# GUIDE Encoders and Decoders for Neural Machine Translation

<https://app.pluralsight.com/guides/encoders-and-decoders-for-neural-machine-translation>

## Introduction

There are over 7,000 languages in the world. However, only 23 languages in total are most spoken around the globe, including English, Mandarin Chinese, Hindi, and Spanish. As the world is connecting faster, language translation bridges the communication gap.



[Image Source](https://en.clipdealer.com/vector/media/A:123557489)

Google Translate can translate not only text but also speech and images in real-time. You can use it on your laptop, mobile, or even smartwatch. This guide will show the technology behind this magic.

Before moving further, I would recommend reading these guides on [RNN](https://app.pluralsight.com/guides/getting-started-with-rnn) and [LSTM](https://app.pluralsight.com/guides/introduction-to-lstm-units-in-rnn).

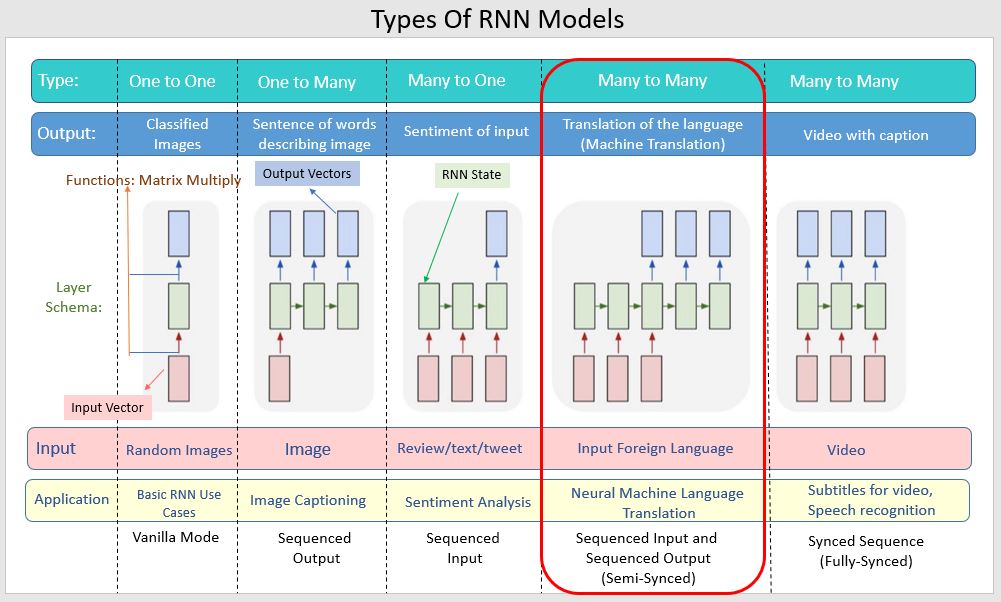
To follow along with this guide, download and unzip the spa-eng.zip file [here](http://www.manythings.org/anki/). You will only use the *spa.txt* file for this process.

Let’s get started.

## The Power of Sequence2Sequence (seq2seq) Modeling

There are multiple tasks that can be solved by using seq2seq modeling, including text summarization, speech recognition, image and video captioning, and question answering. It can also be used in genomics for DNA sequence modeling. A seq2seq model has two parts: an encoder and a decoder. Both work separately and come together to form a huge neural network model.

This architecture has the ability to handle the input and output sequences of variable length. The below image shows the types of RNN models and their use cases.



## Encoder and Decoder

The following sections will cover encoder-decoder in-depth.

