# Code Mix Generation

## Team 17 - Qwertyuiop

**Professor :** Manish Shrivastava
**Mentor :** Prashant Kodali

**Submitted by :**
Anishka Sachdeva (2018101112)
Satyam Viksit Pansari (2018101088)

# Aim

We'll be building a machine learning model to go from one sequence to another i.e.

To build a **2 way translation system** which translates :

**i. Monolingual sequence to code mix sequence**

**ii. Code mix sequence to monolingual sequence**

# Tasks Done

**Baseline :**

- Went through the research papers.
- Studied the different models like LSTM, Bi-LSTM, GRU.
- Preprocessed the data.
- Almost implemented the baseline. Still working on the testing aspect.
- After further testing we will proceed towards baseline+.

**Baseline+ :**

- Completed the baseline model.
- Worked with LID Tags.
- Measured performance of the Seq2Seq with Attention Models using BLEU Score.

# BaseLine

# Task 1 : Dataset Selection

**Dataset used :** https://ritual.uh.edu/lince/datasets (containing sentences from **English -> Hindi-English(codemix)**). The dataset in general contains sentences about movies and its characters.

**Model 1 :**

**Source sequence :** English

**Target sequence :** Codemix

**Model 2 :**

**Source sequence :** Codemix

**Target sequence :** English

# Task 2 : Preprocessing and Tokenization

**Library used** : **Spacy**

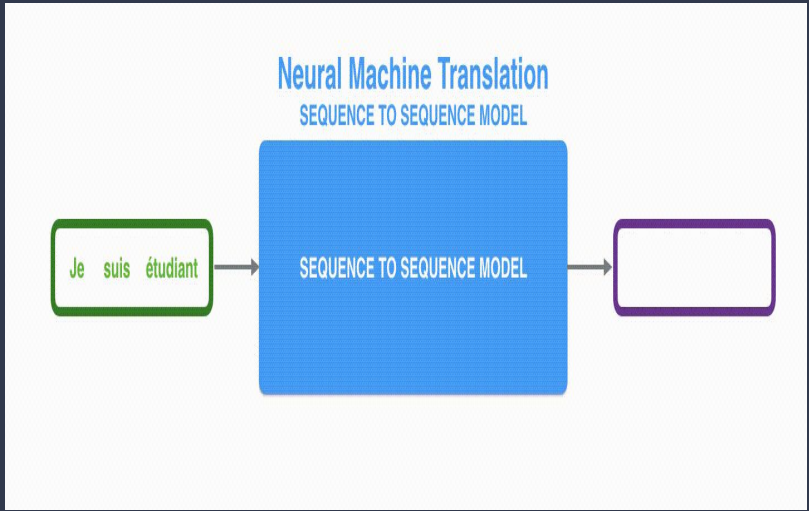
Spacy tool provides tokenizers for all languages and we have used the english one for both the sequences as of now (for baseline) .

**Tokenizer used** : **en_core_web_sm**

Using the min_freq argument, we only allow tokens that appear at least 2 times to appear in our vocabulary.Tokens that appear only once are converted into an <unk> (unknown) token .

Now, vocabulary is built for both the Source and Target sequences.

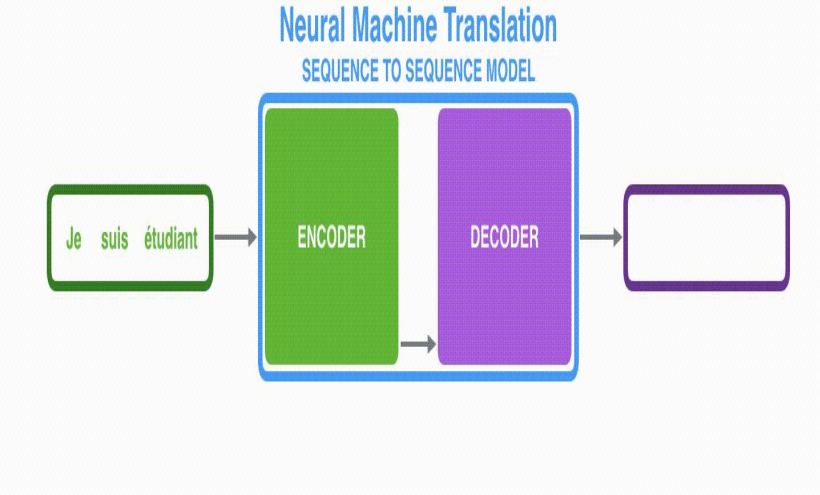
# Task 3 : Building Translator Model (Seq2Seq)




We' have built our model in three parts :

1. The encoder
2. The decoder
3. Seq2seq model (encapsulates the encoder and decoder and will provide a way to interface with each.)

