Stony Brook University

# Final Project: Neural Machine Translation with Attention.

ESE 577: Deep Learning Algorithms and Software.

Spring 2024.

05/13/2024

Suvab Baral, 115646344

Sakshi Nagapure, 116000503

# Table of Contents

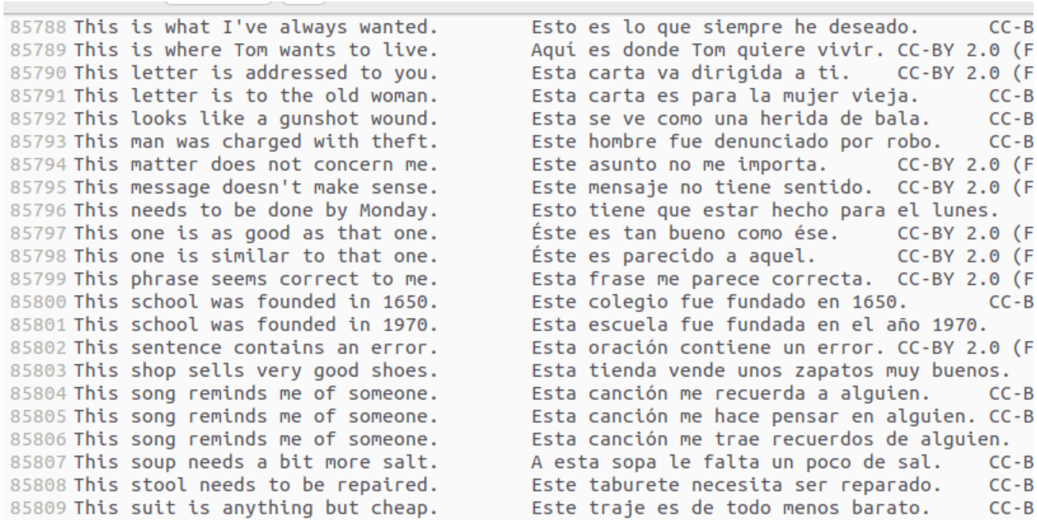
# 1. Introduction

Machine translation is an automated process of translating text from one language to another. This project explores a sequence-to-sequence encoder-decoder architecture with Long Short-Term Memory (LSTM) networks for translating sentences from Spanish to English. The project utilizes word embedding techniques to capture semantic relationships between words.

The report is divided into two major sections where in the first section we discuss preprocesses of words, sentences and ultimately token. We also discuss and show the word embedding we used and finally we discuss the model architecture. After training the model and analyzing the results, we post our findings in the results section of the report.

# 2. Description

The dataset we're working with is called "spa-eng"
"http://storage.googleapis.com/download.tensorflow.org/data/spa-eng.zip"
and it contains parallel text data in Spanish and English languages. Each entry in the dataset consists of a pair of sentences: one in Spanish and its corresponding translation in English. This dataset is commonly used for tasks such as machine translation, where the goal is to translate text from one language to another.



```
This is what I've always wanted.        Esto es lo que siempre he deseado.      CC-B
This is where Tom wants to live.        Aquí es donde Tom quiere vivir. CC-BY 2.0 (F
This letter is addressed to you.        Esta carta va dirigida a ti.    CC-BY 2.0 (F
This letter is to the old woman.        Esta carta es para la mujer vieja.      CC-B
This looks like a gunshot wound.        Esta se ve como una herida de bala.     CC-B
This man was charged with theft.        Este hombre fue denunciado por robo.    CC-B
This matter does not concern me.        Este asunto no me importa.      CC-BY 2.0 (F
This message doesn't make sense.        Este mensaje no tiene sentido.  CC-BY 2.0 (F
This needs to be done by Monday.        Esto tiene que estar hecho para el lunes.
This one is as good as that one.        Éste es tan bueno como ése.     CC-BY 2.0 (F
This one is similar to that one.        Éste es parecido a aquel.       CC-BY 2.0 (F
This phrase seems correct to me.        Esta frase me parece correcta.  CC-BY 2.0 (F
This school was founded in 1650.        Este colegio fue fundado en 1650.       CC-B
This school was founded in 1970.        Esta escuela fue fundada en el año 1970.
This sentence contains an error.        Esta oración contiene un error. CC-BY 2.0 (F
This shop sells very good shoes.        Esta tienda vende unos zapatos muy buenos.
This song reminds me of someone.        Esta canción me recuerda a alguien.     CC-B
This song reminds me of someone.        Esta canción me hace pensar en alguien. CC-B
This song reminds me of someone.        Esta canción me trae recuerdos de alguien.
This soup needs a bit more salt.        A esta sopa le falta un poco de sal.    CC-B
This stool needs to be repaired.        Este taburete necesita ser reparado.    CC-B
This suit is anything but cheap.        Este traje es de todo menos barato.     CC-B
```

In our project, we are utilizing this dataset to train and evaluate a neural machine translation model. The model learns from the parallel sentences in Spanish and English to understand the relationship between words and phrases in both languages, enabling it to generate accurate translations for new input sentences.
Using this dataset, our goal is to create a strong and reliable machine translation system that can accurately translate languages in real-world scenarios.

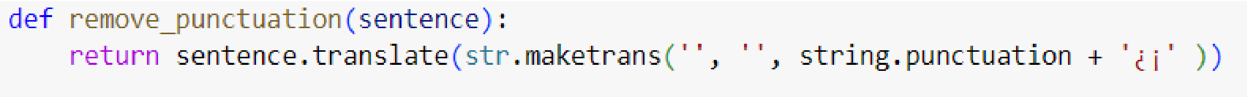
# 3. Algorithm and Implementation

## II. Data cleaning:

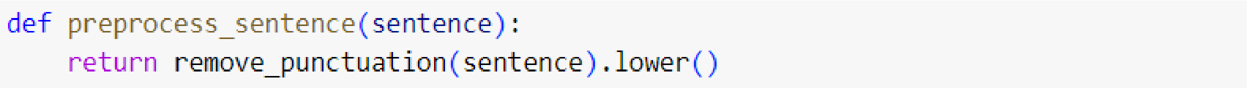
In this section, we discuss the data cleaning process carried out as part of the project implementation. The data cleaning phase is crucial as it prepares the raw input data for further analysis and processing.