### Encoder

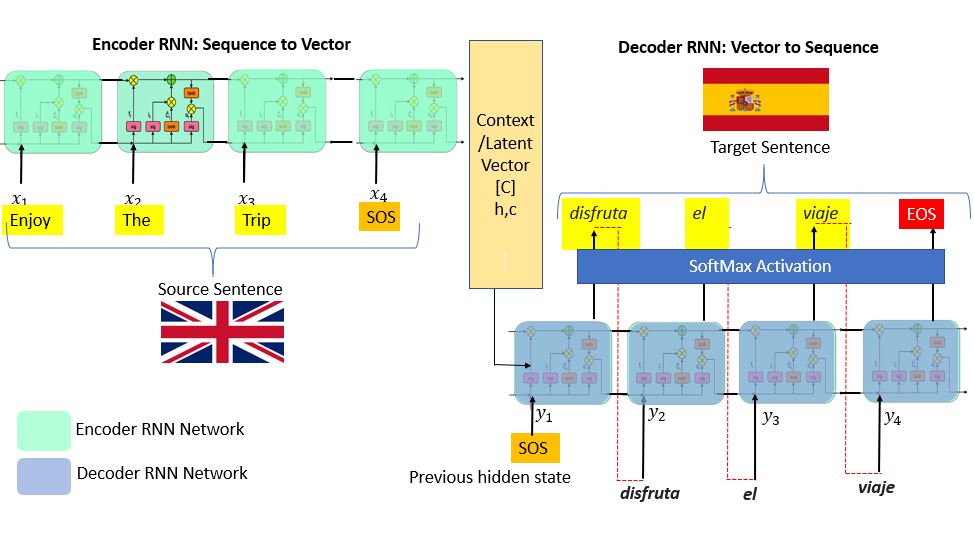
The encoder is at the feeding end; it understands the sequence and reduces the dimension of the input sequence. The sequence has a fixed size known as the *context vector*. This context vector acts like input to the decoder, which generates an output sequence when reaching the end token. Hence, you can call these seq2seq models encoder-decoder models.

This architecture can handle input and output sequences of variable length.

### Decoder

If you use LSTM for the encoder, use the same for the decoder. But it’s slightly more complex than the encoder network. You can say the decoder is in an “aware state.” It knows what words you have generated so far and what the previous hidden state was. The first layer of the decoder is initialized by using the context vector ‘C’ from the encoder network to generate the output. Then a special token is applied at the start to indicate the output generation. It applies a similar token at the end. The first output word is generated by running the stacked LSTM layers. A *SoftMax* activation function applies to the last layer. Its job is to introduce non-linearity in the network. Now this word is passed through the remaining layers and the generation sequence is repeated.

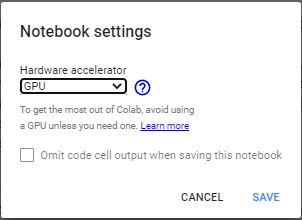
Multiple factors depend upon improving the accuracy of the encoder-decoder model. The hyper-parameters such as optimizers, cross-entropy loss, learning rate, etc., play an important role in improving the model’s performance.



## Importing the Libraries and Loading the Dataset.

This example will cover the simple implementation of seq2seq modeling in [Keras](https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html). I would suggest running the model on GPU. You can take advantage of [Google Colab’s](https://colab.research.google.com/) free GPU feature.

Go to **Edit**, then **Notebook Settings**, make changes, and save .



Mount your drive first:

| 1  2 | 1from google.colab import drive  2drive.mount('/content/drive') |
| --- | --- |

python

Copy and paste the authentication code and press enter.

Set up an environment, install the libraries, and define the parameters:

| 1  2  3  4  5  6  7 | 1import tensorflow as tf  2from tensorflow import keras  3from keras.layers import \*  4from keras.models import \*  5from keras.utils import \*  6from keras.initializers import \*  7from keras.optimizers import \* |
| --- | --- |

python

Define the parameter and set up the path for the spa.txt file you downloaded earlier on your drive. Define batch size, epochs to train for, LSTM latent dimensionality for the encoder, and the number of samples.

| 1  2  3  4  5  6 | 1batch\_size = 64  2epochs = 100  3latent\_dim = 256  4num\_samples = 10000  5# set the data\_path accordingly  6data\_path = "/content/drive/My Drive/spa.txt" |
| --- | --- |

python

Change the data\_path accordingly.

