AML (SPRING 2020) – COURSE PROJECT REPORT

Sequence to Sequence Mapping for Machine Translation

Prashanthi R
Rathi Kashi
May 6, 2020

1 PROBLEM DESCRIPTION AND ABSTRACT

Automatic or machine translation is perhaps one of the most challenging artificial intelligence tasks given the fluidity of human language. Classically, rule-based systems were used for this task, which were replaced in the 1990s with statistical methods. More recently, deep neural network models achieve state-of-the-art results in a field that is aptly named neural machine translation.[1] Our project deals with designing and building a sequence-to-sequence mapper to accomplish neural machine translation. We have chosen to work on a English to Hindi machine translation model. For this purpose, we have used an encoder-decoder architecture with attention.

[1]https://machinelearningmastery.com/introduction-neural-machine-translation/

2 History and Prior Approaches

Neural based machine translation models aim to learn statistical models of machine translation that work by maximizing the probability of the output sequence given the input sequence of text. These models require a large corpus of data with both source and target language text.

In the past, multilayer perceptron neural network models were used for language translation, however these models were limited by a fixed length input sequence and the output sequence had to be of the same length.

In order to overcome the issue of fixed length inputs and outputs, an encoder-decoder model which uses recurrent neural networks was proposed. An encoder neural network reads and encodes a source sentence into a fixed-length vector. A decoder then outputs a translation from the encoded vector. The whole encoder–decoder system, which consists of the encoder and the decoder for a language pair, is jointly trained to maximize the probability of a correct translation given a source sentence.[2]

Recurrent Neural Network is a generalization of feedforward neural network that has an internal memory. RNN is recurrent in nature as it performs the same function for every input of data while the output of the current input depends on the past one computation. After producing the output, it is copied and sent back into the recurrent network. For making a decision, it considers the current input and the output that it has learned from the previous input. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs.[3] LSTM and GRUs are modified versions of RNNs, that address the vanishing gradient problem and thus are used in place or RNNs in the encoder-decoder network.

The encoder-decoder model is effective for short sequences of data, however, for longer sequences, this model fails since the fixed size context vector created after encoding is not able to capture all the semantic details of a very long sentence. A more efficient approach, however, is to read the whole sentence or paragraph, then to produce the translated words one at a time, each time focusing on a different part of the input sentence to gather the semantic details required to produce the next output word. [4] In the words of Bahdanau et. al, "[e]ach time the proposed model generates a word in a translation, it (soft-)searches for a set of positions in a source sentence where the most relevant information is concentrated. The model then predicts a target word based on the context vectors associated with these source positions and all the previous generated target words." Thus, we adopt this approach for our problem.

[2]Neural Machine Translation by Jointly Learning to Align and Translate, 2014.

[3]https://towardsdatascience.com/understanding-rnn-and-lstm-f7cdf6dfc14e

[4]Page 462, Deep Learning, 2016.

3 Adopted Approach

Our model uses an encoder-decoder with attention in order to translate sentences.

Attention is the idea of freeing the encoder-decoder architecture from the fixed-length internal representation. This is achieved by keeping the intermediate outputs from the encoder GRU from each step of the input sequence and training the model to learn to pay selective attention to these inputs and relate them to items in the output sequence.Thus, each item in the output sequence is conditional on selective items in the input sequence. The encoder-decoder recurrent neural network architecture with attention is currently the state-of-the-art on some benchmark problems for machine translation. [5]

3.1 Our Encoder-Decoder Model

A neural machine translation system is a neural network that directly models the conditional probability $p(y|x)$ of translating a source sentence, $x_1, ..., x_n$, to a target sentence, $y_1, ..., y_m$. A basic form of NMT consists of two components: [6]

(a) an encoder which computes a representation s for each source sentence

(b) a decoder which generates one target word at a time and hence decomposes the conditional probability as:

$$\log p(y|x) = \Sigma_{j=1}^{m} \log p(y_j|y_{<j}, s) \tag{3.1}$$

We use Gated Recurrent Units for the encoder and decoder. The probability of decoding each word y_j as:

$$p(y_j|y_{<j}, s) = softmax(g(h_j)) \tag{3.2}$$

with g being the transformation function that outputs a vocabulary-sized vector. Here, hj is the RNN hidden unit, abstractly computed as:

$$h_j = f(h_{j-1}, s) \tag{3.3}$$

where f computes the current hidden state given the previous hidden state and is a GRU unit. The source representation, s, implies a set of source hidden states which are consulted throughout the entire course of the translation process.

