None)]       0
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, None, 300)    11257200    input_2[0][0]
__________________________________________________________________________________________________
input_3 (InputLayer)            [(None, 240)]        0
__________________________________________________________________________________________________
input_4 (InputLayer)            [(None, 240)]        0
__________________________________________________________________________________________________
lstm_2 (LSTM)                   [(None, None, 240),  519360      embedding_1[1][0]
                                                                 input_3[0][0]
                                                                 input_4[0][0]
__________________________________________________________________________________________________
input_5 (InputLayer)            [(None, 42, 240)]    0
__________________________________________________________________________________________________
time_distributed (TimeDistribut (None, None, 37524)  9043284     lstm_2[1][0]
==================================================================================================
Total params: 20,819,844
Trainable params: 20,819,844
Non-trainable params: 0
__________________________________________________________________________________________________
```

```
decoder_model.layers[-3].input
```

```
[<KerasTensor: shape=(None, None, 300) dtype=float32 (created by layer 'embedding_1')>,
 <KerasTensor: shape=(None, 240) dtype=float32 (created by layer 'lstm_1')>,
 <KerasTensor: shape=(None, 240) dtype=float32 (created by layer 'lstm_1')>]
```

Converting from `sequence to text` for model `with just LSTM's` and for model `with Bidirectional LSTM's`.

```
def decode_sequence_seq2seq_model_with_just_lstm(
    input_sequence, encoder_model, decoder_model
):
    # Encode the input as state vectors.
    e_out, e_h, e_c = encoder_model.predict(input_sequence)

    # Generate empty target sequence of length 1.
    target_seq = np.zeros((1, 1))

    # Populate the first word of target sequence with the start word.
    target_seq[0, 0] = target_word_index[start_token]

    stop_condition = False
    decoded_sentence = ''

    while not stop_condition:
        output_tokens, h, c = decoder_model.predict(
            [target_seq] + [e_out, e_h, e_c]
        )

        # Sample a token
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_token = reverse_target_word_index[sampled_token_index]

        if sampled_token != end_token:
            decoded_sentence += ' ' + sampled_token

        # Exit condition: either hit max length or find stop word.
        if (sampled_token == end_token) or (len(decoded_sentence.split()) >= (max_summary_len - 1)):
            stop_condition = True

        # Update the target sequence (of length 1).
        target_seq = np.zeros((1, 1))
        target_seq[0, 0] = sampled_token_index

        # Update internal states
        e_h, e_c = h, c

    return decoded_sentence
```

```
def decode_sequence_seq2seq_model_with_bidirectional_lstm(
    input_sequence, encoder_model, decoder_model
):
    # Encode the input as state vectors.
    e_out, *state_values = encoder_model.predict(input_sequence)

    # Generate empty target sequence of length 1.
    target_seq = np.zeros((1, 1))

    # Populate the first word of target sequence with the start word.
    target_seq[0, 0] = target_word_index[start_token]

    stop_condition = False
    decoded_sentence = ''

    while not stop_condition:
        output_tokens, *decoder_states = decoder_model.predict(
            [target_seq] + [e_out] + state_values
        )

        # Sample a token
        sampled_token_index = np.argmax(output_tokens[0, -1, :]) # Greedy Search
        sampled_token = reverse_target_word_index[sampled_token_index + 1]

        if sampled_token != end_token:
            decoded_sentence += ' ' + sampled_token

        # Exit condition: either hit max length or find stop word.
        if (sampled_token == end_token) or (len(decoded_sentence.split()) >= (max_summary_len - 1)):
            stop_condition = True

        # Update the target sequence (of length 1).
        target_seq = np.zeros((1, 1))
        target_seq[0, 0] = sampled_token_index

        # Update internal states
        state_values = decoder_states

    return decoded_sentence
```

```
def decode_sequence_hybrid_seq2seq_model(
    input_sequence, encoder_model, decoder_model
):
    # Encode the input as state vectors.
```

```
    e_out, *state_values = encoder_model.predict(input_sequence)

    # Generate empty target sequence of length 1.
    target_seq = np.zeros((1, 1))

    # Populate the first word of target sequence with the start word.
    target_seq[0, 0] = target_word_index[start_token]

    stop_condition = False
    decoded_sentence = ''

    while not stop_condition:
        output_tokens, *decoder_states = decoder_model.predict(
            [target_seq] + [e_out] + state_values[:2]
        )

        # Sample a token
        sampled_token_index = np.argmax(output_tokens[0, -1, :]) # Greedy Search
        sampled_token = reverse_target_word_index[sampled_token_index + 1]

        if sampled_token != end_token:
            decoded_sentence += ' ' + sampled_token

        # Exit condition: either hit max length or find stop word.
        if (sampled_token == end_token) or (len(decoded_sentence.split()) >= (max_summary_len - 1)):
            stop_condition = True

        # Update the target sequence (of length 1).
        target_seq = np.zeros((1, 1))
        target_seq[0, 0] = sampled_token_index

        # Update internal states
        state_values = decoder_states

    return decoded_sentence
```

```
def seq2summary(input_sequence):
    new_string = ''
    for i in input_sequence:
        if (
            (i != 0 and i != target_word_index[start_token]) and
            (i != target_word_index[end_token])
        ):
            new_string = new_string + reverse_target_word_index[i] + ' '
    return new_string
```

```
def seq2text(input_sequence):
    new_string = ''
    for i in input_sequence:
        if i != 0:
            new_string = new_string + reverse_source_word_index[i] + ' '
    return new_string
```

```
l = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

if len(l) % 3 != 0:
    while len(l) % 3 != 0:
        l.append(0)
print(l)

lst_i = 3
for i in range(0, len(l), 3):
    print(l[i:i + lst_i])

print(' '.join(['', 'james', 'ethan', '', 'tony']))
print(' '.join(' '.join(['', 'james', 'ethan', '', 'tony']).split()))
```

```
     [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 0, 0]
     [1, 2, 3]
     [4, 5, 6]
     [7, 8, 9]
     [10, 0, 0]
      james ethan  tony
     james ethan tony
```

For predicting `unseen` data pass `decode_sequence` function for which you want to decode.

```
def predict_text(text, decode_sequence, encoder_model, decoder_model):
    original_text = text
    text = clean_text([text])  # generator
    text_list = original_text.split()
```

```
    if len(text_list) <= max_text_len:
        text = expand_contractions(text)
        text = clean_text(text)
        text = f'_START_ {text} _END_'
        text = f'{start_token} {text} {end_token}'

        seq = x_tokenizer.texts_to_sequences([' '.join(text_list)])
        padded = pad_sequences(seq, maxlen=max_text_len, padding='post')
        pred_summary = decode_sequence(
            padded.reshape(1, max_text_len), encoder_model, decoder_model
        )
        return pred_summary
    else:
        pred_summary = ''

        # breaking long texts to individual max_text_len texts and predicting on them
        while len(text_list) % max_text_len == 0:
            text_list.append('')

        lst_i = max_text_len
        for i in range(0, len(text_list), max_text_len):
            _text_list = original_text.split()[i:i + lst_i]
            _text = ' '.join(_text_list)
            _text = ' '.join(
                _text.split()
            )  # to remove spaces that were added to make len(text_list) % max_text_len == 0

            _text = expand_contractions(_text)
            _text = clean_text(_text)  # generator
            _text = f'_START_ {_text} _END_'
            _text = f'{start_token} {_text} {end_token}'
            # print(_text, '\n')

            _seq = x_tokenizer.texts_to_sequences([_text])
            _padded = pad_sequences(_seq, maxlen=max_text_len, padding='post')
            _pred = decode_sequence(
                _padded.reshape(1, max_text_len), encoder_model, decoder_model
            )
            pred_summary += ' ' + ' '.join(_pred.split()[1:-2])
            pred_summary = ' '.join(pred_summary.split())

        return pred_summary
```

## 🔮 Predictions

```
# Testing on training data
for i in range(0, 15):
    print(f"# {i+1} News: ", seq2text(x_train_padded[i]))
    print("Original summary: ", seq2summary(y_train_padded[i]))
    print(
        "Predicted summary: ",
        decode_sequence_seq2seq_model_with_just_lstm(
            x_train_padded[i].reshape(1, max_text_len), encoder_model,
            decoder_model
        )
    )
    print()
```

