Code Mix Generation

Interim Report

Team 17 - Qwertyuiop

Professor : Manish Shrivastava
Mentor : Prashant Kodali

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

Tasks Done

Tasks Done

- Read 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

- Using pytorch we implemented an encoder and decoder for the seq to seq model.
- Now we will present a basic summary of what we have done so far .

About the Dataset

Dataset used : https://ritual.uh.edu/lince/datasets (**English -> Hindi-English(codemix)**)

The dataset in general contains sentences about movies and its characters.

Source sequence : **English**

Target sequence : **Codemix**

We used a BucketIterator instead of the standard Iterator as it creates batches in such a way that it minimizes the amount of padding in both the source and target sentences.

Part 1 : 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 .

It is important to note that our vocabulary has only be built from the training set and not the validation/test set. This prevents "information leakage" into our model, giving us artificially inflated validation/test scores

Vocabulary found : **English: 3262, Codemix : 4138**

Task 2 : Building Translator Model

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.)

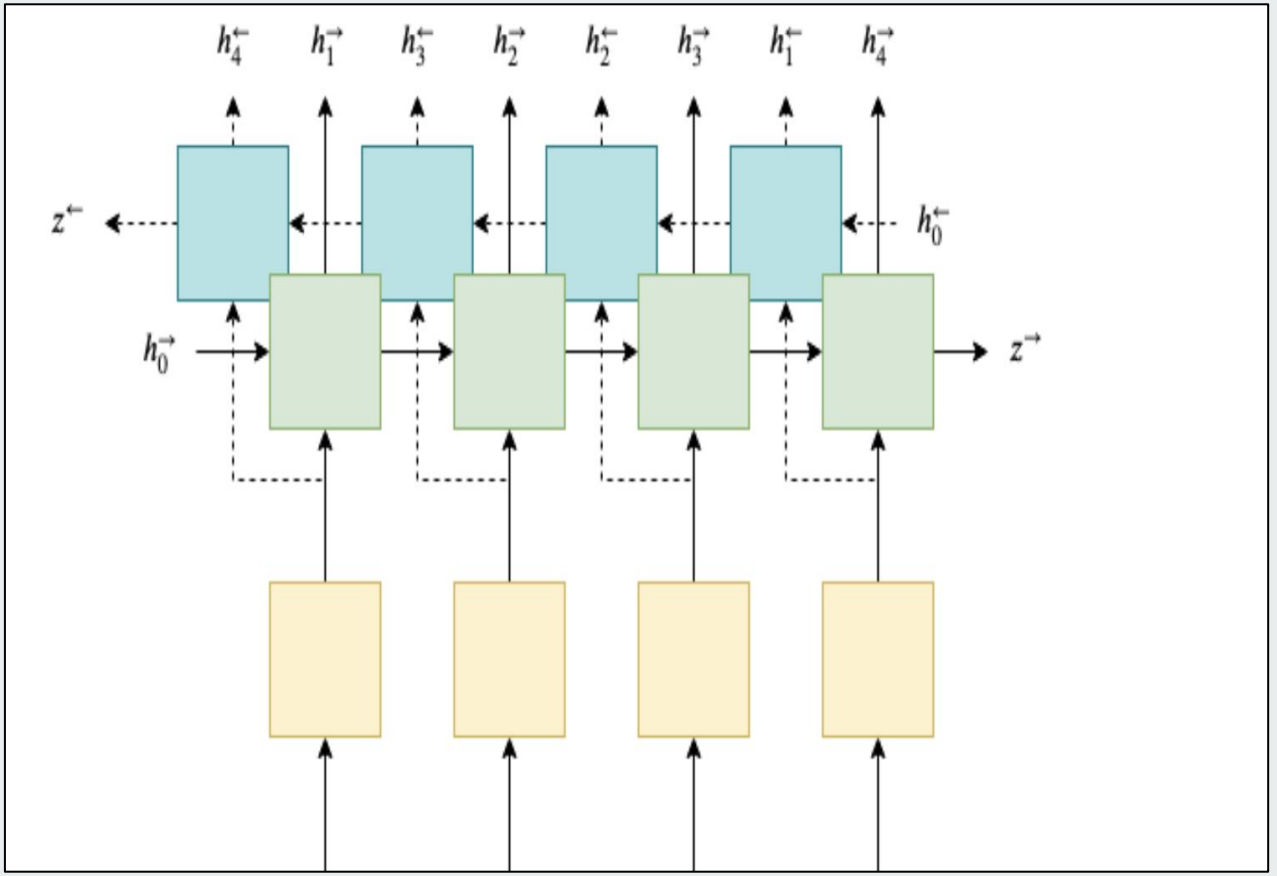
Encoder

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 tells PyTorch to initialize 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, passing them through a linear layer, g , and applying the tanh activation function.

$$z = \tanh(g(h_T^{\rightarrow}, h_T^{\leftarrow})) = \tanh(g(z^{\rightarrow}, z^{\leftarrow})) = s_0$$


$h_4^{\leftarrow}$ $h_1^{\rightarrow}$ $h_3^{\leftarrow}$ $h_2^{\rightarrow}$ $h_2^{\leftarrow}$ $h_3^{\rightarrow}$ $h_1^{\leftarrow}$ $h_4^{\rightarrow}$
$z^{\leftarrow}$
$h_0^{\leftarrow}$
$h_0^{\rightarrow}$
$z^{\rightarrow}$

Encoder

As we want our model to look back over the whole of the source sentence we return outputs, the stacked forward and backward hidden states for every token in the source sentence. We also return hidden, which acts as our initial hidden state in the decoder.

Attention

Attention works by first, calculating an attention vector, a , that is the length of the source sentence. The attention vector has the property that each element is between 0 and 1, and the entire vector sums to 1. We then calculate a weighted sum of our source sentence hidden states, H , to get a weighted source vector, w .