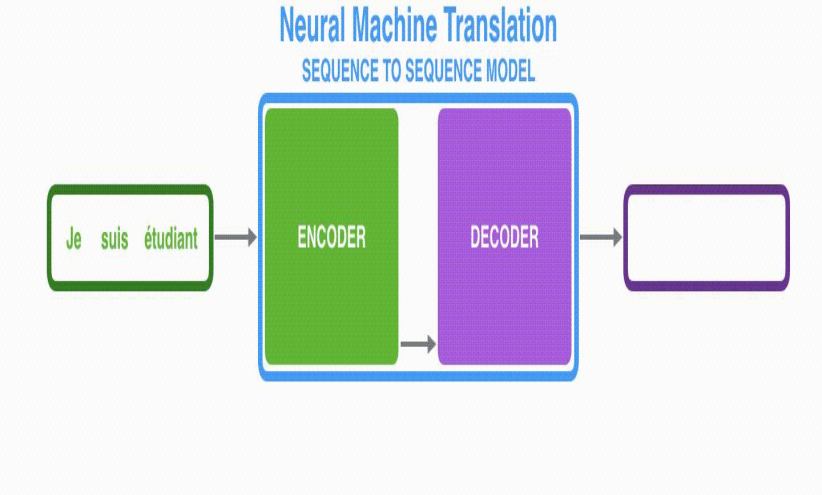
# Seq2seq Model : A High Level Overview




In the machine translation task, we have an input sequence x1,x2,...,xm and our model will generate an output sequence y1,y2,...,yn (note that their lengths can be different).

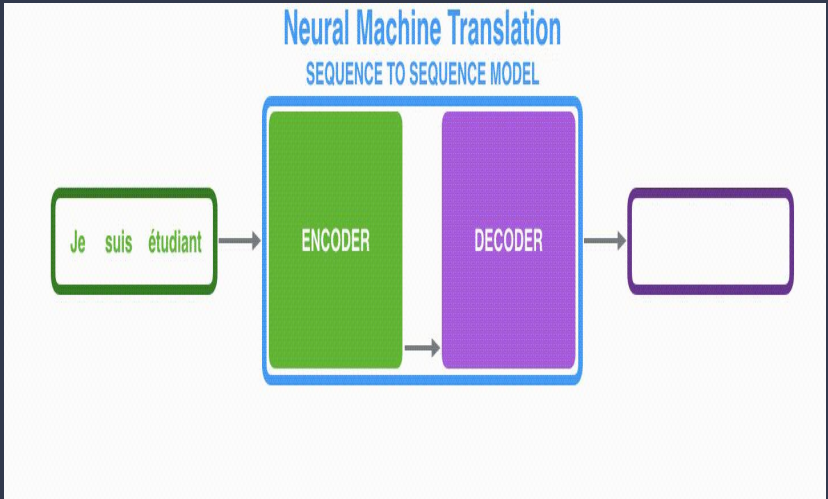
**Encoder-decoder** is the standard modeling paradigm for sequence-to-sequence tasks. It uses a ***recurrent neural network*** **(RNN)** to ***encode*** the source (input) sentence into a single vector called as **context vector**. The vector is then ***decoded*** by a second **RNN** which learns to output the target (output) sentence by generating one word at a time. The hope is that the final encoder state "encodes" all information about the source, and the decoder can generate the target sentence based on this vector.

# Seq2seq Model : A High Level Overview (Continued)




The input sentence is passed through a layer called as an **Embedding Layer** and it's output is further inputted to the **RNN Encoder**. Since the encoder uses **GRU (Gated Recurrent Units)** to encode the sentence, it uses the concept of using hidden states from previous timestamp (hidden state can be considered as a vector representation of the sentence so far). Therefore at each timestamp, the input to the RNN encoder is : embedding of the current word of the input sentence and the previous hidden state. The same procedure is followed for all the words in the input sentence. Once the final word has been passed into the RNN via the embedding layer, we use the final hidden state to be the output vector consisting of all the encoded information about the input sentence. This is a vector representation of the entire source sentence.

