Introduction to NLP
Codemix Generation

Team Name - QuarantinedAgain
Submitted By - Nikunj Nawal(2018111011) and Manas Kabre(2018111014)

Objective:

The aim of this project is to convert a Monolingual(English) sentence to Codemix (English and Hindi or Hinglish) and vice-versa using a Neural Machine Translation (NMT) system. For Example,

You will go to your Nani ka ghar. ⇐⇒ You will go to your grandmother's house.

The right implication implies Codemix (Hinglish) to Monolingual(English) generation or translation, and the left implication implies Monolingual(English) to codemix generation or translation.

Baseline Model:

The baseline model will use an approach similar to that proposed in the paper titled "Sequence to Sequence Learning with Neural Networks". The authors presented a seq2seq learning approach using an encoder and a decoder for a machine translation task.

Baseline + Model:

We tried and explored working with subword embeddings when we got the baseline working.

Baseline Model Approach Explanation:

A sequence to sequence model is that which takes a sequence of items (words, letters, features of an image etc.) and outputs another sequence of items.

A typical sequence to sequence model has 2 major components,

1. Encoder
2. Decoder.

The encoder processes each item in the input sequence, it compiles the information it captures into a vector (called the context). After processing the entire input sequence, the encoder sends the context over to the decoder, which begins producing the output sequence item by item. Both these parts are essentially two different recurrent neural networks (RNN) models combined into one giant network.

1. Text pre-processing-

The data we work with is more often than not unstructured so there are certain things we need to take care of before jumping to the model building part.

a. Tokenization

- We made use of Spacy Tokeniser.
- We used an Monolingual(English) tokenizer (python -m spacy download en_core_web_sm) for both Monolingual(English) and Codemix sentences.
- Next, we create the tokenizer functions. These can be passed to torchtext and will take in the sentence as a string and return the sentence as a list of tokens.
- In one case, SOURCE is Monolingual(English) and TARGET is CodeMix. In the other model, SOURCE is CodeMix and TARGET is Monolingual(English).
- The torchtext's Field (which handles how the data should be processed) also appends the "start of sequence" and "end of sequence" tokens via the init_token and eos_token arguments and converts all words to lowercase.

b. Loading the data

- The data is divided into Train, Validation, and Test data.

c. Building the Vocabulary

- Vocabulary is built for the SOURCE language and TARGET language.

2. Encoder-

The preference was to use a 4-layer LSTM but in the interest of training time we cut it down to 2-layers.

The input sentence, X, after being embedded goes into the first (bottom) layer of the RNN and hidden states, and the outputs by this layer are used as inputs to the RNN in the layer above.

Without going into too much detail about LSTMs, all we need to know is that they're a type of RNN which instead of just taking in a hidden state and returning a new hidden state per time-step, also take in and return a cell state, per time-step.

We can just think of the cell state as another type of hidden state. Similar to the initial hidden state, the initial cell state will be initialized to a tensor of all zeros. Also, our context vector will now be both the final hidden state and the final cell state.

Note: Only our hidden state from the first layer is passed as input to the second.