```
 # 1 News:  spanish fashion retailer zara withdrawn skirt website facing criticism featuring pepe frog internet mtheme turned symbol white natio
 Original summary:  start zara withdraws skirt featuring pepe frog altright mtheme end
 Predicted summary:   start zara slammed racist ad ad featuring pepe end

 # 2 News:  elections fill vacant seats rajya sabha held today mthembers elected unopposed one seats facing byelections kerala mp resigned mber
 Original summary:  start explained details rajya sabha election held today end
 Predicted summary:   start rajya sabha seats ls polls may get majority end

 # 3 News:  first posthumous nobel prize awarded swedens erik karlfeldt literature posthumous nobel peace prize given youngest united nations se
 Original summary:  start rare cases nobel prize given death end
 Predicted summary:   start nobel prize winner awarded honorary drate end
```

```
# 4 News:  titanic actors leonardo dico kate winslet auctioning dinner two ththem leonardo dico foundations fourth annual gala wednesday charity
Original summary:  start titanic stars leonardo kate auction dinner date charity end
Predicted summary:   start leonardo dico painting kate kate middletons aaa1 crore end

# 5 News:  zaheer khans fiancaae actor sagarika ghatge posted picture instagram couple zaheer flaunts new cleanshaven look sagarika captioned pi
Original summary:  start zaheer broke beard rather well says fiancaae sagarika end
Predicted summary:   start zaheer sagarika ghatge shares pic zaheer sagarika ghatge end

# 6 News:  killer whale stopped carrying dead newborn calf least days covered km according researchers killer whales known carry dead calves wee
Original summary:  start killer whale abandons dead newborn calf days end
Predicted summary:   start whale found dead whale game thrones found dead end

# 7 News:  letter election commission madhya pradesh congress committee alleged pm narendra modi violated model code conduct speech pollbound st
Original summary:  start pm modi violated poll code mp congress letter ec end
Predicted summary:   start pm modi violated code mp cm violated code violation end

# 8 News:  malls hotels restaurants bengaluru directed provide free clean drinking water customers citys civic body bruhat bengaluru mahanagara
Original summary:  start bengaluru malls restaurants provide free drinking water end
Predicted summary:   start bengaluru restaurants malls free free toilets end

# 9 News:  reserve bank new zealand acting governor grant spencer said bitcoin unstable useful future adding cryptocurrency looks rthemarkably l
Original summary:  start bitcoin unstable useful nz central bank chief end
Predicted summary:   start bitcoin pretty new zealand says imf end

# 10 News:  indias batting coach sanjay bangar said void team coach anil kumbles resignation team coping well professionals things part parcel c
Original summary:  start void kumbles exit india batting coach end
Predicted summary:   start kumbles coach kumbles exit india coach end

# 11 News:  video showing african woman stripping crowded delhi metro feud passengers surfaced social media woman along another african woman al
Original summary:  start african lady strips delhi metro feud passengers end
Predicted summary:   start woman assaults woman front train delhi airport end

# 12 News:  rbi governor urjit patel resigned post monday citing personal reasons effective immediately privilege honour serve reserve bank indi
Original summary:  start rbi governor urjit patel quits citing personal reasons end
Predicted summary:   start rbi governor urjit patel resigns rbi governor end

# 13 News:  maharashtras pune topped ease living index rankings released ministry housing urban affairs monday navi mumbai grabbed second spot g
Original summary:  start pune ranked first ease living index delhi end
Predicted summary:   start mumbai ranked best cities list ease index end

# 14 News:  changes food served onboard air india flights help airline save aaa1 crore annually mos civil aviation jayant sinha said air india r
Original summary:  start changes food save air india aaa1 crore yearly centre end
Predicted summary:   start air india get aaa1 crore air india flight end

# 15 News:  chinese workers pakistans khanewal attacked policthemen deployed security denied permission visit red light area without accompanied
```

```
# Testing on validation data
for i in range(0, 15):
    print(f"# {i+1} News: ", seq2text(x_val_padded[i]))
    print("Original summary: ", seq2summary(y_val_padded[i]))
    print(
        "Predicted summary: ",
        decode_sequence_seq2seq_model_with_just_lstm(
            x_val_padded[i].reshape(1, max_text_len), encoder_model,
            decoder_model
        )
    )
    print()
```

```
# 1 News:  italian space agency scientists expressed interest developing robotic technology bring asteroid beyond lunar orbit back closer reach
Original summary:  start scientists propose bringing asteroid closer earth end
Predicted summary:   start nasa planning spacecraft saturn moon mission mercury end

# 2 News:  palestinian president mahmoud abbas said social behaviour jews responsible holocaust addressing meeting ramallah monday abbas claimed
Original summary:  start jews social behaviour led holocaust palestine president end
Predicted summary:   start israel calls jerusalthem jews jews jews prez end

# 3 News:  batsman hanuma vihari become first andhra cricketer years picked indias test squad yearold righthanded batsman firstclass average hig
Original summary:  start hanuma vihari uncapped batsman india test squad end
Predicted summary:   start player scores st indian cricketer score tests end

# 4 News:  speaking dealing attention received following acting debut sara ali khan said go back home normal girl normal household kedarnath act
Original summary:  start go back home normal girl normal household sara end
Predicted summary:   start want see home home sara ali khan end

# 5 News:  londonbound pakistan international airlines pia flight delayed three hours saturday fight broke pilot cabin crew mthember pilot refus
Original summary:  start pakistan flight delayed hours fight end
Predicted summary:   start pak pilot forced sleep flight leaves back back end

# 6 News:  international court justice ordered us lift sanctions iran linked humanitarian goods civil aviation iran claims us sanctions violate
Original summary:  start world court orders us lift sanctions iran affecting aid end
Predicted summary:   start iran court bans us sanctions us sanctions end

# 7 News:  according reports actress aishwarya rai bachchan may replace fatherinlaw amitabh bachchan host ninth season television game show kaun
Original summary:  start aishwarya replace amitabh kbc host reports end
Predicted summary:   start aishwarya rai play big bs entry season reports end

# 8 News:  thirtyyearold patil become first visuallychallenged woman ias officer india joined posting assistant collector keralas ernakulam dist
Original summary:  start yrold becomes st visuallychallenged woman ias officer end
Predicted summary:   start woman becomes indias first woman ias officer end
```

```
# 9 News:  women child development ministry proposed compulsory boys girls study home science physical education school order promote gender ser
Original summary:  start home science may become mandatory boys school end
Predicted summary:   start education education ministry allows girls girls end

# 10 News:  tata groups titan considering external internal candidates formal search find successor longtime managing director md bhaskar bhat e
Original summary:  start titan considering internal external candidates replace md end
Predicted summary:   start tata group names mistry appointed independent chairman end

# 11 News:  tesla spacex ceo elon musk wednesday took twitter announce quit us president donald trumps advisory council trump withdraws paris cl
Original summary:  start musk quit trumps council us leaves paris climate deal end
Predicted summary:   start pichai trump meet discuss us president end

# 12 News:  us cancer researcher pleaded guilty conspiring steal trade secrets glaxosmithkline gsk benefit chinese company created yu naturalise
Original summary:  start researcher admits plot steal gsk secrets sell china end
Predicted summary:   start us man sues us trade trade secrets secrets end

# 13 News:  talking olas takeover startup interview cofounder raghunandan wednesday said identity taken away added yes made money gave us financ
Original summary:  start identity taken away taxiforsure founder ola deal end
Predicted summary:   start credit suisse says us back back back paytm ceo end

# 14 News:  one five people killed chartered plane crash mumbai pradeep singh rajput survived plane crash neighbour said revealed rajput told pl
Original summary:  start pilot killed mumbai survived crash report end
Predicted summary:   start killed injured plane crash badrinath express end

# 15 News:  unaccompanied yearold boy rthemoved overbooked easyjet flight left alone departure gate london gatwick airport thursday child caspe
```

## 🎁 Saving the model