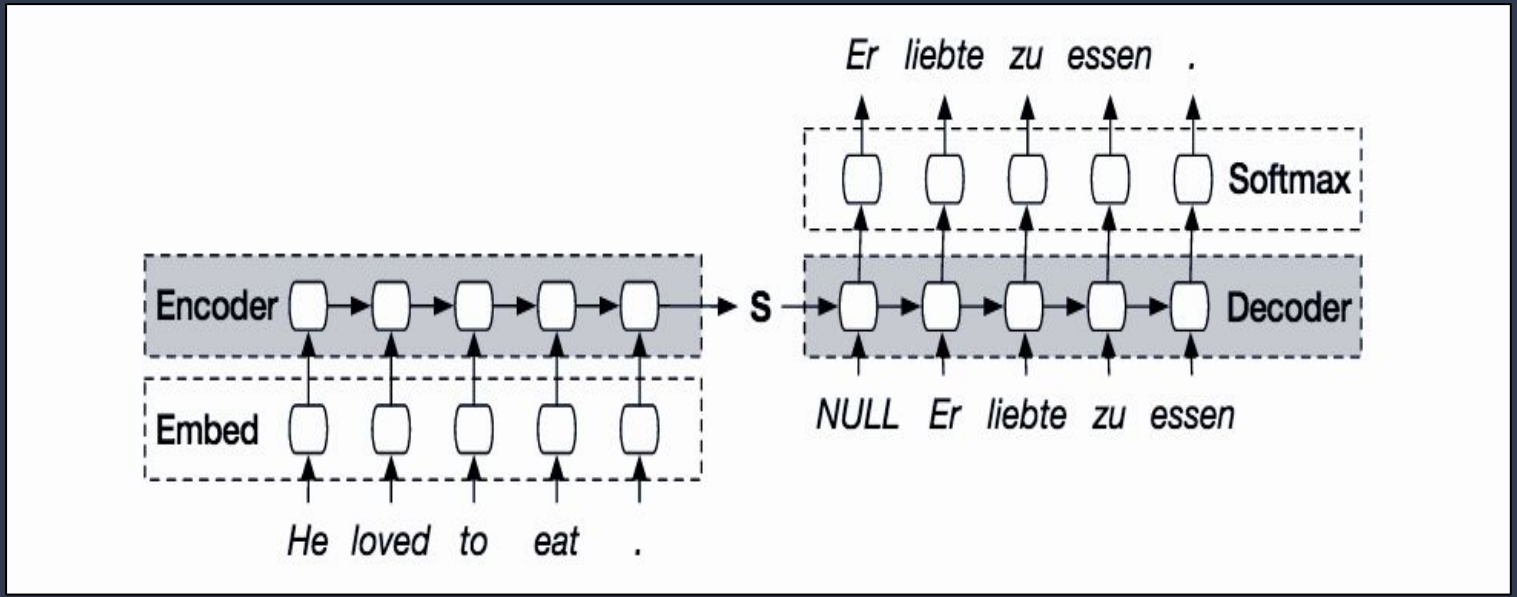
# Seq2seq Model : A High Level Overview (Continued)




Now since we have the final encoded vector, we start decoding it to get the output/target sentence. At each time-step, the input to the decoder RNN is the embedding of current word and the hidden state from the previous timestamp, which is the initial decoder hidden state i.e. the initial decoder hidden state is the final encoder hidden state. In the decoder, we go from the hidden state outputted from the decoder to an actual word, therefore at each timestamp we predict (by passing the hidden state vector through a Linear layer) the next word in the sequence.

Once predicted target sentence is predicted, it is compared against the actual target sentence, to calculate the loss.

Further this loss is used to update all of the parameters in the model.