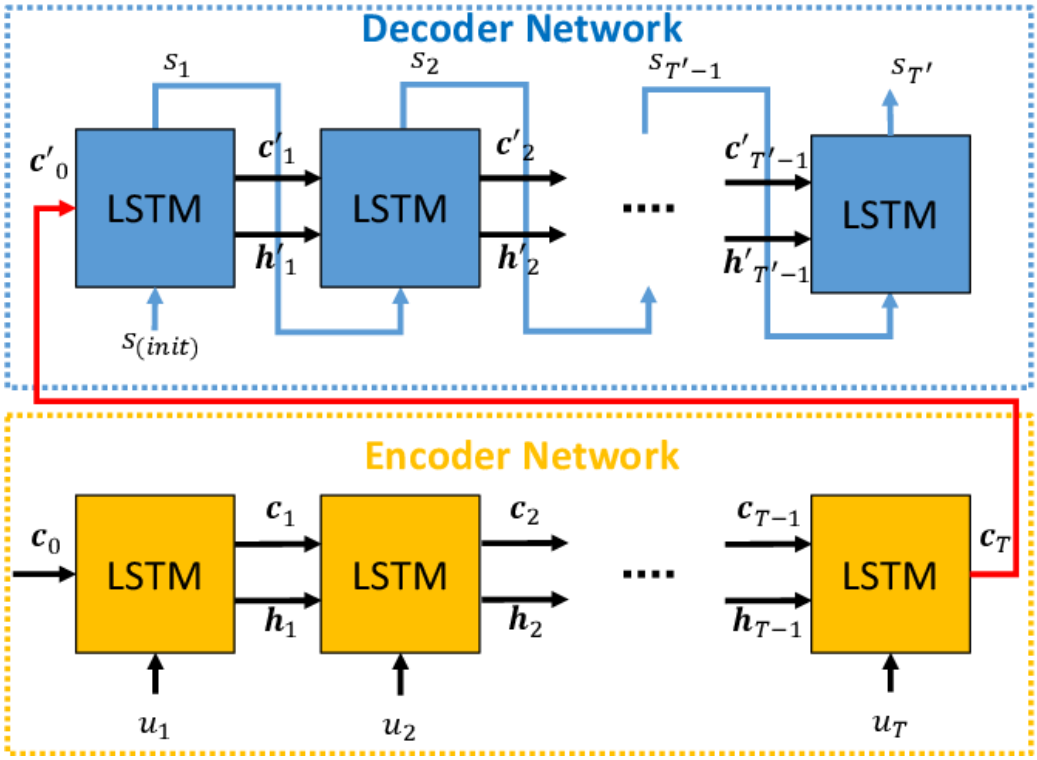
3. Decoder-

The preference was to use a 4-layer LSTM but in the interest of training time we cut it down to 2-layers.

The Decoder class does a single step of decoding, i.e. it outputs a single token per time-step. The first layer will receive a hidden state and cell state from the previous time-step, and feed it through the LSTM with the current embedded token, to produce a new hidden cell state. The subsequent layers will use the hidden state from the layer below, and the previous hidden and cell states from their layer. This provides equations very similar to those in the encoder.

Same as in the case of the encoder, only our hidden state from the first layer is passed as input to the second layer, and not the cell state.

The initial hidden and cell states to our decoder are our context vectors, which are the final hidden and cell states of our encoder from the same layer. We then pass the hidden state from the top layer of the RNN, through a linear layer, to make a prediction of what the next token in the target (output) sequence should be. layer, and not the cell state.



4. Training of Sequence 2 Sequence Model-

After initializing the parameters, the encoder and the decoder, we generate the sequence to sequence model. Then we chose an optimizer and initialised a loss function. The loss taken is $e^{actualLoss}$ to get a better analysis of the models because the loss changed every minute after epoch in the later part of the training and hence it was infeasible to compare without taking its exponent.

Subword Embeddings (For Baseline + Model):

The idea behind subword embedding is to make changes at the tokenization level so that large words can be broken down into smaller words and these words are used for training. Based on the subwords generated as output, these are combined again to form full words and eventually sentences.
We have implemented a standard algorithm at the tokenization level to tokenize the sentence into words and these words intro subwords. Initially, we have made subwords of length 3 and later we have made subwords of length 2. A concept similar to that of bigrams and trigrams is used to generate the 2/3 length subwords.

The algorithm we have used is Byte Pair Encoding[1]. It is a simple data compression algorithm that is extensively used today, even in popular NLP models like BERT. This algorithm is slightly modified for the purpose of the subword pairs.
The algorithm finds pairwise maximum frequency occurrences and adds them into the vocabulary step by step thereby following a bottom-up approach and creating subwords for the given corpus.
If the sentence is = “Jason is playing cricket”
Step 1:
[“J a s o n </w>”, “i s </w>”, “p l a y i n g </w>”, “c r i c k e t </w>”]
Every word is now converted into a charter and pairwise they are combined based on maximum occurrences and a threshold value given for the number of maximum combinations. A possibility could be Jason become Ja so n</w>.

Although we did not get better scores for some reason, we were able to implement the same and see the sentences and words getting broken down into subwords.