```
# HDF5 format
model.save('model.h5')
encoder_model.save('encoder_model.h5')
decoder_model.save('decoder_model.h5')
```

## 🏌️ Running all the 3 different models

After understanding how all the pieces work, running all the `3 models` to understand how it `performs` and its `results`.

**Here there 3 different training models**

- `build_seq2seq_model_with_just_lstm` - **Seq2Seq model with just LSTMs**. Both `encoder` and `decoder` have just `LSTM`s.
- `build_seq2seq_model_with_bidirectional_lstm` - **Seq2Seq model with Bidirectional LSTMs**. Both `encoder` and `decoder` have `Bidirectional LSTM`s.
- `build_hybrid_seq2seq_model` - **Seq2Seq model with hybrid architecture**. Here `encoder` has `Bidirectional LSTM`s while `decoder` has just `LSTM`s.

**Inference methods for the 3 different learning models - just add `_inference` as `prefix`**

- `build_seq2seq_model_with_just_lstm_inference`
- `build_seq2seq_model_with_bidirectional_lstm_inference`
- `build_hybrid_seq2seq_model_inference`

**Decoding sequence for the 3 different learning models - just add `decode_sequence_` as `suffix`**

- `decode_sequence_build_seq2seq_model_with_just_lstm`
- `decode_sequence_build_seq2seq_model_with_bidirectional_lstm`
- `decode_sequence_build_hybrid_seq2seq_model`

```
models_info = {
    'just_lstm': {
        'model': build_seq2seq_model_with_just_lstm,
        'inference': build_seq2seq_model_with_just_lstm_inference,
        'decode_sequence': decode_sequence_seq2seq_model_with_just_lstm
    },
    'bidirectional_lstm': {
        'model': build_seq2seq_model_with_bidirectional_lstm,
        'inference': build_seq2seq_model_with_bidirectional_lstm_inference,
        'decode_sequence': decode_sequence_seq2seq_model_with_bidirectional_lstm
    },
    'hybrid_model': {
        'model': build_hybrid_seq2seq_model,
        'inference': build_hybrid_seq2seq_model_inference,
        'decode_sequence': decode_sequence_hybrid_seq2seq_model
    }
}
```

**Model with just LSTMs**

```
model_func = models_info['just_lstm']['model']
inference_func = models_info['just_lstm']['inference']
decode_sequence_func = models_info['just_lstm']['decode_sequence']
```

```
seq2seq = model_func(
    embedding_dim, latent_dim, max_text_len,
    x_vocab_size, y_vocab_size,
    x_embedding_matrix, y_embedding_matrix
)

model = seq2seq['model']

encoder_input = seq2seq['inputs']['encoder']
decoder_input = seq2seq['inputs']['decoder']

encoder_output = seq2seq['outputs']['encoder']
decoder_output = seq2seq['outputs']['decoder']

encoder_final_states = seq2seq['states']['encoder']
decoder_final_states = seq2seq['states']['decoder']

decoder_embedding_layer = seq2seq['layers']['decoder']['embedding']
last_decoder_lstm = seq2seq['layers']['decoder']['last_decoder_lstm']
decoder_dense = seq2seq['layers']['decoder']['dense']

model.summary()
```

```
Model: "model_3"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_6 (InputLayer)            [(None, 42)]         0
__________________________________________________________________________________________________
embedding_2 (Embedding)         (None, 42, 300)      29932800    input_6[0][0]
__________________________________________________________________________________________________
input_7 (InputLayer)            [(None, None)]       0
__________________________________________________________________________________________________
```

```
lstm_3 (LSTM)                   [(None, 42, 240), (N 519360      embedding_2[0][0]
__________________________________________________________________________________________________
embedding_3 (Embedding)         (None, None, 300)    11257200    input_7[0][0]
__________________________________________________________________________________________________
lstm_4 (LSTM)                   [(None, 42, 240), (N 461760      lstm_3[0][0]
__________________________________________________________________________________________________
lstm_5 (LSTM)                   [(None, None, 240),  519360      embedding_3[0][0]
                                                                 lstm_4[0][1]
                                                                 lstm_4[0][2]
__________________________________________________________________________________________________
time_distributed_1 (TimeDistrib (None, None, 37524)  9043284     lstm_5[0][0]
==================================================================================================
Total params: 51,733,764
Trainable params: 21,800,964
Non-trainable params: 29,932,800
__________________________________________________________________________________________________
Model: "model_3"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_6 (InputLayer)            [(None, 42)]         0
__________________________________________________________________________________________________
embedding_2 (Embedding)         (None, 42, 300)      29932800    input_6[0][0]
__________________________________________________________________________________________________
input_7 (InputLayer)            [(None, None)]       0
__________________________________________________________________________________________________
lstm_3 (LSTM)                   [(None, 42, 240), (N 519360      embedding_2[0][0]
__________________________________________________________________________________________________
embedding_3 (Embedding)         (None, None, 300)    11257200    input_7[0][0]
__________________________________________________________________________________________________
lstm_4 (LSTM)                   [(None, 42, 240), (N 461760      lstm_3[0][0]
__________________________________________________________________________________________________
lstm_5 (LSTM)                   [(None, None, 240),  519360      embedding_3[0][0]
                                                                 lstm_4[0][1]
                                                                 lstm_4[0][2]
__________________________________________________________________________________________________
time_distributed_1 (TimeDistrib (None, None, 37524)  9043284     lstm_5[0][0]
==================================================================================================
Total params: 51,733,764
Trainable params: 21,800,964
Non-trainable params: 29,932,800
__________________________________________________________________________________________________
```

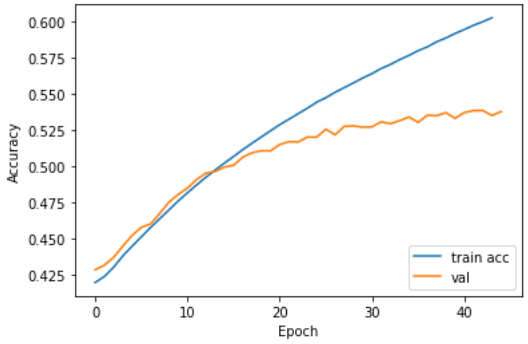
```
history = model.fit(
    [x_train_padded, y_train_padded[:, :-1]],
    y_train_padded.reshape(y_train_padded.shape[0], y_train_padded.shape[1], 1)[:, 1:],
    epochs=num_epochs,
    batch_size=128 * tpu_strategy.num_replicas_in_sync,
    callbacks=callbacks,
    validation_data=(
        [x_val_padded, y_val_padded[:, :-1]],
        y_val_padded.reshape(y_val_padded.shape[0], y_val_padded.shape[1], 1)[:, 1:]
    )
)
```