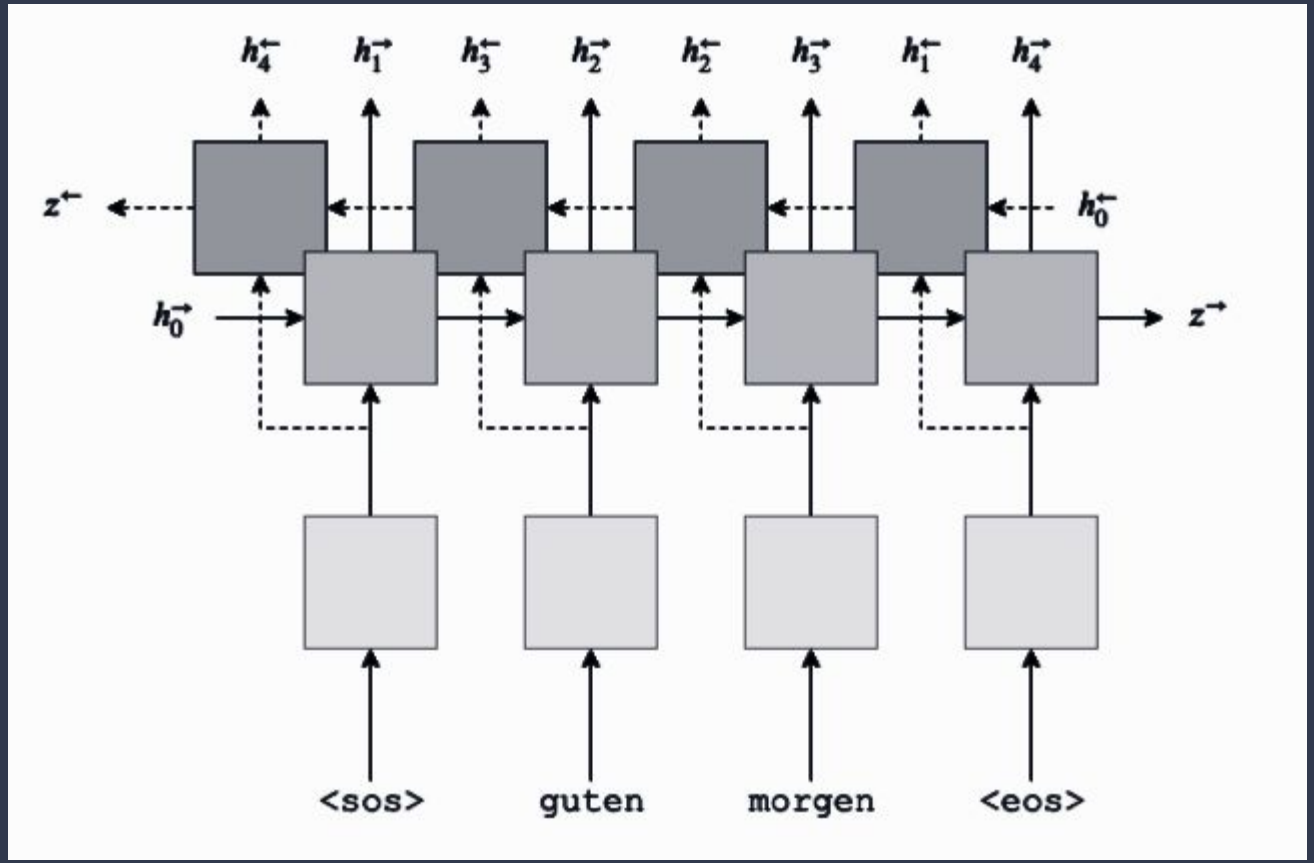

Er liebte zu essen .
Softmax
Encoder
S
Decoder
Embed
NULL Er liebte zu essen
He loved to eat .

# Encoder

With a bidirectional RNN, we have two RNNs in each layer. A forward RNN going over the embedded sentence from left to right (shown below in green), and a backward RNN going over the embedded sentence from right to left (teal).

We pass embedded input to encoder RNN , which initializes both the forward and backward initial hidden states to a tensor of all zeros. We'll also get two context vectors, one from the forward RNN after it has seen the final word in the sentence, and one from the backward RNN after it has seen the first word in the sentence .

As the decoder is not bidirectional, it only needs a single context vector, $z$ , to use as its initial hidden state, $s0$ , and we currently have two, a forward and a backward one. We solve this by concatenating the two context vectors together.




h(t)(forward) = Encoder(e(x(t)(forward)), h(t-1)(forward))

h(t)(backward) = Encoder(e(x(t)(backward)), h(t-1)(backward))

# Attention Mechanism

Attention essentially means which words in the source sentence we should pay the most attention to in order to correctly predict the next word to decode. Attention mechanism nullifies the need to encode the full source sentence into a fixed-length vector. Rather, at each step of the output generation the decoder “attends” to different parts of the source sentence. So basically, based on the input sentence and what it has produced so far, we let the model learn what to attend to.

# Attention Mechanism (Continued)

The important part is that each decoder output word now not depends on only the last state but on a weighted combination of all the input states. Some weight value is associated with each input word. The weights define how much of each input state should be considered for each output. Let the weight be denoted as **W(output_word_index, input_word_index).** So, if W(3,2) has a large value, it would mean that the decoder pays a lot of attention to the second state in the source sentence while producing the third word of the target sentence.