The loss is calculated as follows (this is the function used for training):

$$J_t = \Sigma_{(x,y)\in D} \log p(y|x) \tag{3.4}$$

where D is the training corpus. We have employed the batch training algorithm in order to train the corpus.

[5] https://machinelearningmastery.com/attention-long-short-term-memory-recurrent-neural-networks/

[6] Effective Approaches to Attention-based Neural Machine Translation: Minh-Thang Luong, Hieu Pham and Christopher D. Manning

Output generation is done through a greedy 1-best search process, where p is calculated at every time step and the word that gives the highest probability is chosen as the next word in the sequence.

$$\hat{y} = \underset{\text{y}}{argmax}\ P(y|x) \tag{3.5}$$

3.2 Adding Attention

We used the Bahdanau attention model, where at each time step t, the model infers a variable-length alignment weight vector at based on the current target state h_t and all source states $\bar{h}_s$. A global context vector c_t is then computed as the weighted average, according to a_t, over all the source states.

Given the target hidden state h_t and the source-side context vector c_t, we employ a concatenation layer to combine the information from both vectors to produce an attentional hidden state as follows:

$$\bar{h}_t = \tanh(W_c[c_t; h_t]) \tag{3.6}$$

The attentional vector $\bar{h}_t$ is then fed through the softmax layer to produce the predictive distribution formulated as:

$$p(y_t|y_{<t}, x) = softmax(W_s\bar{h}_t) \tag{3.7}$$

The idea of a global attentional model is to consider all the hidden states of the encoder when deriving the context vector c_t. In this model type, a variable-length alignment vector a_t, whose size equals the number of time steps on the source side, is derived by comparing the current target hidden state h_t with each source hidden state $\bar{h}_t$:

$$\begin{aligned} a_t(s) &= align(h_t, \bar{h}_s) \\ &= \frac{exp(score(h_t, \bar{h}_s)}{\Sigma_s exp(score(h_t, \bar{h}_s))} \end{aligned} \tag{3.8}$$

The score is calculated as follows:

$$score(h_t, \bar{h}_s) = v_a^\top tanh(W_a[h_t; \bar{h}_s]) \tag{3.9}$$

The context vector is calculated as follows:

$$c_t = \Sigma_s a_t(s)\bar{h}_s \tag{3.10}$$

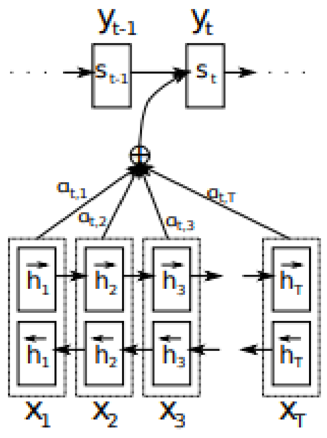


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \dots, x_T)$.

3.3 Drawback of an attention based model

Although effective, the neural machine translation systems still suffer some issues, such as scaling to larger vocabularies of words and the slow speed of training the models. There are the current areas of focus for large production neural translation systems, such as the Google system.

4 Implementation Details

The model we have used for this sequence to sequence mapping is an encoder-decoder model with attention. The model consists of two RNN / LSTM / GRU networks - an encoder and a decoder. The input to the encoder is a sentence in the original language. However, the input to the decoder is the sentence in the target language with a start-of-sentence token (we use "<sos>" to indicate the same). The decoder also takes as input a context vector—which is an internal representation of the input sentence encoded by the encoder. The output of the decoder is a sentence with a trailing end-of-sentence token. The idea is that the encoder skips through input time steps and encodes the entire input sequence into a context vector. The decoder is supposed to skip through the output time steps and decode the context vector into a sentence in the target language.