We implemented two essential functions: `remove_punctuation` and `preprocess_sentence`.

The former function effectively eliminated all punctuation marks, including special characters such as '¿' and '¡', from the sentences using Python's `translate` method.



```
def remove_punctuation(sentence):
    return sentence.translate(str.maketrans('', '', string.punctuation + '¿¡' ))
```



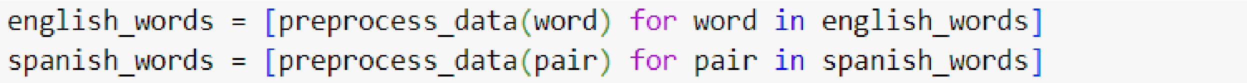
```
def preprocess_sentence(sentence):
    return remove_punctuation(sentence).lower()
```

The latter function then converted the cleaned sentences to lowercase, ensuring uniformity in our dataset. Applying these functions iteratively to each line of the input data allowed us to clean and format the words before adding them to their respective lists.

## II. Word to token

We then discuss the transformation of words into tokens, an essential step in preparing our data for further analysis and modeling. Tokens represent individual units of meaning within a text, facilitating the processing of natural language data.

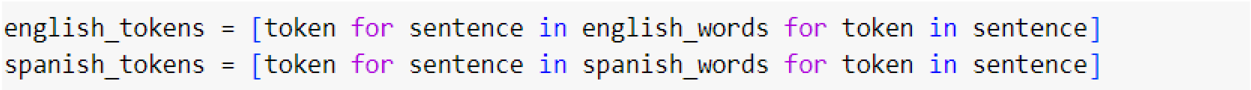
First, we cleaned up our data using a function called `preprocess_data` to make sure everything looked neat and tidy. This step was important because it made our data ready for the next step, which was turning words into tokens.



```
english_words = [preprocess_data(word) for word in english_words]
spanish_words = [preprocess_data(pair) for pair in spanish_words]
```

To do this, we used list comprehensions, which are like shortcuts in Python that help us do things quickly. We took our cleaned-up words and turned them into lists of tokens for both

English and Spanish. This made it easier to understand each word's meaning and use it for more detailed analysis later on.



```
english_tokens = [token for sentence in english_words for token in sentence]
spanish_tokens = [token for sentence in spanish_words for token in sentence]
```

To see what our conversion did, we printed some of the first tokens for English and Spanish as shown below.

```
English Tokens:
['they', 'can', 'play', 'it', 'correctly', 'and', 'at', 'the',
'desired', 'tempo']

Spanish Tokens:
['que', 'lo', 'puedan', 'tocar', 'correctamente', 'y', 'en', 'el',
'tiempo', 'esperado']
```

This helped us see how the words changed into tokens, which are like smaller, easier-to-handle versions of words.

## III. Mapping word/ Token into vectors

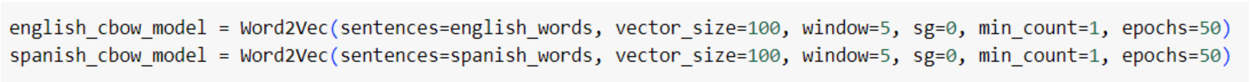
In this section, we focus on : Word Embedding, Training CBOW Models, and Mapping Words to Vectors.

- **Word Embedding:**

  **CBOW Model** (Continuous Bag of Words):

  It is a type of word embedding technique used in natural language processing (NLP) and deep learning. In simple terms, the CBOW model learns to predict a target word based on its surrounding context words. It views a sentence as a bag of words, ignoring the word order within the context window. The model's objective is to minimize the difference between the predicted word and the actual target word.

  We trained two separate models for English and Spanish using Word2Vec. This helps us understand the words better in each language.



```
english_cbow_model = Word2Vec(sentences=english_words, vector_size=100, window=5, sg=0, min_count=1, epochs=50)
spanish_cbow_model = Word2Vec(sentences=spanish_words, vector_size=100, window=5, sg=0, min_count=1, epochs=50)
```

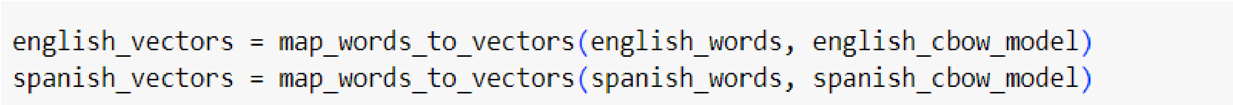
  The numbers like vector_size=100 and window=5 just tell the computer how many pieces to use for each word code and how many words to look at around each word.

- **Training CBOW models:**

  Then we train the CBOW Models on our tokenized English and Spanish word lists (`english_words` and `spanish_words`) with additional parameters like min_count=1 and epochs=50 to ensure all words are considered and to train the model over 50 iterations.

- **Mapping words to vectors:**

  We employed a function called map_words_to_vectors, which takes sentences and the trained CBOW model as inputs.



```
english_vectors = map_words_to_vectors(english_words, english_cbow_model)
spanish_vectors = map_words_to_vectors(spanish_words, spanish_cbow_model)
```

