Football Commentary Summarization

There 3 different training models used here

- `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.

To see the full learning and results of all the 3 model go to the end of the notebook in the `Running all the 3 different models` section

The `model (the trained model)`, `encoder_model (for inference)` and `decoder_model (for inference)` for **Seq2Seq with just LSTMs** are only saved.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
    Mounted at /content/drive
```

```
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, TimeDistrik
```

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

```
     |████████████████████████████████| 106 kB 5.5 MB/s
     |████████████████████████████████| 287 kB 41.4 MB/s
```

```
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)
```

```
INFO:tensorflow:Deallocate tpu buffers before initializing tpu system.
INFO:tensorflow:Deallocate tpu buffers before initializing tpu system.
INFO:tensorflow:Initializing the TPU system: grpc://10.32.173.194:8470
INFO:tensorflow:Initializing the TPU system: grpc://10.32.173.194:8470
INFO:tensorflow:Finished initializing TPU system.
INFO:tensorflow:Finished initializing TPU system.
WARNING:absl:`tf.distribute.experimental.TPUStrategy` is deprecated, please use
INFO:tensorflow:Found TPU system:
INFO:tensorflow:Found TPU system:
INFO:tensorflow:*** Num TPU Cores: 8
INFO:tensorflow:*** Num TPU Cores: 8
INFO:tensorflow:*** Num TPU Workers: 1
INFO:tensorflow:*** Num TPU Workers: 1
INFO:tensorflow:*** Num TPU Cores Per Worker: 8
INFO:tensorflow:*** Num TPU Cores Per Worker: 8
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localhost/replica:0
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localhost/replica:0
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/ta
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/ta
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/ta
```

```
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/ta
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/ta
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/ta
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/ta
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/ta
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/ta
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/ta
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/ta
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/ta
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/ta
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/ta
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/ta
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/ta
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/ta
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/ta
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/ta
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/ta
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/ta
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/ta
```

Getting the data

```
df = pd.DataFrame("", index=np.arange(31), columns=['text','headlines'])
```

```
def get_text(path):
  csv_file = pd.read_csv(path)
  csv_file.dropna(axis = 0,inplace =True)
  txt_op = " ".join(csv_file['text'])
  return txt_op
```

```
path_text = "/content/drive/MyDrive/raw_data"
path_summary = "/content/drive/MyDrive/reference_summaries"
```

```
for i in range(1,32):
  df['text'].iloc[i-1] = get_text(path_text+"/match_"+str(i)+"_comm.csv")
  if(i == 2): continue
  summary_file = open(path_summary+"/match_"+str(i)+"_report.txt","r")
  lines = summary_file.readlines()
  summary_file.close
  df['headlines'].iloc[i-1] = str(lines[1])[1:-1]
```

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.head(5)
```

	text	headlines
0	afternoon all! this match will be the ninth f...	atletico madrid have returned to the top four ...
1	good morning! it is one of the red-letter days...	
2	evening all! sports mole's live la liga covera...	villarreal missed the chance to beat barcelona...
3	hello and welcome sports mole's live text cove...	athletic bilbao will take a slender 2-1 lead i...
4	evening all! sports mole's live copa del rey c...	barcelona have booked their spot in the last-1...

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

text **headlines**

```
# 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 tha
```

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

```
# Remove stopwords from text\
import nltk
nltk.download("stopwords")
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")
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
'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)
```

	text	headline
3	hello welcome sports moles live text coverage ...	athletic bilbao take slender lead next weeks c.
17	evening sports moles live champions league cov...	lionel messi scored hattrick barcelona recorde.
9	evening sports moles live copa del rey coverag...	segunda side hercules held spanish champions b.
8	afternoon sports moles live la liga coverage c...	real madrid captain sergio ramos headed thminu.
19	morning sports moles live la liga coverage cam...	real madrid equalled spanish record games unbe.

```
# 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, **kw
    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).ger
```

```
    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="
        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)
```