```
Epoch 1/50
89/89 [==============================] - 37s 227ms/step - loss: 7.0021 - accuracy: 0.2465 - val_loss: 5.1189 - val_accuracy: 0.4284
Epoch 2/50
89/89 [==============================] - 5s 51ms/step - loss: 5.2113 - accuracy: 0.4181 - val_loss: 4.9496 - val_accuracy: 0.4316
Epoch 3/50
89/89 [==============================] - 5s 51ms/step - loss: 5.0362 - accuracy: 0.4231 - val_loss: 4.7615 - val_accuracy: 0.4370
Epoch 4/50
89/89 [==============================] - 5s 51ms/step - loss: 4.8063 - accuracy: 0.4281 - val_loss: 4.5179 - val_accuracy: 0.4447
Epoch 5/50
89/89 [==============================] - 5s 51ms/step - loss: 4.5459 - accuracy: 0.4360 - val_loss: 4.3064 - val_accuracy: 0.4520
Epoch 6/50
89/89 [==============================] - 5s 52ms/step - loss: 4.3302 - accuracy: 0.4434 - val_loss: 4.1767 - val_accuracy: 0.4577
Epoch 7/50
89/89 [==============================] - 5s 52ms/step - loss: 4.1516 - accuracy: 0.4506 - val_loss: 4.0780 - val_accuracy: 0.4600
Epoch 8/50
89/89 [==============================] - 5s 52ms/step - loss: 4.0120 - accuracy: 0.4563 - val_loss: 3.9575 - val_accuracy: 0.4672
Epoch 9/50
89/89 [==============================] - 5s 51ms/step - loss: 3.8846 - accuracy: 0.4619 - val_loss: 3.8407 - val_accuracy: 0.4751
Epoch 10/50
89/89 [==============================] - 5s 51ms/step - loss: 3.7604 - accuracy: 0.4691 - val_loss: 3.7645 - val_accuracy: 0.4804
Epoch 11/50
89/89 [==============================] - 5s 51ms/step - loss: 3.6537 - accuracy: 0.4750 - val_loss: 3.6950 - val_accuracy: 0.4848
Epoch 12/50
89/89 [==============================] - 5s 51ms/step - loss: 3.5539 - accuracy: 0.4805 - val_loss: 3.6272 - val_accuracy: 0.4907
Epoch 13/50
89/89 [==============================] - 5s 51ms/step - loss: 3.4608 - accuracy: 0.4860 - val_loss: 3.5658 - val_accuracy: 0.4951
Epoch 14/50
89/89 [==============================] - 5s 51ms/step - loss: 3.3766 - accuracy: 0.4911 - val_loss: 3.5267 - val_accuracy: 0.4962
Epoch 15/50
89/89 [==============================] - 5s 51ms/step - loss: 3.2965 - accuracy: 0.4971 - val_loss: 3.4929 - val_accuracy: 0.4993
Epoch 16/50
89/89 [==============================] - 5s 51ms/step - loss: 3.2235 - accuracy: 0.5017 - val_loss: 3.4524 - val_accuracy: 0.5005
Epoch 17/50
89/89 [==============================] - 5s 51ms/step - loss: 3.1538 - accuracy: 0.5062 - val_loss: 3.4047 - val_accuracy: 0.5061
Epoch 18/50
89/89 [==============================] - 5s 51ms/step - loss: 3.0803 - accuracy: 0.5115 - val_loss: 3.3713 - val_accuracy: 0.5091
```

```
Epoch 19/50
89/89 [==============================] - 5s 58ms/step - loss: 3.0169 - accuracy: 0.5162 - val_loss: 3.3442 - val_accuracy: 0.5106
Epoch 20/50
89/89 [==============================] - 5s 53ms/step - loss: 2.9631 - accuracy: 0.5200 - val_loss: 3.3358 - val_accuracy: 0.5104
Epoch 21/50
89/89 [==============================] - 5s 51ms/step - loss: 2.9033 - accuracy: 0.5249 - val_loss: 3.2948 - val_accuracy: 0.5148
Epoch 22/50
89/89 [==============================] - 5s 51ms/step - loss: 2.8580 - accuracy: 0.5285 - val_loss: 3.2748 - val_accuracy: 0.5168
Epoch 23/50
89/89 [==============================] - 5s 51ms/step - loss: 2.8079 - accuracy: 0.5319 - val_loss: 3.2622 - val_accuracy: 0.5166
Epoch 24/50
89/89 [==============================] - 5s 52ms/step - loss: 2.7563 - accuracy: 0.5369 - val_loss: 3.2345 - val_accuracy: 0.5200
Epoch 25/50
89/89 [==============================] - 5s 52ms/step - loss: 2.7087 - accuracy: 0.5405 - val_loss: 3.2245 - val_accuracy: 0.5198
Epoch 26/50
89/89 [==============================] - 5s 51ms/step - loss: 2.6670 - accuracy: 0.5451 - val_loss: 3.1917 - val_accuracy: 0.5254
Epoch 27/50
89/89 [==============================] - 5s 51ms/step - loss: 2.6220 - accuracy: 0.5481 - val_loss: 3.2013 - val_accuracy: 0.5215
Epoch 28/50
89/89 [==============================] - 5s 51ms/step - loss: 2.5846 - accuracy: 0.5517 - val_loss: 3.1739 - val_accuracy: 0.5273
Epoch 29/50
```

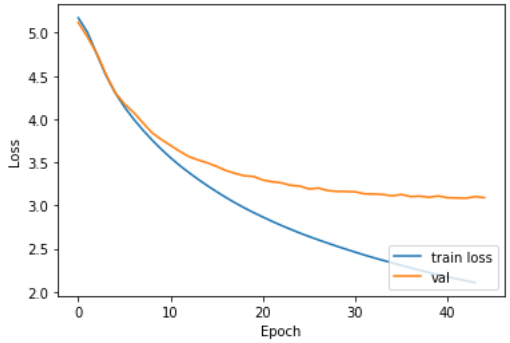
```
# Accuracy
plt.plot(history.history['accuracy'][1:], label='train acc')
plt.plot(history.history['val_accuracy'], label='val')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
```

```
<matplotlib.legend.Legend at 0x7f7647f0e610>
```




```
# Loss
plt.plot(history.history['loss'][1:], label='train loss')
plt.plot(history.history['val_loss'], label='val')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='lower right')
```

```
<matplotlib.legend.Legend at 0x7f7646257690>
```




```
# Inference
encoder_model, decoder_model = inference_func(
    max_text_len, latent_dim, encoder_input, encoder_output,
    encoder_final_states, decoder_input, decoder_output,
    decoder_embedding_layer, decoder_dense, last_decoder_lstm
)
```

```
encoder_model.summary()
```

```
Model: "model_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_6 (InputLayer)         [(None, 42)]              0
_________________________________________________________________
embedding_2 (Embedding)      (None, 42, 300)           29932800
_________________________________________________________________
lstm_3 (LSTM)                [(None, 42, 240), (None,  519360
```

```
lstm_4 (LSTM)                [(None, 42, 240), (None,  461760
==================================================================
Total params: 30,913,920
Trainable params: 981,120
Non-trainable params: 29,932,800
__________________________________________________________________
```

```
decoder_model.summary()
```

```
Model: "model_5"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_7 (InputLayer)            [(None, None)]       0
__________________________________________________________________________________________________
embedding_3 (Embedding)         (None, None, 300)    11257200    input_7[0][0]
__________________________________________________________________________________________________
input_8 (InputLayer)            [(None, 240)]        0
__________________________________________________________________________________________________
input_9 (InputLayer)            [(None, 240)]        0
__________________________________________________________________________________________________
lstm_5 (LSTM)                   [(None, None, 240),  519360      embedding_3[1][0]
                                                                 input_8[0][0]
                                                                 input_9[0][0]
__________________________________________________________________________________________________
input_10 (InputLayer)           [(None, 42, 240)]    0
__________________________________________________________________________________________________
time_distributed_1 (TimeDistrib (None, None, 37524)  9043284     lstm_5[1][0]
==================================================================================================
Total params: 20,819,844
Trainable params: 20,819,844
Non-trainable params: 0
__________________________________________________________________________________________________
```

```
# Testing on training data
for i in range(0, 10):
    print(f"# {i+1} News: ", seq2text(x_train_padded[i]))
    print("Original summary: ", seq2summary(y_train_padded[i]))
    print(
        "Predicted summary: ",
        decode_sequence_func(
            x_train_padded[i].reshape(1, max_text_len), encoder_model,
            decoder_model
        )
    )
    print()
```

```
# 1 News:  spanish fashion retailer zara withdrawn skirt website facing criticism featuring pepe frog internet mtheme turned symbol white national
Original summary:  start zara withdraws skirt featuring pepe frog altright mtheme end
Predicted summary:   start zara launches ad featuring pepe frog altright dress end

# 2 News:  elections fill vacant seats rajya sabha held today mthembers elected unopposed one seats facing byelections kerala mp resigned mber mth
Original summary:  start explained details rajya sabha election held today end
Predicted summary:   start rajya sabha seats rajya sabha seats end

# 3 News:  first posthumous nobel prize awarded swedens erik karlfeldt literature posthumous nobel peace prize given youngest united nations secre
Original summary:  start rare cases nobel prize given death end
Predicted summary:   start nobel prize awarded nobel prize stockholm end