  It calculates the average vector for the words in each sentence, creating a single vector that represents the entire sentence. This helps us assign numerical codes to words or tokens based on what they mean in the model's learning space.We showed a few of these vectors for English and Spanish in our exploration below.

```
Word vectors in English:
Word: ['go']
Vector: [ 0.92366594 -0.7280116  -1.2059723  -2.3805184  -2.6583457
1.2683383
  0.6637322  -4.2237654  -3.5456178   1.7656622   2.7739527   1.6525729
 -1.0402126   0.5505132   2.346422   -0.31488332  1.2411407  -3.4963331
  0.44979593  1.8337791   0.5368441   5.735879    0.20823744  1.7234814
 -1.4656916   1.2293121  -2.2275267  -2.2815802   1.796643   -1.1820097
  2.0012927   1.5665565   0.04818101 -0.64601684 -0.03076627  0.66409117
 -1.4649093   0.58311     1.0100989   2.022628   -2.8650196  -0.21138239
  0.35404295 -0.28793836 -2.5470254  -2.1047397   1.6670882   1.7896149
 -3.9670653   0.80156493 -2.216885    2.5048666   1.3856401   2.763508
  0.8530027  -4.0131164   1.1729964  -2.3164601   1.3451235  -0.4901065
  0.9201349   1.3216107  -3.3779306   0.3452631   0.67750704 -0.83478993
 -0.43170938 -2.411647   -3.1084654  -0.9141381   0.61315966  0.9741446
 -2.033862   -0.56748515 -0.54188967  1.5232862  -2.883257    1.5744911
  0.34059215 -1.783904    0.7581417   0.200792    1.76337     0.8859576
  1.7508523   0.43281186 -2.4366908   0.08051158 -1.3614373   2.5544105
  0.69853574 -0.6999985   1.6231018  -0.64933753  1.1915553   0.87682104
  0.65181124 -0.03693308  2.1427631   1.1630725 ]
```

```
Word vectors in Spanish:
Word: ['hola']
Vector: [-0.02559798  0.11122648  0.40514195  1.2555202  -0.40214255
0.5874659
 -0.58721983  0.0510927   0.67088556 -0.6464764   1.5735848  -0.19287728
  0.13664687 -0.7624332   1.4506966   0.4946646   1.05813    -0.3892327
  0.22278574  0.01962568  0.49930462 -1.4226657  -0.3116571   0.07854027
 -0.06268866 -0.54573494 -0.55765676  0.7616842   0.13498832 -0.23330365
  0.57926303  0.49065772  0.51173705 -0.20583135  0.8803216  -0.24697973
  0.22562978 -0.11658735  0.0345217   1.084458    0.35474524 -0.75325197
 -0.1266411  -0.2613485   0.44969627  0.33251768 -0.25587037 -0.24538468
 -0.34931362  0.8101145   0.11951178  1.0826403   0.25955823 -0.361626
 -0.17686209  0.22507724 -0.94371694  0.12608373 -0.8845634   0.25932586
 -0.28095752 -0.07441682 -0.0053942  -0.34598696  0.5407425   0.08051082
  0.3709762   1.1596289  -0.0501066  -0.13961044 -1.1521978  -0.34912002
  0.35650894 -0.20751745 -0.48066428  1.3773065   0.03124664  0.15567198
  0.8323724   0.61123574 -0.42100626 -0.37391084  0.1866976   0.3869032
  0.18627584  0.45640647 -0.19204782 -0.28345716  1.2374809   0.05000104
  0.64734995  0.3972179  -0.77789307 -0.29285735 -0.33573633 -0.44427395

        0.16688606  0.93111765 -1.2114156  -0.7568519 ]
```

After printing a few word vectors from our English and Spanish CBOW models, we observed that each word is represented as a vector of 100 numerical values as specified in our algorithm.

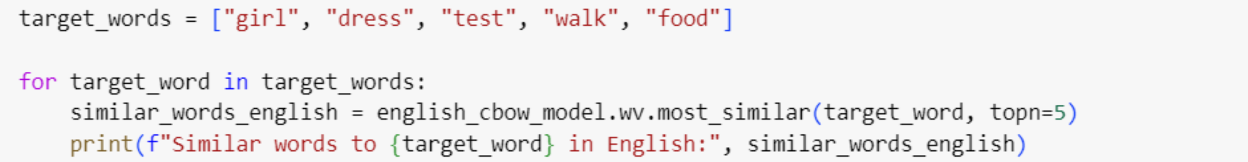
The purpose of printing these vectors was to understand how our word embedding technique maps words into a numerical space, enabling the model to learn relationships and similarities between words based on their contexts.

In conclusion, by converting words into vectors, we have prepared the foundational data required for training our neural translation model. These word vectors serve as the basis for capturing the meaning and context of words, facilitating accurate and meaningful translations.

## V. Nearest word examples in English and Spanish

We used the trained models to find words that are similar to specific target words in English and Spanish. The purpose was to understand how the model interprets and connects words based on their meanings.

In English, we provided target words like "girl," "dress," "test," "walk," and "food."



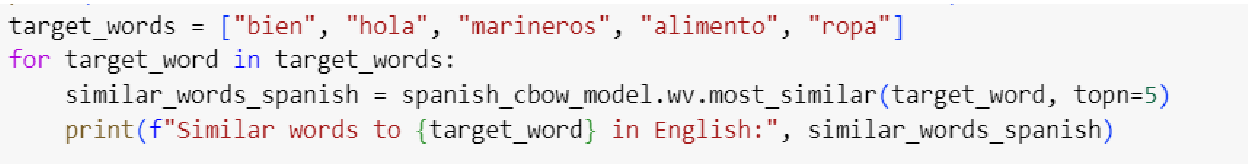
```
target_words = ["girl", "dress", "test", "walk", "food"]

for target_word in target_words:
    similar_words_english = english_cbow_model.wv.most_similar(target_word, topn=5)
    print(f"Similar words to {target_word} in English:", similar_words_english)
```

It then identified words that are closely related to these targets, such as "woman," "clothing," "exam," "stroll," and "meal." These similar words were chosen based on how often they appeared together with the target words in the training data, capturing semantic relationships.

```
Similar words to girl in English: ['boy', 'woman', 'child', 'doll', 'man']
Similar words to dress in English: ['skirt', 'hat',suit', 'sweater', 'shirt']
Similar words to test in English: ['exam', 'examination', 'trip','lecture',
'picnic']
Similar words to walk in English: ['drive','swim', 'shower', 'run', 'pray']
Similar words to food in English:
['fruit','vegetables','pizza','bread','rice']
```

Similarly, in Spanish, we provided target words like "bien" (good), "hola" (hello), "marineros" (sailors), "alimento" (food), and "ropa" (clothing).



```
target_words = ["bien", "hola", "marineros", "alimento", "ropa"]
for target_word in target_words:
    similar_words_spanish = spanish_cbow_model.wv.most_similar(target_word, topn=5)
    print(f"Similar words to {target_word} in English:", similar_words_spanish)
```