## Pre-processing

You won’t be required to conduct in-depth text pre-processing steps. But if you want to know more about noises associated and text pre-processing, kindly refer to this [Importance of Text Processing](https://app.pluralsight.com/guides/importance-of-text-pre-processing) guide. You can use tokenization; its job is to convert the input sentence into a sequence of integers. To achieve this, pass your data by using Keras’s Tokenizer() class.

Next, vectorize the data. It will read each line and append a list to it. The top three lines are below .

| 1  2  3  4  5  6 | 1input\_texts = []  2target\_texts = []  3input\_characters = set()  4target\_characters = set()  5with open(data\_path, "r", encoding="utf-8") as f:  6 lines = f.read().split("\n") |
| --- | --- |

python

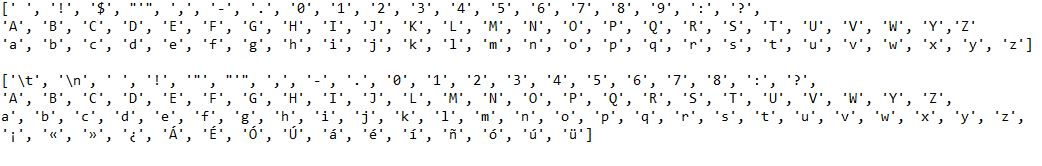
This example sets the parameter to 10,000 samples. The first two lines of the code below will put the English text in the input\_text and Spanish text in target\_text.

| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | 1for line in lines[: min(num\_samples, len(lines) - 1)]:  2 input\_text, target\_text, \_ = line.split("\t")  3 ############### A ###############  4 target\_text = "\t" + target\_text + "\n"  5 input\_texts.append(input\_text)  6 target\_texts.append(target\_text)  7 ############### B ###############  8 for char in input\_text:  9 if char not in input\_characters:  10 input\_characters.add(char)  11 for char in target\_text:  12 if char not in target\_characters:  13 target\_characters.add(char)  14print(input\_characters)  15print(target\_characters) |
| --- | --- |

python

The next step is to define the start and the end of sequence character using tab ( \t ) at the start of the character and \n at the end of the character.

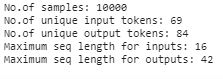
Along with the English and Spanish text, you’ll also want a list of their unit characters. The corresponding list output is below.



Define the parameters. They are important while building the model and feature engineering.

| 1  2  3  4  5  6  7  8  9  10  11  12 | 1input\_characters = sorted(list(input\_characters))  2target\_characters = sorted(list(target\_characters))  3num\_encoder\_tokens = len(input\_characters)  4num\_decoder\_tokens = len(target\_characters)  5max\_encoder\_seq\_length = max([len(txt) for txt in input\_texts])  6max\_decoder\_seq\_length = max([len(txt) for txt in target\_texts])  7  8print("No.of samples:", len(input\_texts))  9print("No.of unique input tokens:", num\_encoder\_tokens)  10print("No.of unique output tokens:", num\_decoder\_tokens)  11print("Maximum seq length for inputs:", max\_encoder\_seq\_length)  12print("Maximum seq length for outputs:", max\_decoder\_seq\_length) |
| --- | --- |

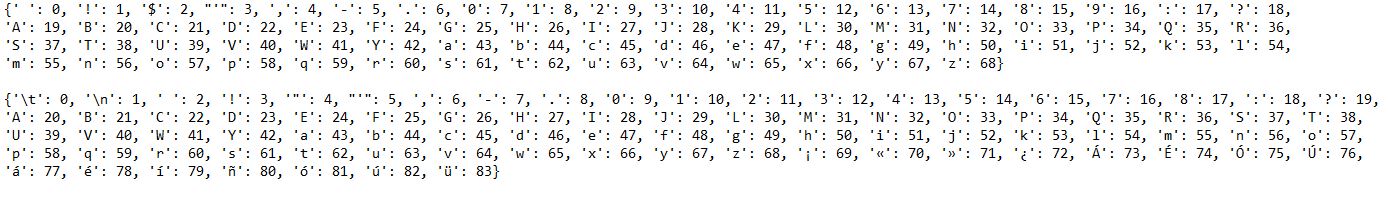
python



Now that you have a list of the characters, perform index mapping to input and target it.