# 4 News:  titanic actors leonardo dico kate winslet auctioning dinner two ththem leonardo dico foundations fourth annual gala wednesday charity d
Original summary:  start titanic stars leonardo kate auction dinner date charity end
Predicted summary:   start titanic stars leonardo kate dinner date auction end

# 5 News:  zaheer khans fiancaae actor sagarika ghatge posted picture instagram couple zaheer flaunts new cleanshaven look sagarika captioned pict
Original summary:  start zaheer broke beard rather well says fiancaae sagarika end
Predicted summary:   start zaheer sagarika ghatge shares pic wife ritika end

# 6 News:  killer whale stopped carrying dead newborn calf least days covered km according researchers killer whales known carry dead calves week
Original summary:  start killer whale abandons dead newborn calf days end
Predicted summary:   start dead whale found carrying plastic beach end

# 7 News:  letter election commission madhya pradesh congress committee alleged pm narendra modi violated model code conduct speech pollbound stat
Original summary:  start pm modi violated poll code mp congress letter ec end
Predicted summary:   start pm modi violated poll code code cong mp end

# 8 News:  malls hotels restaurants bengaluru directed provide free clean drinking water customers citys civic body bruhat bengaluru mahanagara pa
Original summary:  start bengaluru malls restaurants provide free drinking water end
Predicted summary:   start bengaluru restaurants restaurants malls malls malls malls end

# 9 News:  reserve bank new zealand acting governor grant spencer said bitcoin unstable useful future adding cryptocurrency looks rthemarkably lik
Original summary:  start bitcoin unstable useful nz central bank chief end
Predicted summary:   start bitcoin price old notes says rbi end

# 10 News:  indias batting coach sanjay bangar said void team coach anil kumbles resignation team coping well professionals things part parcel org
Original summary:  start void kumbles exit india batting coach end
Predicted summary:   start kumble coach coach selection kumbles exit end
```

**Model with Bidirectional LSTMs**

```
model_func = models_info['bidirectional_lstm']['model']
inference_func = models_info['bidirectional_lstm']['inference']
decode_sequence_func = models_info['bidirectional_lstm']['decode_sequence']
```

```
seq2seq = model_func(
    embedding_dim, latent_dim, max_text_len,
    x_vocab_size, y_vocab_size,
    x_embedding_matrix, y_embedding_matrix
)

model = seq2seq['model']

encoder_input = seq2seq['inputs']['encoder']
decoder_input = seq2seq['inputs']['decoder']

encoder_output = seq2seq['outputs']['encoder']
decoder_output = seq2seq['outputs']['decoder']

encoder_final_states = seq2seq['states']['encoder']
decoder_final_states = seq2seq['states']['decoder']

decoder_embedding_layer = seq2seq['layers']['decoder']['embedding']
last_decoder_lstm = seq2seq['layers']['decoder']['last_decoder_lstm']
decoder_dense = seq2seq['layers']['decoder']['dense']

model.summary()
```

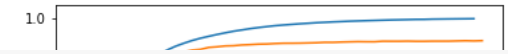
```
Model: "seq2seq_model_with_bidirectional_lstm"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_11 (InputLayer)           [(None, 42)]         0
__________________________________________________________________________________________________
encoder_embedding (Embedding)   (None, 42, 300)      29932800    input_11[0][0]
__________________________________________________________________________________________________
encoder_bidirectional_lstm_1 (B [(None, 42, 480), (N 1038720     encoder_embedding[0][0]
__________________________________________________________________________________________________
input_12 (InputLayer)           [(None, None)]       0
__________________________________________________________________________________________________
encoder_bidirectional_lstm_2 (B [(None, 42, 480), (N 1384320     encoder_bidirectional_lstm_1[0][0
__________________________________________________________________________________________________
decoder_embedding (Embedding)   (None, None, 300)    11257200    input_12[0][0]
__________________________________________________________________________________________________
encoder_bidirectional_lstm_3 (B [(None, 42, 480), (N 1384320     encoder_bidirectional_lstm_2[0][0
__________________________________________________________________________________________________
decoder_bidirectional_lstm_1 (B [(None, None, 480),  1038720     decoder_embedding[0][0]
                                                                 encoder_bidirectional_lstm_3[0][1
                                                                 encoder_bidirectional_lstm_3[0][2
                                                                 encoder_bidirectional_lstm_3[0][3
                                                                 encoder_bidirectional_lstm_3[0][4
__________________________________________________________________________________________________
time_distributed_2 (TimeDistrib (None, None, 37524)  18049044    decoder_bidirectional_lstm_1[0][0
==================================================================================================
Total params: 64,085,124
Trainable params: 22,895,124
Non-trainable params: 41,190,000
__________________________________________________________________________________________________
Model: "seq2seq_model_with_bidirectional_lstm"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_11 (InputLayer)           [(None, 42)]         0
__________________________________________________________________________________________________
encoder_embedding (Embedding)   (None, 42, 300)      29932800    input_11[0][0]
__________________________________________________________________________________________________
encoder_bidirectional_lstm_1 (B [(None, 42, 480), (N 1038720     encoder_embedding[0][0]
__________________________________________________________________________________________________
input_12 (InputLayer)           [(None, None)]       0
__________________________________________________________________________________________________
encoder_bidirectional_lstm_2 (B [(None, 42, 480), (N 1384320     encoder_bidirectional_lstm_1[0][0
__________________________________________________________________________________________________
decoder_embedding (Embedding)   (None, None, 300)    11257200    input_12[0][0]
__________________________________________________________________________________________________
encoder_bidirectional_lstm_3 (B [(None, 42, 480), (N 1384320     encoder_bidirectional_lstm_2[0][0
__________________________________________________________________________________________________
decoder_bidirectional_lstm_1 (B [(None, None, 480),  1038720     decoder_embedding[0][0]
                                                                 encoder_bidirectional_lstm_3[0][1
                                                                 encoder_bidirectional_lstm_3[0][2
                                                                 encoder_bidirectional_lstm_3[0][3
                                                                 encoder_bidirectional_lstm_3[0][4
__________________________________________________________________________________________________
time_distributed_2 (TimeDistrib (None, None, 37524)  18049044    decoder_bidirectional_lstm_1[0][0
==================================================================================================
Total params: 64,085,124
Trainable params: 22,895,124
```

```
history = model.fit(
    [x_train_padded, y_train_padded[:, :-1]],
    y_train_padded.reshape(y_train_padded.shape[0], y_train_padded.shape[1], 1)[:, 1:],
    epochs=num_epochs,
    batch_size=128 * tpu_strategy.num_replicas_in_sync,
    callbacks=callbacks,
    validation_data=(
        [x_val_padded, y_val_padded[:, :-1]],
        y_val_padded.reshape(y_val_padded.shape[0], y_val_padded.shape[1], 1)[:, 1:]
    )
)
```