As explained above, if the context vector is of a fixed-size, it may not store a lot of information for longer sentences or may store more information than necessary for shorter sentences. Thus, rose the need for a mechanism that helps identify how much each input word gives the context or meaning to a given sentence. This is the reason we chose to deploy the encoder-decoder model with attention. Attention allows for the context vector to have variable size based on how much of the sentence needs to be passed on to the decoder as context.

4.1 Dataset

For the purpose of this project, we use the Indic Language Multilingual Parallel Corpus built by Kyoto University and the National Institute of Information and Communications Technology, Japan. The parallel corpus for English-Hindi has 84557 sentences each for training. However, we use only 48000 of these data points for the training of our model. We use about another 1000 data points for testing purposes.

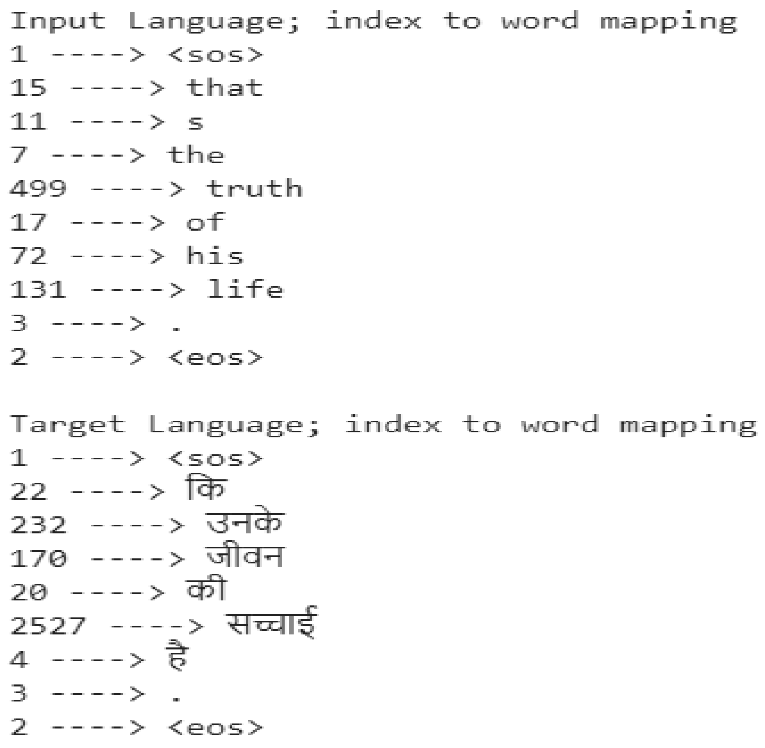
4.2 Data Pre-processing

For the both the input language and target language sentences, we had to strip the return and newline characters.

For the input English sentences, we converted them from Unicode to lowercase ASCII letters. After this we created spaces between punctuation and words. This was done so that the model does not learn, for example, "cat" and "cat." and "cat?" as three different words. Further, we added the start-of-sentence, "<sos>" and end-of-sentence "<eos>" tokens to the input sentences.

For the output sentences, we used the IndicNLP library to tokenize the Hindi words. After this, we used the Tokenizer class from Keras Text Processing to split the input sentences into tokens. We then used fit_on_texts function to store the input and output tokens as a dictionary of words with associated fixed indices for the vocabulary of unique words for both the input and the target languages. Then we obtained numeric sequences of sentences of both the input and output languages based on the indices obtained above. Since the sentences in each language differ in size, the numeric sequences had to be padded with 0s to the length of the longest sentence in each language. Thus, the English sequences were 72 in length and the Hindi sequences were 69 in length for this dataset. We have also reversed the order of the words of the input and pre-padded them like Sutskever et. al [8] talk about in their paper. In their words - "we found that reversing the order of the words in all source sentences (but not target sentences) improved the LSTM's performance markedly, because doing so introduced many short term dependencies between the source and the target sentence which made the optimization problem easier."

