Code Mix Generation

Project Outline

Team 17 - Qwertyuiop

Professor : Manish Shrivastava
Mentor : Prashant Kodali

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

Problem Statement and 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

Datasets

1. Parallel corpora for code mix and monolingual.
2. Tools for LID and normalisation.

BaseLine Model

So far for our baseline model, we are thinking of using seq-to-seq model with attention to generate a model similar to machine translation models to convert a monolingual sentence to codemix sentence and vice-versa.

Task : Building the seq-to-seq model with Attention

BaseLine Explanation

Building the seq-to-seq 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. 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.

Let’s talk about the seq-to-seq model architecture briefly :

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 **LSTM (Long Short Term Memory)** 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.

Building the seq-to-seq Model - A high level overview (Continued)

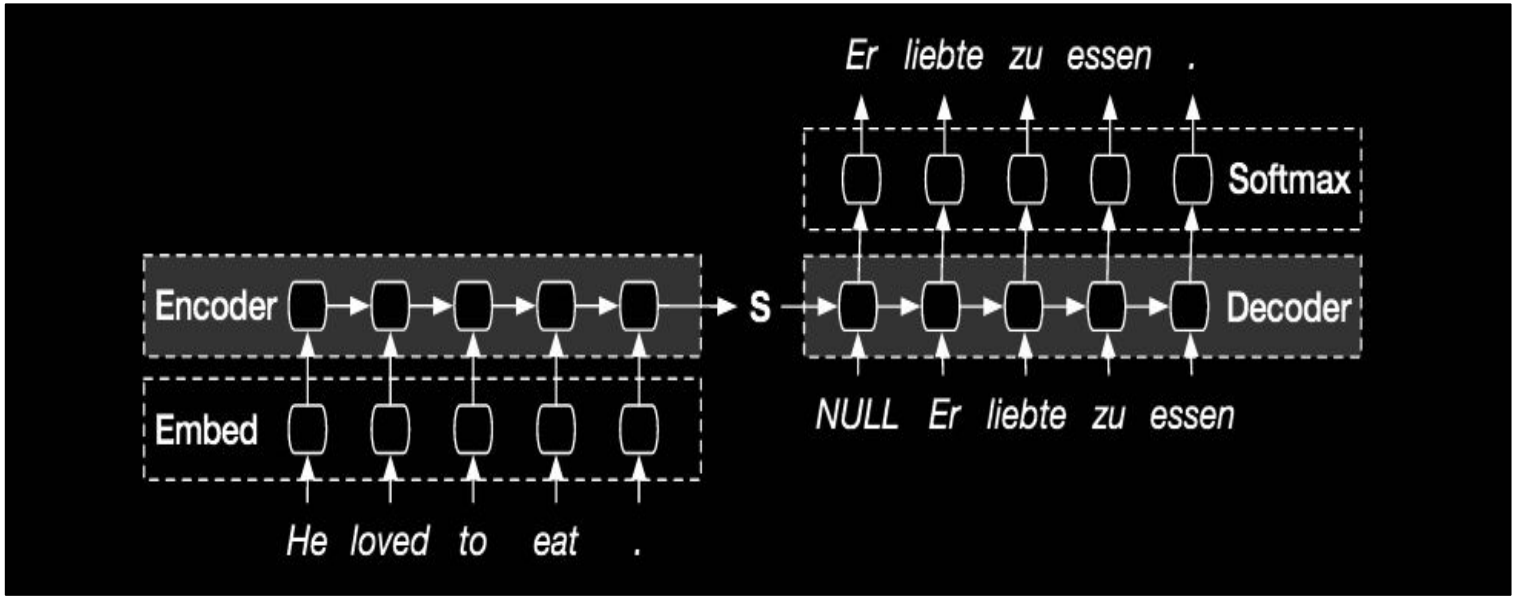
Now since we have the final encoded vector, we start decoding it to get the output/target sentence. At each time-step, the input to the decoder RNN (blue) is the embedding of current word and the hidden state from the previous timestamp, which is the initial decoder hidden state i.e. the initial decoder 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.

Note 1 : We have both, the input/source embedding layer and the output/target embedding layer, but actually they are two different embedding layers with their own parameters.

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.

Note 2 : The encoders and decoders are **multi-layer RNNs (LSTM).** But for the understanding point of view, we have explained it as a single layer concept. In the multi-layer RNN, the input sentence after being embedded goes into the first (bottom) layer of the RNN and then the hidden state and output by this layer are used as inputs to the RNN in the above layer and so on. Using multi-layer RNN means each layer in RNN outputting an output vector in the final hidden state.