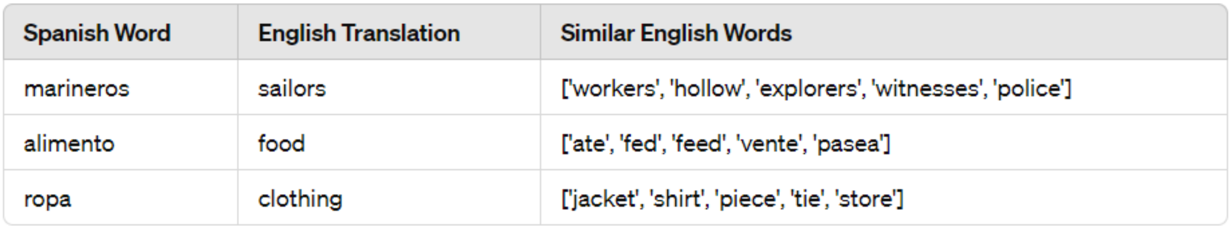
It found similar words like "good," "hello," "sailors," "food," and "clothing" by analyzing the contextual patterns of these words in the Spanish language corpus.

```
Similar words to bien in Spanish: ['mal','cargado','ocupado', 'exigente',
'meticulosamente']
Similar words to hola in Spanish: ['exactamente', 'mintiendo','denante',
'bromeando','crezcas']
Similar words to marineros in Spanish: [('obreros', 'huecos', 'exploradores',
'testigos', 'policías']
Similar words to alimento in Spanish: ['até','alimentaba','alimente',
'vente','pasea']
Similar words to ropa in Spanish: ['chaqueta', 'camisa', 'pieza', 'corbata',
'tienda']
```

As shown above, each word is represented as a vector of 100 numbers. Two words are considered close, if the "distance" between these two vector notations are relatively closer than other words. In this example, for each word, we show the results of 5 such "close words".

Consequently, we can also see that the words that are actually close to each other reflect a similar concept if not a similar meaning.

Here is a table showcasing the results of the word similarity analysis for selected Spanish words and their English translations, along with similar English words identified by the analysis:



| Spanish Word | English Translation | Similar English Words |
|---|---|---|
| marineros | sailors | ['workers', 'hollow', 'explorers', 'witnesses', 'police'] |
| alimento | food | ['ate', 'fed', 'feed', 'vente', 'pasea'] |
| ropa | clothing | ['jacket', 'shirt', 'piece', 'tie', 'store'] |

## VI. Preprocessing for feeding into the model:

After converting and training the word embedding model, we still need to preprocess the data a certain bit to make it compatible to be fed into a neural machine translation model.

We perform the following preprocessing:

1. Pad the sequence of words for consistent size.

```
max_english_seq_length = 100
max_spanish_seq_length = 100

# Padding sequences
english_vectors_padded = pad_sequences(english_vectors,
maxlen=max_english_seq_length, padding='post', truncating='post')
spanish_vectors_padded = pad_sequences(spanish_vectors,
maxlen=max_spanish_seq_length, padding='post', truncating='post')
```

This line pads or truncates the vectors (presumably representing text sequences) to ensure that they all have the same length. Padding is added after the sequences ('post') to make them match.

2. Convert the padded sequence into a numpy array.

```
english_vectors_np = np.array(english_vectors_padded)
spanish_vectors_np = np.array(spanish_vectors_padded)
```

3. Reshape it into a 3-dimensional array to make it easily compatible with LSTM encoder inputs.

```
english_vectors_np = np.expand_dims(english_vectors_np, axis=-1)
spanish_vectors_np = np.expand_dims(spanish_vectors_np, axis=-1)
```

   This operation effectively changes the shape of the array from, for example, (100, 50) to (100, 50, 1). This is often done to prepare data for feeding into neural networks where an extra dimension is needed, especially for single-channel data like single-feature vectors which is the word vector translation in our case.

## VII. Train-test split.

```
from sklearn.model_selection import train_test_split

english_train, english_test, spanish_train, spanish_test =
train_test_split(english_vectors_np, spanish_vectors_np, test_size=0.3)
```

We use the widely used **‘train_test_split’** of sklearn.model_selection in order to split the data randomly into 70% for training and 30% for testing.

In total, the number of data for training is:

```
Training data length:  83274
```

The number of data for testing is:

```
Testing data length:  35690
```

Although for a natural language processing model, this number may not be very large, however, our results show that training an appropriate model with this much data should be good enough (not the best) to get the model to learn a single language translation, in our case English to Spanish.

## VIII. Model architecture.

We have designed a very simple LSTM model with encoder and decoder layers that will help in the translation of english sequences to spanish.

### A. Input layer:

The input layer is a simple Input layer from Keras that has the shape of the maximum possible sequence of english tokens in our dataset.

```
encoder_inputs = Input(shape=(vector_size, 1))
```

The maximum sequence length for english and spanish tokens are calculated by counting the maximum number of words in a single sentence of the training set:

```
max_eng_seq_length = max(len(sentence) for sentence in english_words)
max_spa_seq_length = max(len(sentence) for sentence in spanish_words)
```

### B. Encoder Layer:

We define two LSTM layers as part of the encoders in order to capture the features received in the word vectors. Both of these LSTM layers have ***return_sequence***=True and ***return_state***=True.

This enables the layer to return the entire sequence , even those of the hidden state to be passed on to the next layer. This is specifically helpful because for us the entire input sequence is relevant for making the prediction.