```
Epoch 1/50
89/89 [==============================] - 61s 334ms/step - loss: 6.4102 - accuracy: 0.3250 - val_loss: 4.3666 - val_accuracy: 0.4567
Epoch 2/50
89/89 [==============================] - 7s 83ms/step - loss: 4.2414 - accuracy: 0.4610 - val_loss: 3.2875 - val_accuracy: 0.5459
Epoch 3/50
89/89 [==============================] - 7s 82ms/step - loss: 3.2620 - accuracy: 0.5488 - val_loss: 2.4934 - val_accuracy: 0.6362
Epoch 4/50
89/89 [==============================] - 7s 82ms/step - loss: 2.5241 - accuracy: 0.6396 - val_loss: 1.9252 - val_accuracy: 0.7176
Epoch 5/50
89/89 [==============================] - 7s 83ms/step - loss: 1.9854 - accuracy: 0.7159 - val_loss: 1.5201 - val_accuracy: 0.7789
Epoch 6/50
89/89 [==============================] - 7s 84ms/step - loss: 1.5887 - accuracy: 0.7749 - val_loss: 1.2316 - val_accuracy: 0.8263
Epoch 7/50
89/89 [==============================] - 7s 83ms/step - loss: 1.2919 - accuracy: 0.8190 - val_loss: 1.0260 - val_accuracy: 0.8556
Epoch 8/50
89/89 [==============================] - 7s 82ms/step - loss: 1.0624 - accuracy: 0.8531 - val_loss: 0.8688 - val_accuracy: 0.8783
Epoch 9/50
89/89 [==============================] - 7s 83ms/step - loss: 0.8869 - accuracy: 0.8784 - val_loss: 0.7577 - val_accuracy: 0.8945
Epoch 10/50
89/89 [==============================] - 7s 84ms/step - loss: 0.7514 - accuracy: 0.8976 - val_loss: 0.6759 - val_accuracy: 0.9049
Epoch 11/50
89/89 [==============================] - 7s 83ms/step - loss: 0.6417 - accuracy: 0.9126 - val_loss: 0.6091 - val_accuracy: 0.9133
Epoch 12/50
89/89 [==============================] - 8s 87ms/step - loss: 0.5520 - accuracy: 0.9246 - val_loss: 0.5561 - val_accuracy: 0.9202
Epoch 13/50
89/89 [==============================] - 7s 83ms/step - loss: 0.4770 - accuracy: 0.9349 - val_loss: 0.5170 - val_accuracy: 0.9248
Epoch 14/50
89/89 [==============================] - 7s 82ms/step - loss: 0.4198 - accuracy: 0.9424 - val_loss: 0.4810 - val_accuracy: 0.9296
Epoch 15/50
89/89 [==============================] - 7s 83ms/step - loss: 0.3682 - accuracy: 0.9490 - val_loss: 0.4515 - val_accuracy: 0.9335
Epoch 16/50
89/89 [==============================] - 7s 83ms/step - loss: 0.3267 - accuracy: 0.9544 - val_loss: 0.4341 - val_accuracy: 0.9350
Epoch 17/50
89/89 [==============================] - 7s 83ms/step - loss: 0.2889 - accuracy: 0.9595 - val_loss: 0.4103 - val_accuracy: 0.9391
Epoch 18/50
89/89 [==============================] - 7s 83ms/step - loss: 0.2577 - accuracy: 0.9638 - val_loss: 0.3956 - val_accuracy: 0.9403
Epoch 19/50
89/89 [==============================] - 7s 83ms/step - loss: 0.2310 - accuracy: 0.9677 - val_loss: 0.3823 - val_accuracy: 0.9421
Epoch 20/50
89/89 [==============================] - 7s 82ms/step - loss: 0.2090 - accuracy: 0.9709 - val_loss: 0.3722 - val_accuracy: 0.9426
Epoch 21/50
89/89 [==============================] - 7s 83ms/step - loss: 0.1879 - accuracy: 0.9744 - val_loss: 0.3620 - val_accuracy: 0.9441
Epoch 22/50
89/89 [==============================] - 7s 83ms/step - loss: 0.1712 - accuracy: 0.9768 - val_loss: 0.3536 - val_accuracy: 0.9448
Epoch 23/50
89/89 [==============================] - 7s 83ms/step - loss: 0.1555 - accuracy: 0.9794 - val_loss: 0.3460 - val_accuracy: 0.9458
Epoch 24/50
89/89 [==============================] - 7s 83ms/step - loss: 0.1416 - accuracy: 0.9815 - val_loss: 0.3417 - val_accuracy: 0.9462
Epoch 25/50
89/89 [==============================] - 7s 83ms/step - loss: 0.1295 - accuracy: 0.9833 - val_loss: 0.3338 - val_accuracy: 0.9471
Epoch 26/50
89/89 [==============================] - 7s 83ms/step - loss: 0.1177 - accuracy: 0.9851 - val_loss: 0.3289 - val_accuracy: 0.9477
Epoch 27/50
89/89 [==============================] - 7s 83ms/step - loss: 0.1081 - accuracy: 0.9864 - val_loss: 0.3244 - val_accuracy: 0.9477
Epoch 28/50
89/89 [==============================] - 7s 83ms/step - loss: 0.1000 - accuracy: 0.9873 - val_loss: 0.3233 - val_accuracy: 0.9479
Epoch 29/50
89/89 [==============================] - 7s 83ms/step - loss: 0.0915 - accuracy: 0.9886 - val_loss: 0.3203 - val_accuracy: 0.9483
```

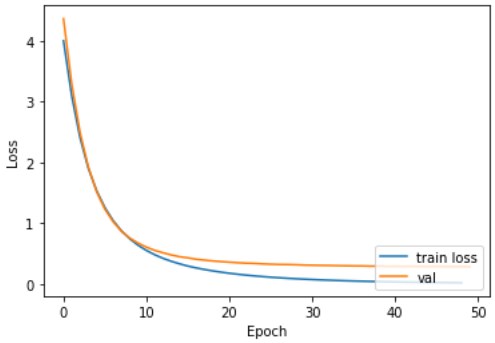
```
# Accuracy
plt.plot(history.history['accuracy'][1:], label='train acc')
plt.plot(history.history['val_accuracy'], label='val')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
```

```
<matplotlib.legend.Legend at 0x7f764510aed0>
```




```
# Loss
plt.plot(history.history['loss'][1:], label='train loss')
plt.plot(history.history['val_loss'], label='val')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='lower right')
```

```
<matplotlib.legend.Legend at 0x7f76457f3c10>
```




```
# Inference
encoder_model, decoder_model = inference_func(
    max_text_len, latent_dim, encoder_input, encoder_output,
    encoder_final_states, decoder_input, decoder_output,
    decoder_embedding_layer, decoder_dense, last_decoder_lstm
)
```

```
encoder_model.summary()
```

```
Model: "model_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_11 (InputLayer)        [(None, 42)]              0
_________________________________________________________________
encoder_embedding (Embedding (None, 42, 300)           29932800
_________________________________________________________________
encoder_bidirectional_lstm_1 [(None, 42, 480), (None,  1038720
_________________________________________________________________
encoder_bidirectional_lstm_2 [(None, 42, 480), (None,  1384320
_________________________________________________________________
encoder_bidirectional_lstm_3 [(None, 42, 480), (None,  1384320
=================================================================
Total params: 33,740,160
Trainable params: 3,807,360
Non-trainable params: 29,932,800
_________________________________________________________________
```

```
decoder_model.summary()
```

```
Model: "model_7"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_12 (InputLayer)           [(None, None)]       0
__________________________________________________________________________________________________
decoder_embedding (Embedding)   (None, None, 300)    11257200    input_12[0][0]
__________________________________________________________________________________________________
input_13 (InputLayer)           [(None, 240)]        0
__________________________________________________________________________________________________
input_14 (InputLayer)           [(None, 240)]        0
__________________________________________________________________________________________________
input_15 (InputLayer)           [(None, 240)]        0
__________________________________________________________________________________________________
input_16 (InputLayer)           [(None, 240)]        0
__________________________________________________________________________________________________
decoder_bidirectional_lstm_1 (B [(None, None, 480),  1038720     decoder_embedding[1][0]
                                                                 input_13[0][0]
                                                                 input_14[0][0]
                                                                 input_15[0][0]
                                                                 input_16[0][0]
__________________________________________________________________________________________________
input_17 (InputLayer)           [(None, 42, 480)]    0
__________________________________________________________________________________________________
time_distributed_2 (TimeDistrib (None, None, 37524)  18049044    decoder_bidirectional_lstm_1[1][0
==================================================================================================
Total params: 30,344,964
Trainable params: 19,087,764
Non-trainable params: 11,257,200
__________________________________________________________________________________________________
```

```
# Testing on training data
for i in range(0, 10):
    print(f"# {i+1} News: ", seq2text(x_train_padded[i]))
    print("Original summary: ", seq2summary(y_train_padded[i]))
    print(
        "Predicted summary: ",
        decode_sequence_func(
            x_train_padded[i].reshape(1, max_text_len), encoder_model,
            decoder_model
        )
    )
    print()
```

```
 # 1 News:  spanish fashion retailer zara withdrawn skirt website facing criticism featuring pepe frog internet mtheme turned symbol white national
 Original summary:  start zara withdraws skirt featuring pepe frog altright mtheme end
 Predicted summary:   end end end end end end end end end end end