0.0 -

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_')
```

WordCloud

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.

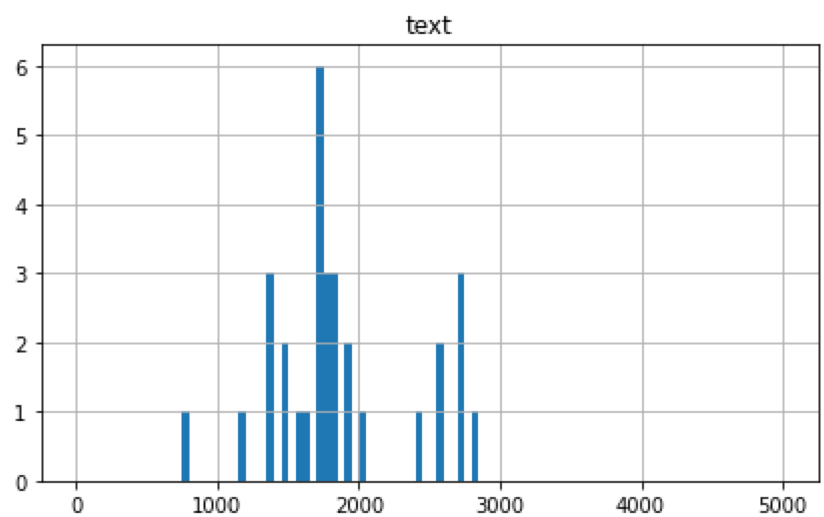
So we can just use these **sostok** & **eostok** instead of `_START_` & `_END_`, well you can but I tried both ways and while not using these `_START_` & `_END_` I was getting `undesired results` 🤯 😅 i.e. model's `results weren't good.`

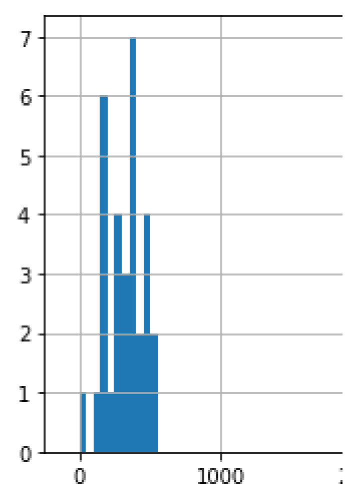
```
df.sample(5)
```

	text	headline
18	afternoon sports moles live la liga coverage c...	sostok _START_ barcelona risen atletico madric
12	hello welcome sports moles live text commentar...	sostok _START_ barcelona endured frustrating a
22	good afternoon fa cup third round well truly u...	sostok _START_ arsenal forced come behind book
6	morning sports moles live la liga coverage con...	sostok _START_ barcelona moved within three pc
3	hello welcome sports moles live text coverage ...	sostok _START_ athletic bilbao take slender le

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, figsiz
plt.show()
```






```
# 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, 500))

# Check how many % of summary have 0-42 words
print(get_word_percent(df.text, 2800))
```

```
0.94
0.97
```

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 = 2800
max_summary_len = 500


# 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: 28
```

	text	summary
23	good evening everyone thank joining us bring l...	sostok _START_ pep guardiola enjoyed winning s...
22	hello welcome sports moles live text coverage ...	sostok _START_ borussia dortmund claimed compr...
25	hello welcome sports moles live coverage	sostok _START_ goals ryan shawcross peter

```
# 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': 64.8, 'total_coverage': 10.74, 'count': 3197, 'total_count': 4934}
```

```
# 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)
```

```
4935
```

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': 84.43, 'total_coverage': 37.01, 'count': 1697, 'total_count': 2010}
```

```
# 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)
```

```
2011
```

```
# 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
```

```
!wget http://nlp.stanford.edu/data/glove.6B.zip
```

```
--2022-06-14 16:57:49--  http://nlp.stanford.edu/data/glove.6B.zip
Resolving nlp.stanford.edu (nlp.stanford.edu)... 171.64.67.140
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:80... connected
HTTP request sent, awaiting response... 302 Found
Location: https://nlp.stanford.edu/data/glove.6B.zip [following]
--2022-06-14 16:57:50--  https://nlp.stanford.edu/data/glove.6B.zip
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443... connecte
HTTP request sent, awaiting response... 301 Moved Permanently
Location: http://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
--2022-06-14 16:57:50--  http://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)... 171.64.64.22
Connecting to downloads.cs.stanford.edu (downloads.cs.stanford.edu)|171.64.64.22
HTTP request sent, awaiting response... 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: ‘glove.6B.zip’

glove.6B.zip        100%[===================>] 822.24M  5.05MB/s    in 2m 40s

2022-06-14 17:00:30 (5.13 MB/s) - ‘glove.6B.zip’ saved [862182613/862182613]
```

```
!unzip glove*.zip
```

```
Archive:  glove.6B.zip
  inflating: glove.6B.50d.txt
  inflating: glove.6B.100d.txt
  inflating: glove.6B.200d.txt
  inflating: glove.6B.300d.txt
```

```
!ls
!pwd
```

```
cleaned_data.csv   glove.6B.200d.txt  glove.6B.zip  y_tokenizer
drive              glove.6B.300d.txt  sample_data
glove.6B.100d.txt  glove.6B.50d.txt   x_tokenizer
/content
```