Next up is the attention layer. This will take in the previous hidden state of the decoder, $st-1$, and all of the stacked forward and backward hidden states from the encoder, H . The layer will output an attention vector, at , that is the length of the source sentence, each element is between 0 and 1 and the entire vector sums to 1.

Intuitively, this layer takes what we have decoded so far, $st-1$, and all of what we have encoded, H , to produce a vector, at , 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 .

Attention (Continued)

First, we calculate the energy between the previous decoder hidden state and the encoder hidden states. As our encoder hidden states are a sequence of T tensors, and our previous decoder hidden state is a single tensor, the first thing we do is repeat the previous decoder hidden state T times. We then calculate the energy, Et , between them by concatenating them together and passing them through a linear layer (attention) and a $\tanh$ activation function.

This can be thought of as calculating how well each encoder hidden state "matches" the previous decoder hidden state.

$$E_t = \tanh(\text{attn}(s_{t-1}, H))$$

Decoder

Next up is the decoder. The decoder contains the attention layer, attention, which takes the previous hidden state, $st-1$, all of the encoder hidden states, H , and returns the attention vector, at .

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.

Seq2seq

Briefly going over all of the steps:

1. the outputs tensor is created to hold all predictions, Y
2. the source sequence, X, is fed into the encoder to receive z and H
3. the initial decoder hidden state is set to be the context vector, $s0=z=hT$
4. we use a batch of <sos> tokens as the first input, $y1$
5. we then decode within a loop:
    a. inserting the input token yt, previous hidden state, $st-1$, and all encoder outputs, H, into the decoder
    b. receiving a prediction, $\hat{y}\,t+1$, and a new hidden state, st
    c. we then decide if we are going to teacher force or not, setting the next input as appropriate.

Modified Timeline

- **1st April : Test and completely finish the baseline model**
- **9th April : Implementation of baseline +**
- **15th April : Do qualitative analysis on results and do minor improvements. Calculate Bleu Scores and improve the model.**
- **20th April : Final Submission.**

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

Link to the Interim Report Video

https://drive.google.com/file/d/1QZo6fnwc619WIK9xS8KVZIN5rGdqDJ-U/view?usp=sharing

Thank You.