 # 2 News:  elections fill vacant seats rajya sabha held today mthembers elected unopposed one seats facing byelections kerala mp resigned mber mth
 Original summary:  start explained details rajya sabha election held today end
 Predicted summary:   end end end end end end end end end end end

 # 3 News:  first posthumous nobel prize awarded swedens erik karlfeldt literature posthumous nobel peace prize given youngest united nations secre
 Original summary:  start rare cases nobel prize given death end
 Predicted summary:   end end end end end end end end end end end

 # 4 News:  titanic actors leonardo dico kate winslet auctioning dinner two ththem leonardo dico foundations fourth annual gala wednesday charity d
 Original summary:  start titanic stars leonardo kate auction dinner date charity end
 Predicted summary:   end end end end end end end end end end end

 # 5 News:  zaheer khans fiancaae actor sagarika ghatge posted picture instagram couple zaheer flaunts new cleanshaven look sagarika captioned pict
 Original summary:  start zaheer broke beard rather well says fiancaae sagarika end
 Predicted summary:   end end end end end end end end end end end

 # 6 News:  killer whale stopped carrying dead newborn calf least days covered km according researchers killer whales known carry dead calves week
 Original summary:  start killer whale abandons dead newborn calf days end
 Predicted summary:   end end end end end end end end end end end

 # 7 News:  letter election commission madhya pradesh congress committee alleged pm narendra modi violated model code conduct speech pollbound stat
 Original summary:  start pm modi violated poll code mp congress letter ec end
 Predicted summary:   end end end end end end end end end end end

 # 8 News:  malls hotels restaurants bengaluru directed provide free clean drinking water customers citys civic body bruhat bengaluru mahanagara pa
 Original summary:  start bengaluru malls restaurants provide free drinking water end
 Predicted summary:   end end end end end end end end end end end

 # 9 News:  reserve bank new zealand acting governor grant spencer said bitcoin unstable useful future adding cryptocurrency looks rthemarkably lik
 Original summary:  start bitcoin unstable useful nz central bank chief end
 Predicted summary:   end end end end end end end end end end end

 # 10 News:  indias batting coach sanjay bangar said void team coach anil kumbles resignation team coping well professionals things part parcel org
 Original summary:  start void kumbles exit india batting coach end
 Predicted summary:   end end end end end end end end end end end
```

## Model with hybrid architecture

```
model_func = models_info['hybrid_model']['model']
inference_func = models_info['hybrid_model']['inference']
decode_sequence_func = models_info['hybrid_model']['decode_sequence']
```

```
seq2seq = model_func(
    embedding_dim, latent_dim, max_text_len,
    x_vocab_size, y_vocab_size,
    x_embedding_matrix, y_embedding_matrix
)

model = seq2seq['model']

encoder_input = seq2seq['inputs']['encoder']
decoder_input = seq2seq['inputs']['decoder']

encoder_output = seq2seq['outputs']['encoder']
decoder_output = seq2seq['outputs']['decoder']

encoder_final_states = seq2seq['states']['encoder']
decoder_final_states = seq2seq['states']['decoder']

decoder_embedding_layer = seq2seq['layers']['decoder']['embedding']
last_decoder_lstm = seq2seq['layers']['decoder']['last_decoder_lstm']
decoder_dense = seq2seq['layers']['decoder']['dense']

model.summary()
```

```
Model: "seq2seq_model_with_bidirectional_lstm"
__________________________________________________________________________________________________
```

```
Layer (type)                 Output Shape              Param #   Connected to
==================================================================================================
input_18 (InputLayer)        [(None, 42)]              0
__________________________________________________________________________________________________
encoder_embedding (Embedding)   (None, 42, 300)        29932800  input_18[0][0]
__________________________________________________________________________________________________
encoder_bidirectional_lstm_1 (B [(None, 42, 480), (N 1038720   encoder_embedding[0][0]
__________________________________________________________________________________________________
input_19 (InputLayer)        [(None, None)]            0
__________________________________________________________________________________________________
encoder_bidirectional_lstm_2 (B [(None, 42, 480), (N 1384320   encoder_bidirectional_lstm_1[0][0
__________________________________________________________________________________________________
decoder_embedding (Embedding)   (None, None, 300)      11257200  input_19[0][0]
__________________________________________________________________________________________________
encoder_bidirectional_lstm_3 (B [(None, 42, 480), (N 1384320   encoder_bidirectional_lstm_2[0][0
__________________________________________________________________________________________________
decoder_lstm_1 (LSTM)        [(None, None, 240),  519360    decoder_embedding[0][0]
                                                            encoder_bidirectional_lstm_3[0][1
                                                            encoder_bidirectional_lstm_3[0][2
__________________________________________________________________________________________________
time_distributed_3 (TimeDistrib (None, None, 37524)  9043284   decoder_lstm_1[0][0]
==================================================================================================
Total params: 54,560,004
Trainable params: 13,370,004
Non-trainable params: 41,190,000
__________________________________________________________________________________________________
Model: "seq2seq_model_with_bidirectional_lstm"
__________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #   Connected to
==================================================================================================
input_18 (InputLayer)        [(None, 42)]              0
__________________________________________________________________________________________________
encoder_embedding (Embedding)   (None, 42, 300)        29932800  input_18[0][0]
__________________________________________________________________________________________________
encoder_bidirectional_lstm_1 (B [(None, 42, 480), (N 1038720   encoder_embedding[0][0]
__________________________________________________________________________________________________
input_19 (InputLayer)        [(None, None)]            0
__________________________________________________________________________________________________
encoder_bidirectional_lstm_2 (B [(None, 42, 480), (N 1384320   encoder_bidirectional_lstm_1[0][0
__________________________________________________________________________________________________
decoder_embedding (Embedding)   (None, None, 300)      11257200  input_19[0][0]
__________________________________________________________________________________________________
encoder_bidirectional_lstm_3 (B [(None, 42, 480), (N 1384320   encoder_bidirectional_lstm_2[0][0
__________________________________________________________________________________________________
decoder_lstm_1 (LSTM)        [(None, None, 240),  519360    decoder_embedding[0][0]
                                                            encoder_bidirectional_lstm_3[0][1
                                                            encoder_bidirectional_lstm_3[0][2
__________________________________________________________________________________________________
time_distributed_3 (TimeDistrib (None, None, 37524)  9043284   decoder_lstm_1[0][0]
==================================================================================================
Total params: 54,560,004
Trainable params: 13,370,004
Non-trainable params: 41,190,000
__________________________________________________________________________________________________
```

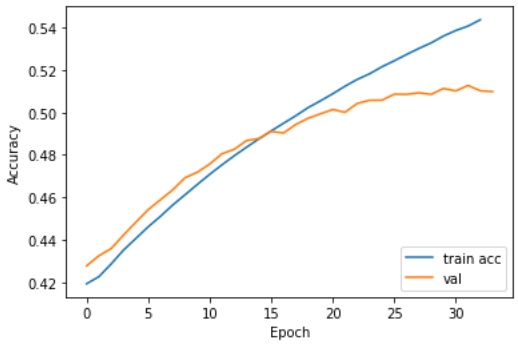
```
history = model.fit(
    [x_train_padded, y_train_padded[:, :-1]],
    y_train_padded.reshape(y_train_padded.shape[0], y_train_padded.shape[1], 1)[:, 1:],
    epochs=num_epochs,
    batch_size=128 * tpu_strategy.num_replicas_in_sync,
    callbacks=callbacks,
    validation_data=(
        [x_val_padded, y_val_padded[:, :-1]],
        y_val_padded.reshape(y_val_padded.shape[0], y_val_padded.shape[1], 1)[:, 1:]
    )
)
```