We also have an attention layer which filters these sequence outputs and focuses only on the relevant parts of the input.

```
# Encoder Layer
encoder_lstm1 = LSTM(latent_dim, return_sequences=True,
return_state=True)
encoder_outputs1, state_h1, state_c1 = encoder_lstm1(encoder_inputs)

encoder_lstm2 = LSTM(latent_dim, return_sequences=True,
return_state=True)
encoder_outputs, state_h, state_c = encoder_lstm2(encoder_outputs1)
encoder_states = [state_h, state_c]
```

### C. Decoder Layer:

Based on our literature survey on building machine translation programs and on transformers, we found that the decoder often benefits from being initialized with the final states of the encoder. This is because the final states of the encoder contain a compressed representation of the input sequence, capturing its context and relevant information. Because of this reason, we attach the final states of the encoder as the initial state of the decoder.

```
# Decoder Input
decoder_inputs = Input(shape=(vector_size, 1))

# Decoder Layer
decoder_lstm1 = LSTM(latent_dim, return_sequences=True,
return_state=True)
decoder_outputs1, _, _ = decoder_lstm1(decoder_inputs,
initial_state=encoder_states)
```

## D. Attention Mechanism:

```
attention_layer = Attention()
# Attaching weights to each decoder output based on encoder output
attention_out = attention_layer([decoder_outputs1, encoder_outputs])
# Feeding the output as input to subsequent sequence
decoder_concat_input = Concatenate(axis=-1)([decoder_outputs1,
attention_out])
```

This is one of the most important layers in this model.  This layer helps focus or filter on the important and relevant information of the input making it possible to relate an input vector to an output. There are basically two parts here:

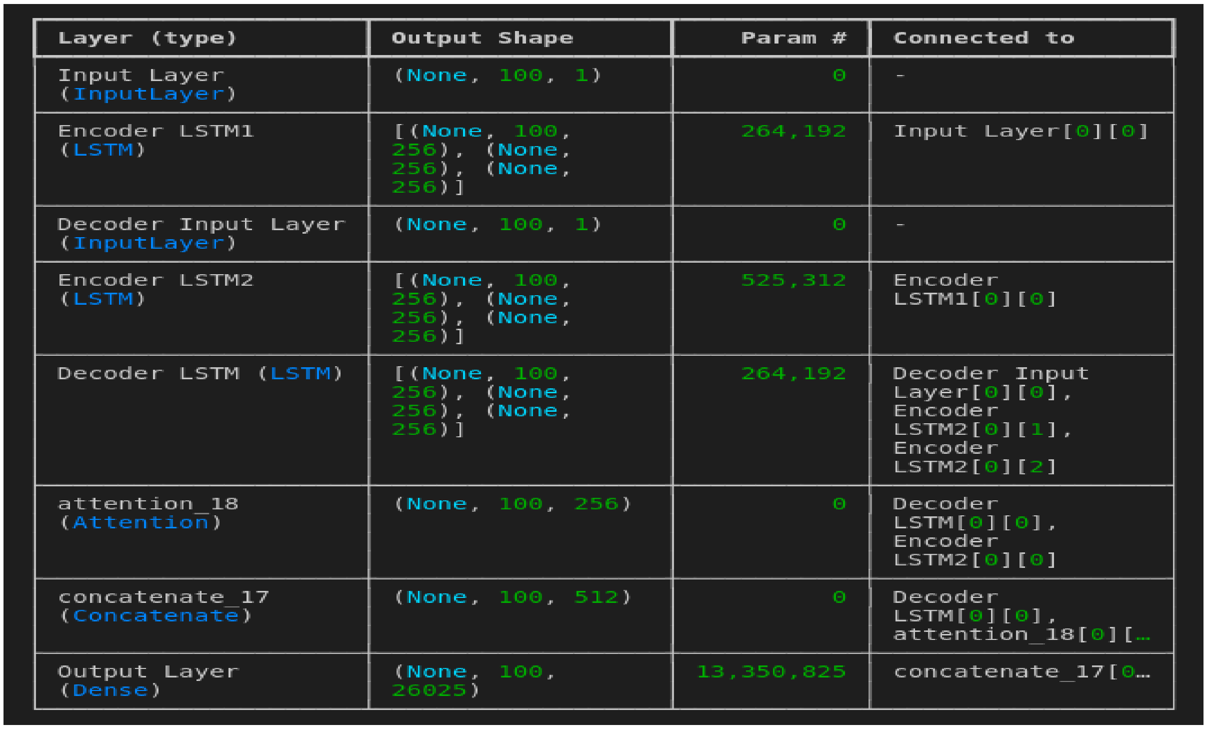
a. Attention Weights:
   The ***attention_out*** computes the attention weights for each timestep of the decoder output (decoder_outputs1) with respect to the encoder outputs (encoder_outputs). It provides a context vector at each decoder timestep by considering the relevance of each encoder output timestep to the current decoder timestep.
b. Attention output as input to next sequence:
   The output from this layer is provided as input to the subsequent layers of the decoder.

## E. Output Layer:

The output for our model is a fully-connected dense layer with output neurons as the size of the spanish dictionary. The activation function is a ***softmax*** function which **helps attach probabilities to all the possible outputs generated by the model**. Ultimately, we choose the word that has the highest probability as the output during that time step.

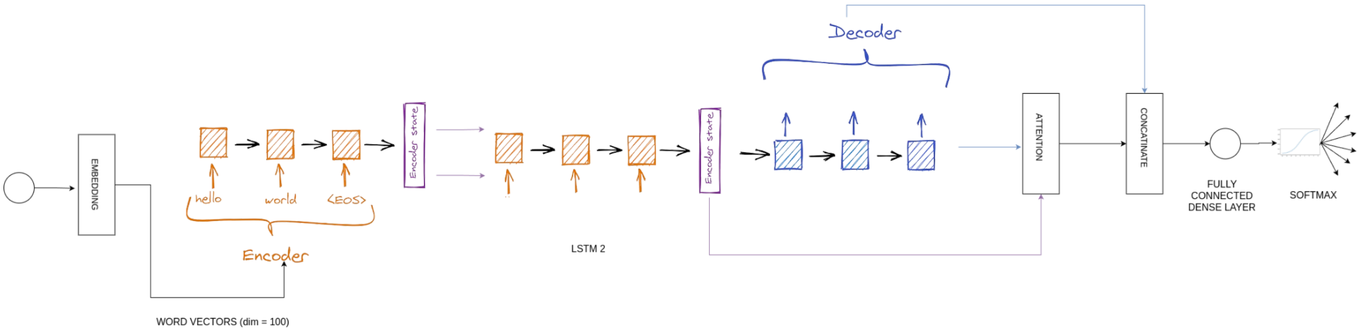
```
decoder_dense = Dense(spanish_vocab_size, activation='softmax')
output_layer = decoder_dense(decoder_concat_input)
```