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

    path_to_glove_file = '/content/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 4221 words (713 misses)
Found 400000 word vectors.
Converted 1796 words (214 misses)
```

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

```
(4935, 300)
(2011, 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.

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
        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
        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_mc
    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
        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
        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_mo
    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, 2800)]        0           []

 embedding (Embedding)          (None, 2800, 300)     1480500     ['input_1[0][0]

 input_2 (InputLayer)           [(None, None)]        0           []

 lstm (LSTM)                    [(None, 2800, 240),   519360      ['embedding[0][
                                 (None, 240),
                                 (None, 240)]

 embedding_1 (Embedding)        (None, None, 300)     603300      ['input_2[0][0]

 lstm_1 (LSTM)                  [(None, 2800, 240),   461760      ['lstm[0][0]']
                                 (None, 240),
                                 (None, 240)]

 lstm_2 (LSTM)                  [(None, None, 240),   519360      ['embedding_1[0
                                 (None, 240),                      'lstm_1[0][1]'
                                 (None, 240)]                      'lstm_1[0][2]'

 time_distributed (TimeDistribu  (None, None, 2011)   484651      ['lstm_2[0][0]'
 ted)

========================================================================================
Total params: 4,068,931
Trainable params: 2,588,431
Non-trainable params: 1,480,500
________________________________________________________________________________________
```

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 'embeddin
 <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, ver
]
```

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:]
    )
)
```

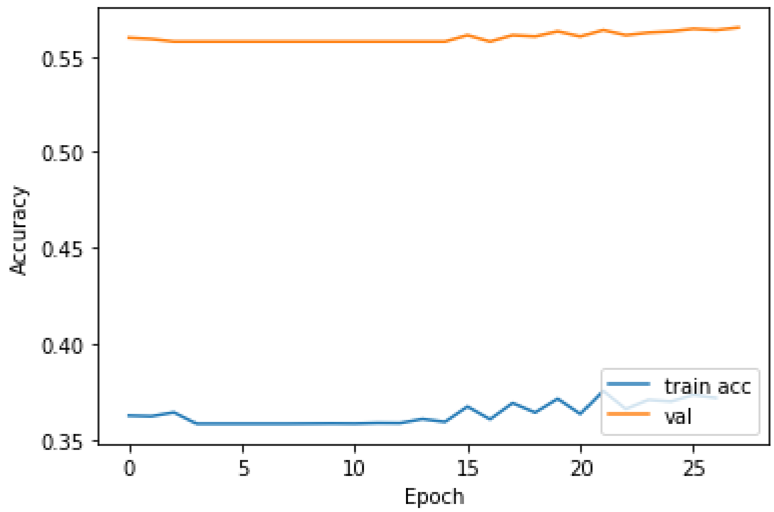
```
Epoch 2/50

1/1 [==============================] - 2s 2s/step - loss: 7.5366 - accuracy: 0.3
Epoch 3/50
```