**Note :** The weights are typically normalized to sum to 1 (so they are a distribution over the input states).

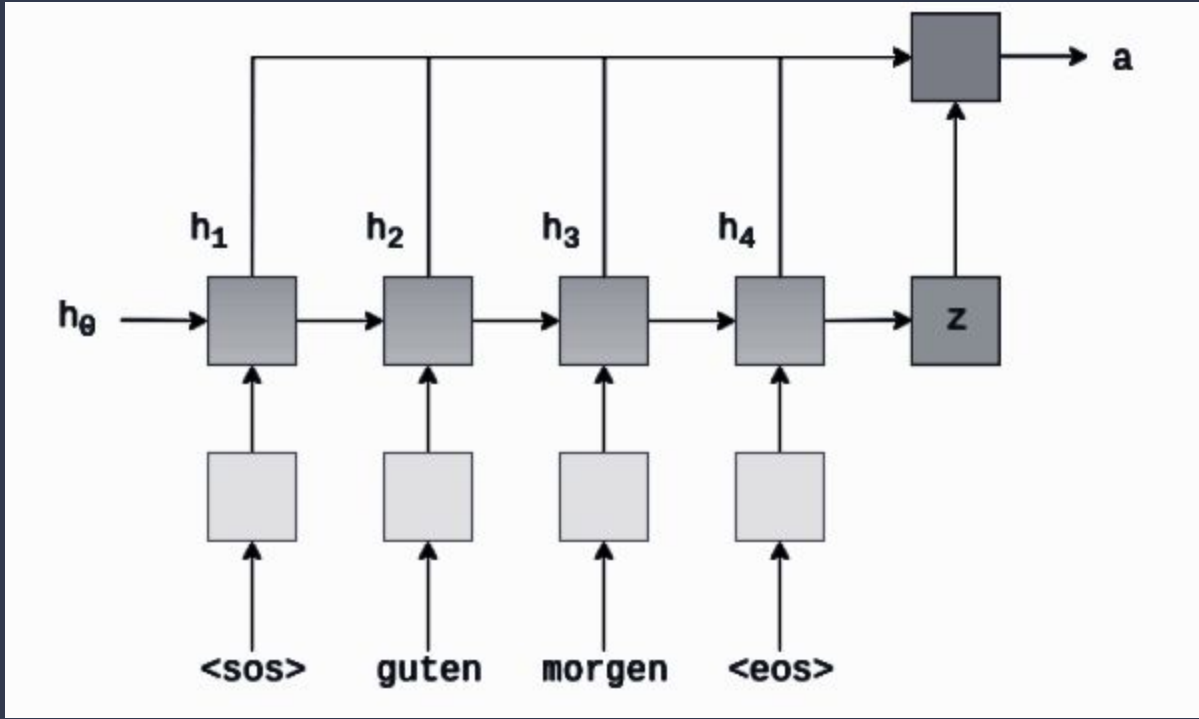
# Attention Mechanism (Continued)

The attention layer will take in the previous hidden state of the decoder, and all of the stacked forward and backward hidden states from the encoder. The layer will output an attention vector,

Intuitively, this layer takes what we have decoded so far, , and all of what we have encoded, to produce a vector, that represents which words in the source sentence we should pay the most attention to in order to correctly predict the next word to decode.

We first calculate the energy between the encoder states and the previous dcode state which tells how much each encoder hidden state "matches" the previous decoder hidden state.

We then use the Energy vector calculated to find the attention vector by multiplying it with a weights vector. These weights tell us how much we should attend to each token in the source sequence.
Finally, we ensure the attention vector fits the constraints of having all elements between 0 and 1 and the vector summing to 1 by passing it through a softmax layer.