## F. Final View of Model:



| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| Input Layer (InputLayer) | (None, 100, 1) | 0 | - |
| Encoder LSTM1 (LSTM) | [(None, 100, 256), (None, 256), (None, 256)] | 264,192 | Input Layer[0][0] |
| Decoder Input Layer (InputLayer) | (None, 100, 1) | 0 | - |
| Encoder LSTM2 (LSTM) | [(None, 100, 256), (None, 256), (None, 256)] | 525,312 | Encoder LSTM1[0][0] |
| Decoder LSTM (LSTM) | [(None, 100, 256), (None, 256), (None, 256)] | 264,192 | Decoder Input Layer[0][0],<br>Encoder LSTM2[0][1],<br>Encoder LSTM2[0][2] |
| attention_18 (Attention) | (None, 100, 256) | 0 | Decoder LSTM[0][0],<br>Encoder LSTM2[0][0] |
| concatenate_17 (Concatenate) | (None, 100, 512) | 0 | Decoder LSTM[0][0],<br>attention_18[0][… |
| Output Layer (Dense) | (None, 100, 26025) | 13,350,825 | concatenate_17[0… |

MODEL ARCHITECTURE OF ENGLISH - SPANISH TRANSLATION



## G. Training the model:

We use the RMSProp as the optimizer with learning_rate = 0.1 since it is known to help the program converge faster for a classification problem. For the loss function, we tested with ***mean-squared-error(mse)*** as well as with ***sparse-categorical-cross entropy***. Considering this problem as a classification, we assumed the sparse-categorical-cross entropy might be more performant than the mse error, however, we did not observe much of a difference in the loss or accuracy while using both of them.

```
optimizer = RMSprop(learning_rate=0.1)
model.compile(optimizer=optimizer, loss='mse')

model.fit([english_train, spanish_train[:, :-1], spanish_train[:, 1:],
         batch_size=32,
         epochs=30,
         validation_split=0.1)
```

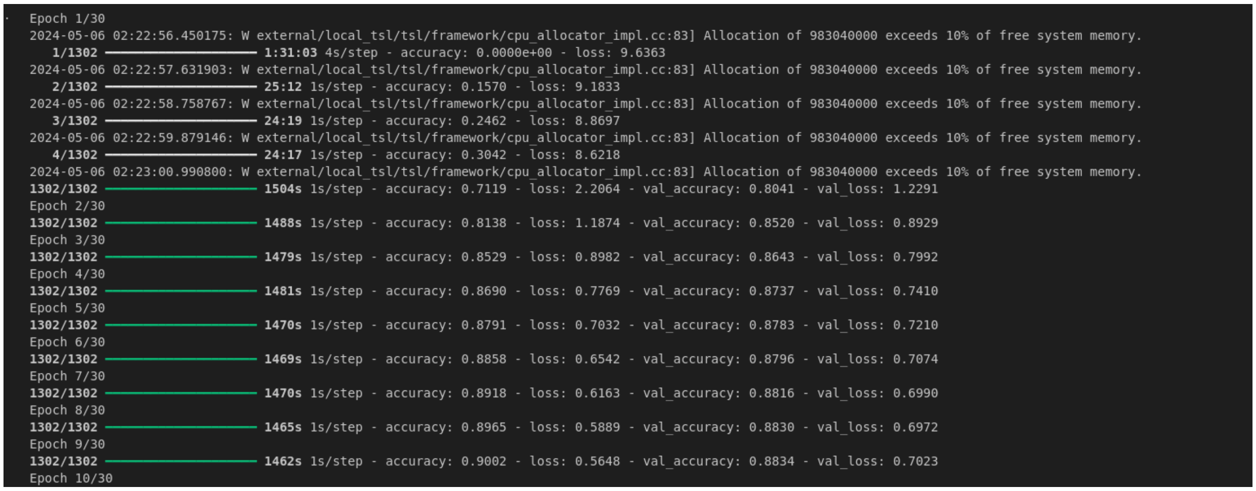
The training is done by passing the encoder input and decoder input along with the target sequence. Since we are interested in sequence-to-sequence translation, the target output is always provided as the next output in the sequence.

Training is done over 30 epochs, with a batch size of 32.

# 4. Results and analysis:

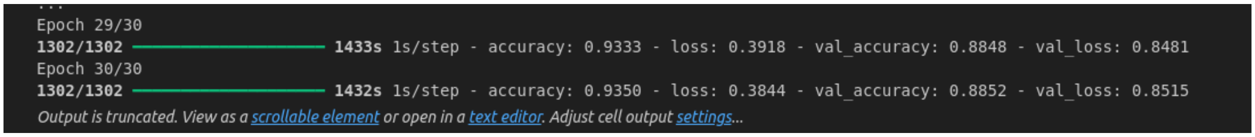
With training the model on a total of **118964** tokens, running it over 30 epochs, the model took a hefty  ~11 hours to train and produce the results. The sections below show a detailed results and analysis of the model that we developed and tested.