```
Epoch 1/50
89/89 [==============================] - 49s 270ms/step - loss: 7.0281 - accuracy: 0.2449 - val_loss: 5.1333 - val_accuracy: 0.4277
Epoch 2/50
89/89 [==============================] - 6s 71ms/step - loss: 5.2092 - accuracy: 0.4184 - val_loss: 4.9388 - val_accuracy: 0.4325
Epoch 3/50
89/89 [==============================] - 6s 71ms/step - loss: 5.0413 - accuracy: 0.4215 - val_loss: 4.7435 - val_accuracy: 0.4359
Epoch 4/50
89/89 [==============================] - 6s 72ms/step - loss: 4.7893 - accuracy: 0.4276 - val_loss: 4.5190 - val_accuracy: 0.4422
Epoch 5/50
89/89 [==============================] - 6s 72ms/step - loss: 4.5488 - accuracy: 0.4336 - val_loss: 4.3426 - val_accuracy: 0.4483
Epoch 6/50
89/89 [==============================] - 6s 72ms/step - loss: 4.3701 - accuracy: 0.4389 - val_loss: 4.2001 - val_accuracy: 0.4542
Epoch 7/50
89/89 [==============================] - 6s 72ms/step - loss: 4.2124 - accuracy: 0.4449 - val_loss: 4.0844 - val_accuracy: 0.4589
Epoch 8/50
89/89 [==============================] - 6s 72ms/step - loss: 4.0762 - accuracy: 0.4501 - val_loss: 3.9960 - val_accuracy: 0.4635
Epoch 9/50
89/89 [==============================] - 6s 72ms/step - loss: 3.9537 - accuracy: 0.4553 - val_loss: 3.9038 - val_accuracy: 0.4692
Epoch 10/50
89/89 [==============================] - 6s 71ms/step - loss: 3.8444 - accuracy: 0.4606 - val_loss: 3.8370 - val_accuracy: 0.4718
Epoch 11/50
89/89 [==============================] - 6s 73ms/step - loss: 3.7421 - accuracy: 0.4653 - val_loss: 3.7691 - val_accuracy: 0.4757
Epoch 12/50
89/89 [==============================] - 6s 72ms/step - loss: 3.6493 - accuracy: 0.4699 - val_loss: 3.7009 - val_accuracy: 0.4806
```

```
Epoch 13/50
89/89 [==============================] - 6s 72ms/step - loss: 3.5633 - accuracy: 0.4743 - val_loss: 3.6704 - val_accuracy: 0.4826
Epoch 14/50
89/89 [==============================] - 6s 72ms/step - loss: 3.4848 - accuracy: 0.4787 - val_loss: 3.6112 - val_accuracy: 0.4868
Epoch 15/50
89/89 [==============================] - 6s 72ms/step - loss: 3.4073 - accuracy: 0.4837 - val_loss: 3.5741 - val_accuracy: 0.4877
Epoch 16/50
89/89 [==============================] - 6s 71ms/step - loss: 3.3347 - accuracy: 0.4877 - val_loss: 3.5347 - val_accuracy: 0.4911
Epoch 17/50
89/89 [==============================] - 6s 72ms/step - loss: 3.2722 - accuracy: 0.4909 - val_loss: 3.5182 - val_accuracy: 0.4903
Epoch 18/50
89/89 [==============================] - 6s 71ms/step - loss: 3.2119 - accuracy: 0.4952 - val_loss: 3.4849 - val_accuracy: 0.4943
Epoch 19/50
89/89 [==============================] - 6s 72ms/step - loss: 3.1513 - accuracy: 0.4991 - val_loss: 3.4489 - val_accuracy: 0.4973
Epoch 20/50
89/89 [==============================] - 6s 72ms/step - loss: 3.0967 - accuracy: 0.5024 - val_loss: 3.4254 - val_accuracy: 0.4994
Epoch 21/50
89/89 [==============================] - 7s 81ms/step - loss: 3.0532 - accuracy: 0.5054 - val_loss: 3.4063 - val_accuracy: 0.5014
Epoch 22/50
89/89 [==============================] - 6s 71ms/step - loss: 3.0057 - accuracy: 0.5081 - val_loss: 3.4027 - val_accuracy: 0.5001
Epoch 23/50
89/89 [==============================] - 6s 72ms/step - loss: 2.9516 - accuracy: 0.5124 - val_loss: 3.3704 - val_accuracy: 0.5043
Epoch 24/50
89/89 [==============================] - 6s 72ms/step - loss: 2.9063 - accuracy: 0.5159 - val_loss: 3.3526 - val_accuracy: 0.5058
Epoch 25/50
89/89 [==============================] - 6s 72ms/step - loss: 2.8613 - accuracy: 0.5192 - val_loss: 3.3427 - val_accuracy: 0.5058
Epoch 26/50
89/89 [==============================] - 6s 72ms/step - loss: 2.8218 - accuracy: 0.5225 - val_loss: 3.3200 - val_accuracy: 0.5086
Epoch 27/50
89/89 [==============================] - 6s 72ms/step - loss: 2.7825 - accuracy: 0.5257 - val_loss: 3.3137 - val_accuracy: 0.5085
Epoch 28/50
89/89 [==============================] - 6s 72ms/step - loss: 2.7486 - accuracy: 0.5277 - val_loss: 3.3045 - val_accuracy: 0.5093
Epoch 29/50
```

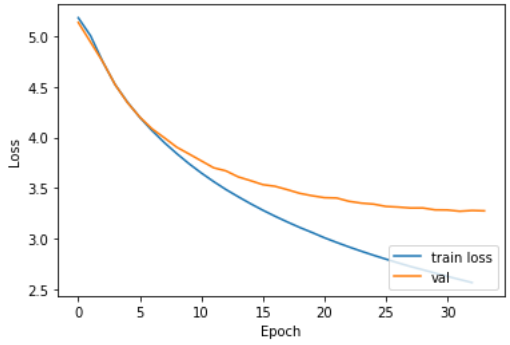
```
# Accuracy
plt.plot(history.history['accuracy'][1:], label='train acc')
plt.plot(history.history['val_accuracy'], label='val')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
```

```
<matplotlib.legend.Legend at 0x7f764427d7d0>
```




```
# Loss
plt.plot(history.history['loss'][1:], label='train loss')
plt.plot(history.history['val_loss'], label='val')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='lower right')
```

```
<matplotlib.legend.Legend at 0x7f7642ad9e90>
```




```
# Inference
encoder_model, decoder_model = inference_func(
    max_text_len, latent_dim, encoder_input, encoder_output,
    encoder_final_states, decoder_input, decoder_output,
    decoder_embedding_layer, decoder_dense, last_decoder_lstm
)
```

```
encoder_model.summary()
```

```
Model: "model_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_18 (InputLayer)        [(None, 42)]              0
_________________________________________________________________
encoder_embedding (Embedding (None, 42, 300)           29932800
_________________________________________________________________
encoder_bidirectional_lstm_1 [(None, 42, 480), (None,  1038720
_________________________________________________________________
encoder_bidirectional_lstm_2 [(None, 42, 480), (None,  1384320
_________________________________________________________________
encoder_bidirectional_lstm_3 [(None, 42, 480), (None,  1384320
=================================================================
Total params: 33,740,160
Trainable params: 3,807,360
Non-trainable params: 29,932,800
_________________________________________________________________
```

```
decoder_model.summary()
```

```
Model: "model_9"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_19 (InputLayer)           [(None, None)]       0
__________________________________________________________________________________________________
decoder_embedding (Embedding)   (None, None, 300)    11257200    input_19[0][0]
__________________________________________________________________________________________________
input_20 (InputLayer)           [(None, 240)]        0
__________________________________________________________________________________________________
input_21 (InputLayer)           [(None, 240)]        0
__________________________________________________________________________________________________
decoder_lstm_1 (LSTM)           [(None, None, 240),  519360      decoder_embedding[1][0]
                                                                 input_20[0][0]
                                                                 input_21[0][0]
__________________________________________________________________________________________________
input_22 (InputLayer)           [(None, 42, 480)]    0
__________________________________________________________________________________________________
time_distributed_3 (TimeDistrib (None, None, 37524)  9043284     decoder_lstm_1[1][0]
==================================================================================================
Total params: 20,819,844
Trainable params: 9,562,644
Non-trainable params: 11,257,200
__________________________________________________________________________________________________
```

```
# Testing on training data
```