a
h1
h2
h3
h4
h0
z
<sos>
guten
morgen
<eos>

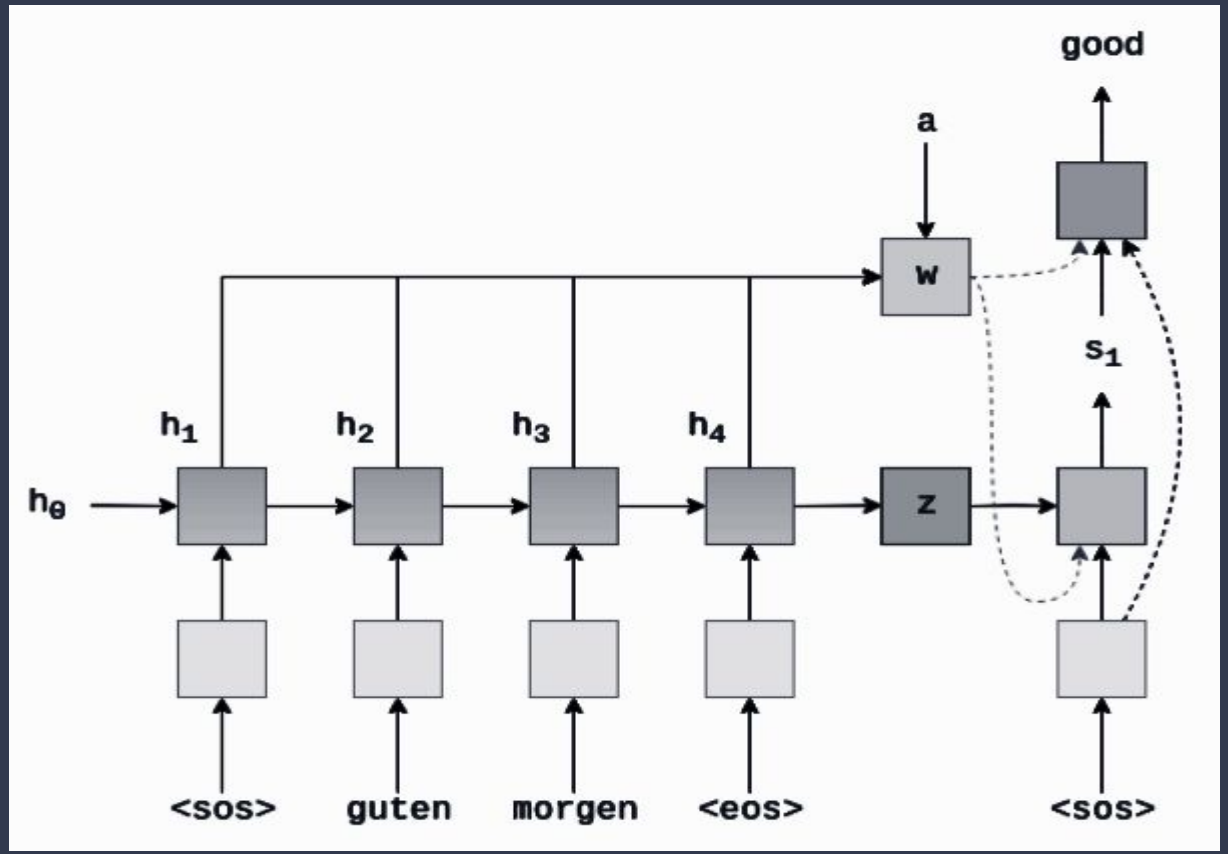
# Decoder

Next up is the decoder. The decoder contains the attention layer, attention, which takes the previous hidden state, all of the encoder hidden states, $H$, and returns the attention vector .

We then use this attention vector to create a weighted source vector, $wt$ , denoted by weighted, which is a weighted sum of the encoder hidden states, $H$ , using $at$ as the weights.

The embedded input word, the weighted source vector, and the previous decoder hidden state are then all passed into the decoder RNN, with d(yt) and wt being concatenated together.

Finally, the prediction of the next word is made.




$s(t) = Decoder(e(y(t)),w(t), s(t-1))$

$y = f(e(y(t), w(t), s(t))$

# BaseLine+

# Task 1 : LID Tags

LID stands for **Language Identification**. It means to give a language tag to all the words in the dataset i.e. in the codemix, since the words belong to Hindi and English language so all the words get a tag either of Hindi or English.

**Tool used for LID Tags :** https://github.com/irshadbhat/litcm

# Task 2 : Performance Measure

We calculated the Blue Score (BLEU (Bilingual Evaluation Understudy)) to measure the performance of our model.

The BLEU algorithm compares consecutive phrases of the automatic translation with the consecutive phrases it finds in the reference translation, and counts the number of matches, in a weighted fashion.

A higher match degree indicates a higher degree of similarity with the reference translation, and higher score.

# Resources/References

**Resources :**

1. https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html
2. https://colah.github.io/posts/2015-08-Understanding-LSTMs/

**Research Papers :**

1. https://arxiv.org/pdf/1409.3215.pdf
2. https://arxiv.org/pdf/1409.0473.pdf
3. https://arxiv.org/pdf/1706.03762.pdf

# Thank You.