3. Checkpoints and the Metric-

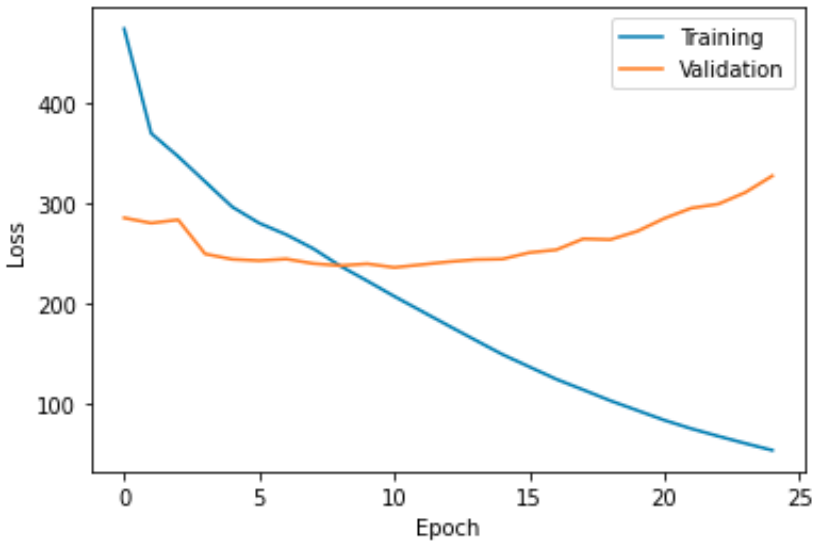
The best model after running 25 epochs was stored in a “.pt” file.

BLEU score was the metric value used. The Bilingual Evaluation Understudy Score (BLEU) is a metric for evaluating a generated sentence to a reference sentence. It gives output between 0 and 1 (both inclusive). Higher the BLEU score better are the results. The reasons to use the BLEU score as a metric value are it is language-independent, quick and inexpensive to calculate and it correlates highly with human evaluation.

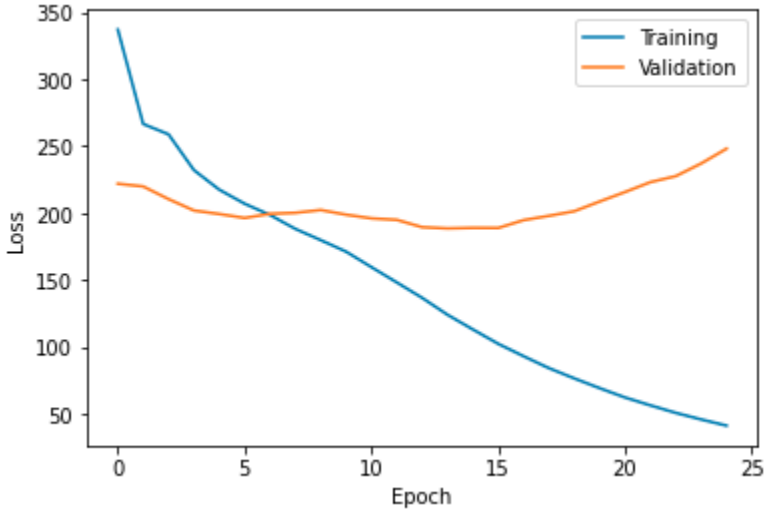
Analysis:

Model	BLEU (score)
Monolingual(English) to Codemix	0.24228
Codemix to Monolingual(English)	0.35248
Monolingual(English) to Codemix with subword	0.20127
Codemix to Monolingual(English) with subword	0.27993

1 "Byte Pair Encoding — The Dark Horse of Modern NLP | by ...." 22 Nov. 2019, https://towardsdatascience.com/byte-pair-encoding-the-dark-horse-of-modern-nlp-eb36c7df4f10. Accessed 1 May. 2021.



Monolingual(English) to Codemix



Codemix to Monolingual(English)

Observations:

- As we can observe from the curve above, the blue line (representing training loss) is constantly decreasing as we perform epochs, thus implying that the training is happening rightly.
- Validation error initially decreases slightly and then increases (we have taken the best model from the available models we created at every epoch).

- Ideally using subwords we expected an increase in the blue score, however, subwords were not able to capture the semantic relations between the two languages properly.
- Sub words generated looked well structured, however, performance was not as expected.

Challenges Faced:

- Initially understanding the subword algorithm was tedious. We misunderstood it initially however were able to understand the same later and implement the same.
- Generating the right N factor to get subwords was kinda challenging, tuning this hyperparameter took a while
- Training the models on our machines was a difficult task as it took very long to run 20-30 epochs.
- Working with the dataset and understanding it was slightly challenging initially, but eventually, we were able to use and work with it.

Improvement Scope:

- If we are able to get a bigger data set, I believe our model would perform better.
- At the subword Level we have used BPE, we were able to do our implementation, however, I feel there can be an improved one if we just don't do a pairwise combination and further use triplets, etc.
- Given a set of data where we are getting too many unknown tokens, we can set a threshold for the same and then use statistical models to predict that token using the nearby context. The other solution can be considering that model has not learned the parameters yet and needs more epochs of training, so we can further train the model
- We were limited by our resources and could not train on many epochs, however, with the right resources, we will be able to train the model better.