Pictorial Representation of seq-to-seq Model



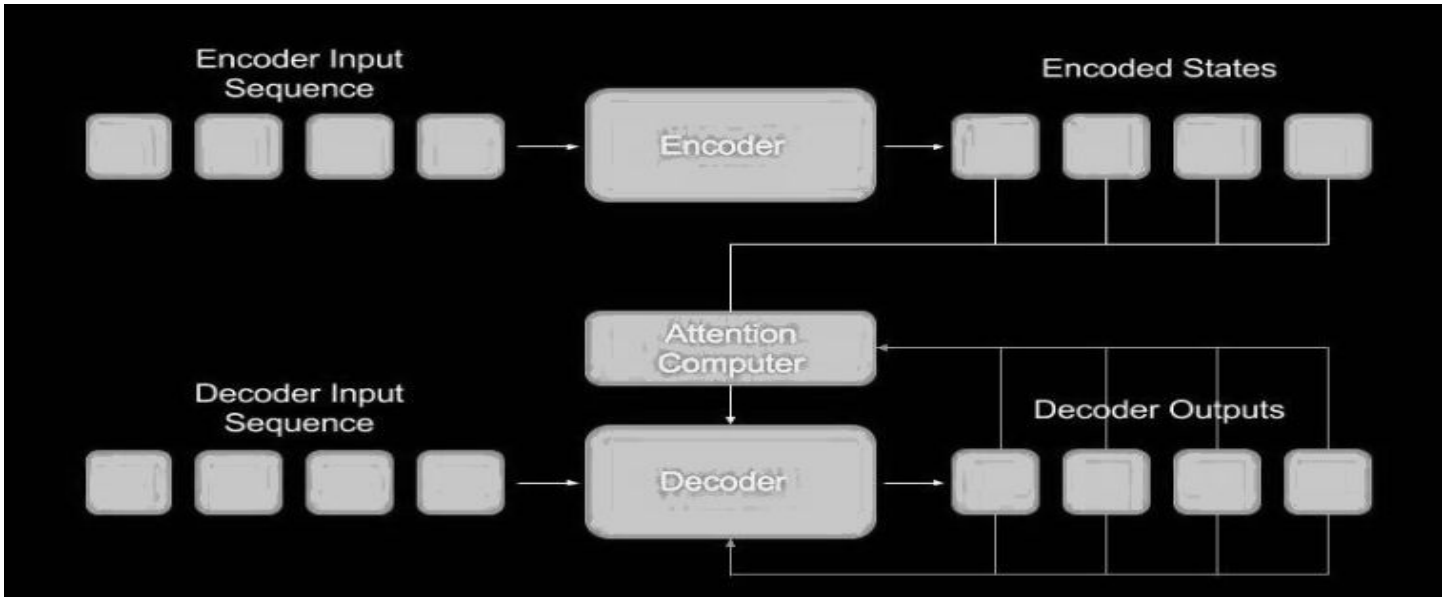
Adding Attention to seq-to-seq Model

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.

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).

Pictorial Representation of seq-to-seq Model with Attention



BaseLine +

We'll try to explore one of the following for the baseline + once we are done with the baseline.

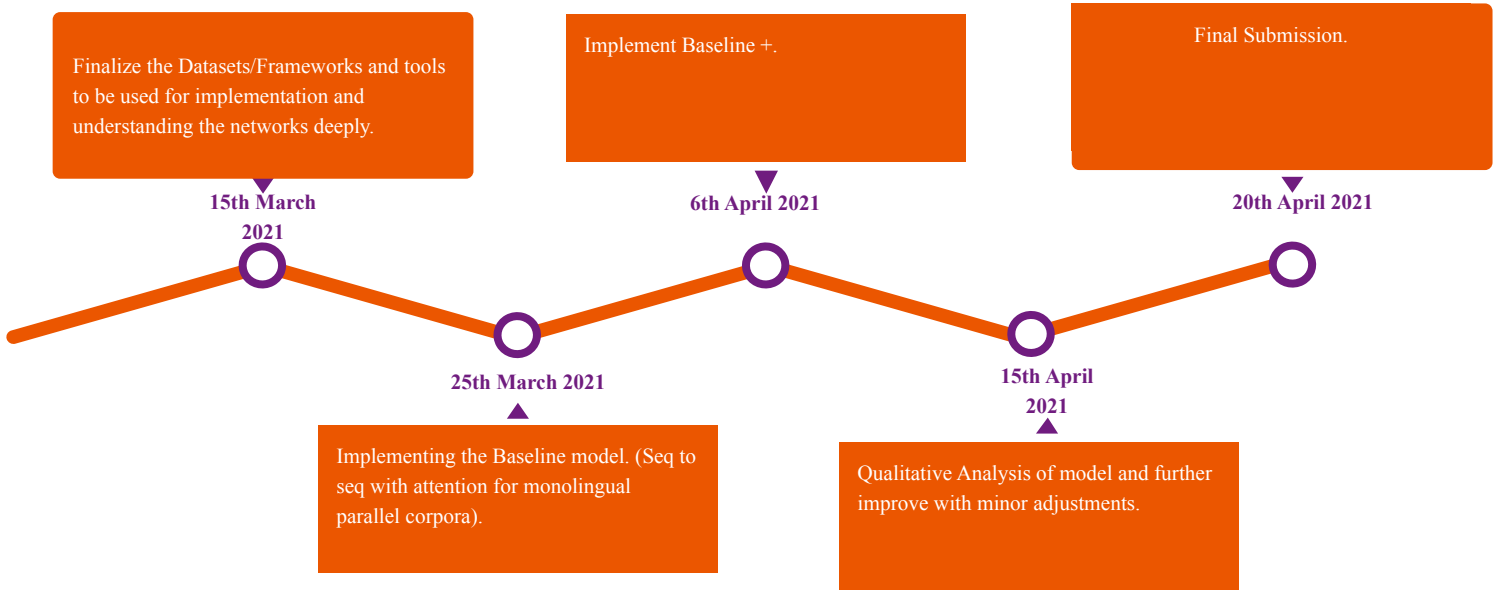
1. Working with subword embeddings
2. Working with multilingual embeddings.

We'll also try to explore :

1. Paying attention over LID tags to enforce token generation from that particular language.

We will further look into it as the project and our understanding progresses.

Timeline (Tentative)



References/Resources

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.

Link to the Project Outline Video

https://drive.google.com/file/d/1zQk_KOT_FW-Xwkkz_4cac_tuh05hNiA-/view?usp=sharing