## I. Learning at each epoch:



```
Epoch 1/30
2024-05-06 02:22:56.450175: W external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 983040000 exceeds 10% of free system memory.
   1/1302 ━━━━━━━━━━━━━━━━━━━━ 1:31:03 4s/step - accuracy: 0.0000e+00 - loss: 9.6363
2024-05-06 02:22:57.631903: W external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 983040000 exceeds 10% of free system memory.
   2/1302 ━━━━━━━━━━━━━━━━━━━━ 25:12 1s/step - accuracy: 0.1570 - loss: 9.1833
2024-05-06 02:22:58.758767: W external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 983040000 exceeds 10% of free system memory.
   3/1302 ━━━━━━━━━━━━━━━━━━━━ 24:19 1s/step - accuracy: 0.2462 - loss: 8.8697
2024-05-06 02:22:59.879146: W external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 983040000 exceeds 10% of free system memory.
   4/1302 ━━━━━━━━━━━━━━━━━━━━ 24:17 1s/step - accuracy: 0.3042 - loss: 8.6218
2024-05-06 02:23:00.990800: W external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 983040000 exceeds 10% of free system memory.
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 1504s 1s/step - accuracy: 0.7119 - loss: 2.2064 - val_accuracy: 0.8041 - val_loss: 1.2291
Epoch 2/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 1488s 1s/step - accuracy: 0.8138 - loss: 1.1874 - val_accuracy: 0.8520 - val_loss: 0.8929
Epoch 3/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 1479s 1s/step - accuracy: 0.8529 - loss: 0.8982 - val_accuracy: 0.8643 - val_loss: 0.7992
Epoch 4/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 1481s 1s/step - accuracy: 0.8690 - loss: 0.7769 - val_accuracy: 0.8737 - val_loss: 0.7410
Epoch 5/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 1470s 1s/step - accuracy: 0.8791 - loss: 0.7032 - val_accuracy: 0.8783 - val_loss: 0.7210
Epoch 6/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 1469s 1s/step - accuracy: 0.8858 - loss: 0.6542 - val_accuracy: 0.8796 - val_loss: 0.7074
Epoch 7/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 1470s 1s/step - accuracy: 0.8918 - loss: 0.6163 - val_accuracy: 0.8816 - val_loss: 0.6990
Epoch 8/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 1465s 1s/step - accuracy: 0.8965 - loss: 0.5889 - val_accuracy: 0.8830 - val_loss: 0.6972
Epoch 9/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 1462s 1s/step - accuracy: 0.9002 - loss: 0.5648 - val_accuracy: 0.8834 - val_loss: 0.7023
Epoch 10/30
```

We can see that the model is learning properly with the decrease in training as well as validation loss, while increasing the accuracy.



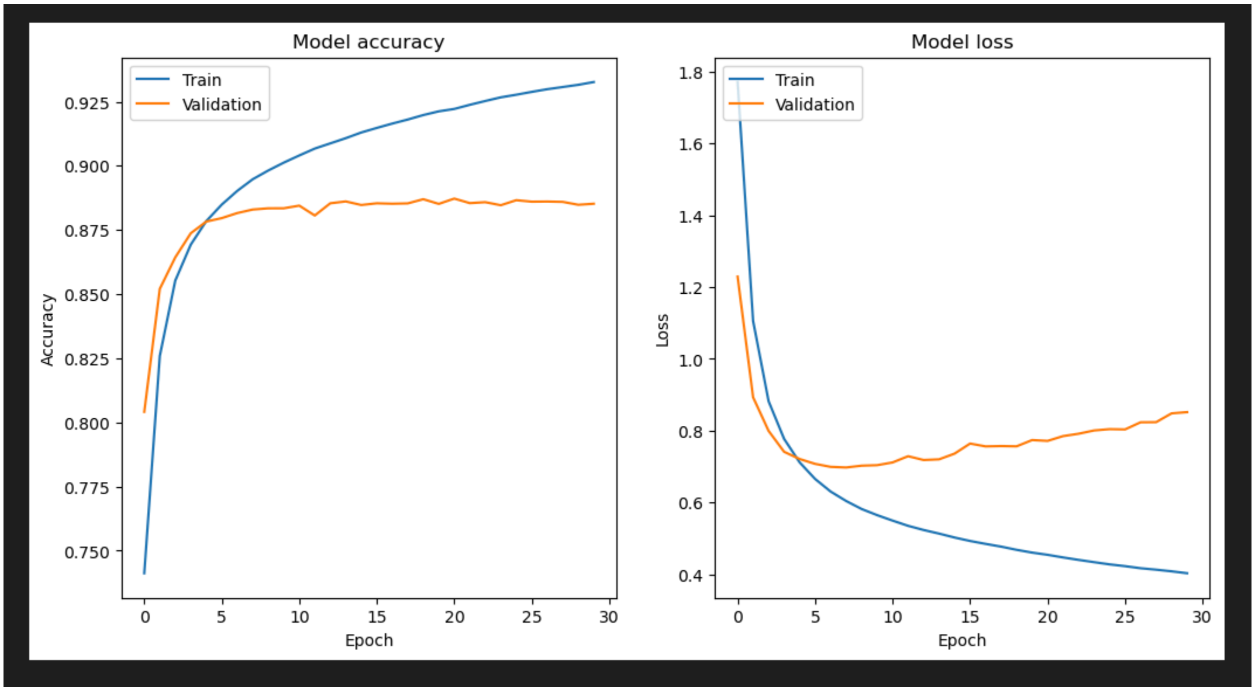
```
...
Epoch 29/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 1433s 1s/step - accuracy: 0.9333 - loss: 0.3918 - val_accuracy: 0.8848 - val_loss: 0.8481
Epoch 30/30
1302/1302 ━━━━━━━━━━━━━━━━━━━━ 1432s 1s/step - accuracy: 0.9350 - loss: 0.3844 - val_accuracy: 0.8852 - val_loss: 0.8515
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Ultimately, we see the final loss is around 30%, which needs improvement but can still be considered an acceptable amount for a simple translation model.

## II. Accuracy and Loss

The model at the end of the training and testing reports an accuracy of around ~80% accuracy for the test data, which it had never seen before.

We also test the model for some random english translation and validate the translated text with the Google Translate API. We have portrayed the results of the same in the further sections below.



This is the learning curve or the loss curve of the training and validation of the model. While the training curve is as expected, we see a small gap between the validation and training curve.

## A. Signs of Overfitting

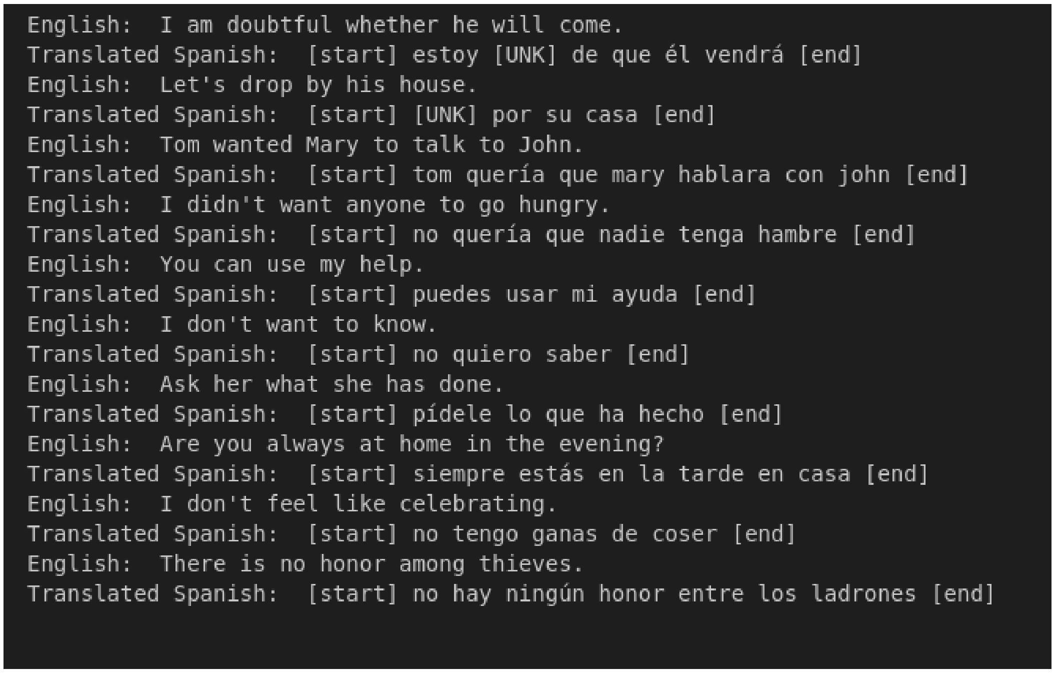
In the learning curve above, we see that the validation accuracy increases for a certain period of time but after a certain while, the validation accuracy remains close to constant. But on the other hand, we see that the training accuracy continues to increase throughout the training period. The case of the loss is also similar.
This shows signs of overfitting.

Our future work is to enhance this model by adding different Normalization and Dropouts layers to further improve the accuracy and make it near 90% by avoiding overfitting.
While a total of 118964 tokens is a good amount of data, we believe it is not a large or optimal amount of data to train a highly accurate model, so we might also have to dig deeper to find a larger dataset for training this translation model.

## III. Translation examples:

In this section, we show some examples of the translation from English to Spanish performed by the model and validate it with the actual translation from Google Translate API.

We randomly feed the model with 10 different English sentences and show the results in Spanish:



```
English:  I am doubtful whether he will come.
Translated Spanish:  [start] estoy [UNK] de que él vendrá [end]
English:  Let's drop by his house.
Translated Spanish:  [start] [UNK] por su casa [end]
English:  Tom wanted Mary to talk to John.
Translated Spanish:  [start] tom quería que mary hablara con john [end]
English:  I didn't want anyone to go hungry.
Translated Spanish:  [start] no quería que nadie tenga hambre [end]
English:  You can use my help.
Translated Spanish:  [start] puedes usar mi ayuda [end]
English:  I don't want to know.
Translated Spanish:  [start] no quiero saber [end]
English:  Ask her what she has done.
Translated Spanish:  [start] pídele lo que ha hecho [end]
English:  Are you always at home in the evening?
Translated Spanish:  [start] siempre estás en la tarde en casa [end]
English:  I don't feel like celebrating.
Translated Spanish:  [start] no tengo ganas de coser [end]
English:  There is no honor among thieves.
Translated Spanish:  [start] no hay ningún honor entre los ladrones [end]
```

Validation table:
The following table acts as a truth table for the results.

| English [Input] | Spanish [Prediction] | Actual | Comments |
|---|---|---|---|
| I am doubtful whether he will come. | estoy [UNK] de que él vendrá | I'm [UNK] that he will come | One character not predicted by model. |
| You can use my help. | puedes usar mi ayuda | You can use my help | 100 % Correct. |
| I didn't want anyone to go hungry | no quería que nadie tenga hambre | I didn't want anyone to go hungry | 100 % Correct. |
| Are you always at home in the evening? | siempre estás en la tarde en casa | you are always at home in the afternoon | Grammatical difference / error. |
| Tom wanted Mary to talk to John. | tom quería que mary hablara con john | Tom wanted Mary to talk to John. | 100 % Correct. |

### Comments on the translation:

We can observe that in most cases, the model correctly is able to perform a sequence to sequence prediction. However, there are certain cases where the prediction is not very accurate.

In the first row, we can see that it is not able to predict the word **‘doubtful’**. It could be because it had never seen this word during training and could also not find any word close to this word in the vector representation.

Likewise, in the second last row, we can see that the grammatical composition of the sentence is incorrect. The input is a question but the translation is a sentence.

# 5. Conclusion

In conclusion, this project investigated the effectiveness of a sequence-to-sequence encoder-decoder architecture with LSTMs for Spanish to English machine translation. The model achieved promising translation results, but slight overfitting was observed. To address this, future work will incorporate Dropout layers and improved normalization techniques. This will enhance the model's generalization capabilities and potentially lead to even more accurate translations.

# 6. References

1. Ian Goodfellow et al., Deep Learning, The MIT Press, 2016.

2. J. Kaplan et al., "Scaling Laws for Neural Language Models," 2020. [Online]. https://arxiv.org/abs/2001.08361

3. "Neural machine translation with Keras NLP," Keras Documentation. https://keras.io/examples/nlp/neural_machine_translation_with_keras_nlp/.

4. T.-L. Ha, J. Niehues, and A. Waibel, "Toward Multilingual Neural Machine Translation with Universal Encoder and Decoder," 2016. https://arxiv.org/abs/1611.04798

5. T. Kenter, A. Borisov, and M. de Rijke, "Siamese CBOW: Optimizing Word Embeddings for Sentence Representations," 2016. https://arxiv.org/abs/1606.04640Ian Goodfellow et al., Deep Learning, The MIT Press, 2016.