| 1  2  3  4  5 | 1input\_token\_index = dict([(char, i) for i, char in enumerate(input\_characters)])  2target\_token\_index = dict([(char, i) for i, char in enumerate(target\_characters)])  3  4print(input\_token\_index)  5print(target\_token\_index) |
| --- | --- |

python



Notice that each character is now associated with an integer value.

Refer the [Keras documentation](https://keras.io/examples/nlp/lstm_seq2seq/) on pre-processing for more detail.

## Feature Engineering

To generate feature vectors, on-hot encoding is used. Turn 3D numpy arrays to store one-hot encoding. To generate the feature’s variables, encoder\_input\_data, decoder\_input\_data, decoder\_target\_data are used. encoder\_input\_data and decoder\_input\_data contain one-hot vectorization of English and Spanish sentences, respectively.

The first dimension, input\_texts, states the number of sample texts (10,000 in this case). The second dimension, max\_encoder\_seq\_length (English) and max\_decoder\_seq\_length (Spanish), is the longest encoder/decoder sequence length within the samples. The third dimension, num\_encoder\_tokens (English) and num\_decoder\_tokens (Spanish), contains unique characters in input\_charaters and output\_characters.

The decoder\_target\_data is like decoder\_input\_data, the only difference is that the decoder\_target\_data is offset by one timestamp. The decoder\_target\_data[:, t, :] is the same as decoder\_input\_data[:, t + 1, :] .

Now that everything is set, build the model and put the above variables and feature vectors to their proper encoder-decoder model.

| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | 1encoder\_input\_data = np.zeros(  2 (len(input\_texts), max\_encoder\_seq\_length, num\_encoder\_tokens), dtype="float32"  3)  4  5decoder\_input\_data = np.zeros(  6 (len(input\_texts), max\_decoder\_seq\_length, num\_decoder\_tokens), dtype="float32"  7)  8  9decoder\_target\_data = np.zeros(  10 (len(input\_texts), max\_decoder\_seq\_length, num\_decoder\_tokens), dtype="float32"  11)  12  13for i, (input\_text, target\_text) in enumerate(zip(input\_texts, target\_texts)):  14 for t, char in enumerate(input\_text):  15 encoder\_input\_data[i, t, input\_token\_index[char]] = 1.0  16 encoder\_input\_data[i, t + 1 :, input\_token\_index[" "]] = 1.0  17 for t, char in enumerate(target\_text):  18 decoder\_input\_data[i, t, target\_token\_index[char]] = 1.0  19 if t > 0:  20 decoder\_target\_data[i, t - 1, target\_token\_index[char]] = 1.0  21 decoder\_input\_data[i, t + 1 :, target\_token\_index[" "]] = 1.0  22 decoder\_target\_data[i, t:, target\_token\_index[" "]] = 1.0 |
| --- | --- |

python

## Conclusion

The fundamental idea of this guide was to give a brief understanding of the seq2seq model, encoder, and decoder. [This guide](https://app.pluralsight.com/guides/nmt:-encoder-and-decoder-with-keras) will help you take this to the next level by teaching you how to build a model using LSTM RNN.

You can now choose any language of your choice. Just download the language you want to translate and define a proper path of the data. Before moving further, make sure you understand LSTM well. Feel free to ask at [Codealphabet](https://codealphabet.com/contact) if you have any queries regarding this guide.

#NLP