```
1/1 [==============================] - 2s 2s/step - loss: 7.2534 - accuracy: 0.3
Epoch 4/50
1/1 [==============================] - 2s 2s/step - loss: 6.5081 - accuracy: 0.3
Epoch 5/50
1/1 [==============================] - 2s 2s/step - loss: 5.8058 - accuracy: 0.3
Epoch 6/50
1/1 [==============================] - 2s 2s/step - loss: 5.3695 - accuracy: 0.3
Epoch 7/50
1/1 [==============================] - 2s 2s/step - loss: 5.0629 - accuracy: 0.3
Epoch 8/50
1/1 [==============================] - 2s 2s/step - loss: 4.8576 - accuracy: 0.3
Epoch 9/50
1/1 [==============================] - 2s 2s/step - loss: 4.7955 - accuracy: 0.3
Epoch 10/50
1/1 [==============================] - 2s 2s/step - loss: 4.9481 - accuracy: 0.3
Epoch 11/50
1/1 [==============================] - 2s 2s/step - loss: 4.7344 - accuracy: 0.3
Epoch 12/50
1/1 [==============================] - 2s 2s/step - loss: 4.5800 - accuracy: 0.3
Epoch 13/50
1/1 [==============================] - 2s 2s/step - loss: 4.5143 - accuracy: 0.3
Epoch 14/50
1/1 [==============================] - 2s 2s/step - loss: 4.4598 - accuracy: 0.3
Epoch 15/50
1/1 [==============================] - 2s 2s/step - loss: 4.4355 - accuracy: 0.3
Epoch 16/50
1/1 [==============================] - 2s 2s/step - loss: 4.4023 - accuracy: 0.3
Epoch 17/50
1/1 [==============================] - 2s 2s/step - loss: 4.3912 - accuracy: 0.3
Epoch 18/50
1/1 [==============================] - 2s 2s/step - loss: 4.3689 - accuracy: 0.3
Epoch 19/50
1/1 [==============================] - 2s 2s/step - loss: 4.3546 - accuracy: 0.3
Epoch 20/50
1/1 [==============================] - 2s 2s/step - loss: 4.3244 - accuracy: 0.3
Epoch 21/50
1/1 [==============================] - 2s 2s/step - loss: 4.3111 - accuracy: 0.3
Epoch 22/50
1/1 [==============================] - 2s 2s/step - loss: 4.3038 - accuracy: 0.3
Epoch 23/50
1/1 [==============================] - 2s 2s/step - loss: 4.3148 - accuracy: 0.3
Epoch 24/50
1/1 [==============================] - 2s 2s/step - loss: 4.2659 - accuracy: 0.3
Epoch 25/50
1/1 [==============================] - 2s 2s/step - loss: 4.2445 - accuracy: 0.3
Epoch 26/50
1/1 [==============================] - 2s 2s/step - loss: 4.2268 - accuracy: 0.3
Epoch 27/50
1/1 [==============================] - 2s 2s/step - loss: 4.2094 - accuracy: 0.3
Epoch 28/50
1/1 [==============================] - ETA: 0s - loss: 4.1954 - accuracy: 0.3718
Epoch 28: ReduceLROnPlateau reducing learning rate to 0.00010000000474974513.
1/1 [==============================] - 2s 2s/step - loss: 4.1954 - accuracy: 0.3
Epoch 28: early stopping
```

Plotting model's performance

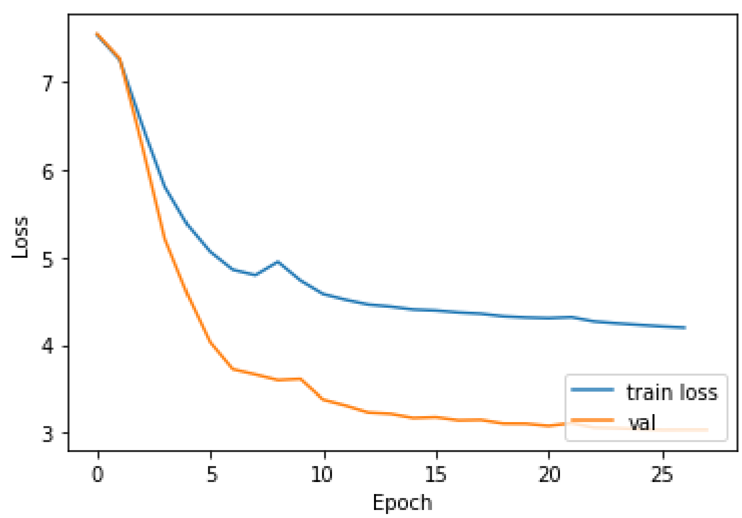
```
# Accuracy
plt.plot(history.history['accuracy'][1:], label='train acc')
plt.plot(history.history['val_accuracy'], label='val')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
```

```
<matplotlib.legend.Legend at 0x7fb180248fd0>
```




```
# 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 0x7fb17d5b8990>
```



✓ 0s completed at 12:08