Below is a screenshot of how tokens of a sentence are mapped to their corresponding indices obtained from Tokenizer.fit_on_texts:



```
Input Language; index to word mapping
1 ----> <sos>
15 ----> that
11 ----> s
7 ----> the
499 ----> truth
17 ----> of
72 ----> his
131 ----> life
3 ----> .
2 ----> <eos>

Target Language; index to word mapping
1 ----> <sos>
22 ----> कि
232 ----> उनके
170 ----> जीवन
20 ----> की
2527 ----> सच्चाई
4 ----> है
3 ----> .
2 ----> <eos>
```

Further, the data points are shuffled and split into training data and testing data. The training and testing data are split further into batches of 64 data points.

4.3 Background concepts

For the purposes of this model, we have used the Bahdanau attention for the encoder. For explanation, we use the following notations:

FC = Fully connected (dense) layer
EO = Encoder output
H = hidden state
X = input to the decoder

In Bahdanau attention model, a score is computed using the encoder output (EO) and the hidden state (H). Futher, attention weights are computed using the score by passing them through a softmax layer (along the maximum length of input sentence dimension). It is computed along this axis, because a model with attention requires the weight of each input word to identify how much it contributes to the overall context or meaning of the sentence. The context vector is then computed by doing a dotproduct of each of the encoder outputs with the attention weights. The input to the decoder, X (a word from the target language, "<sos>" for the first decoder unit) is passed through an embedding layer for the target language. Both the context vector and this embedded input, X, is passed to the decoder at every stage. These can be understood mathematically as given below:

1. score = FC (tanh(FC(EO) + FC (H)))
2. attention weights = softmax(score) (along the maximum length of input dimension.)
3. context vector = dotproduct(EO, attention weights) (along the maximum length of input dimension.)
4. embedded input = embedding layer(X)
5. decoder input = [embedding input, context vector]

Useful sources: tensorflow.org [5], Bahdanau et.al's academic paper. [6]

4.4 Model

4.4.1 Encoder

Our encoder is a simple network with an embedding layer of fixed dimension 256 for the input language through which input is passed before going to a GRU layer. The GRU layer with 1024 units takes a batch of 64 input sentences and a hidden state vector of shape (Batch size, 1024) initialised with zeroes. The encoder returns an output and a hidden state.

4.4.2 Attention Layer

The attention layer essentially comprises of three fully connected dense layer of a size that can be input. As explained in the Background Concepts section, this layer computes the score, the attention weights and subsequently the context vector using the encoder output, the hidden layer and the attention weights.

4.4.3 Decoder

The decoder is also a simple network, but it is different from the encoder in that it incorporates attention during its execution. It comprises of an embedding layer of fixed dimension 256 and a GRU layer. First the encoder output and the hidden state are passed to the attention network to obtain attention weights and the context vector. The input decoder token, which is usually the start-of-sentence or in our case: the "<sos>" token, for the first decoder unit), is passed to an embedding layer for the target language. Now the context vector and this embedded decoder input token is passed through the GRU of the decoder to obtain an output and a hidden state. For subsequent layers, the decoder input token that is passed to the embedding layer is the decoder output of the previous decoder unit. The decoder network also has a final Dense Layer which is of the size of the total vocabulary of the target language. The output from this Dense layer determines which word is predicted. The index that gets the largest probability from the Dense layer is the index of the predicted word from the overall target language vocabulary.

4.5 Training

For the training process, batches of inputs are passed through the encoder to obtain encoder outputs and hidden states. These encoder outputs and hidden states are then passed through the decoder network (which passes them through the attention layer first). The decoder network then makes a prediction.

What is important to note here is that, when the model is training, the predictions might be really poor. Thus, passing one decoder unit output, presumably a poor prediction, to the next decoder unit might lead to very bizarre predictions. In order to teach the decoder, we use this idea of teacher forcing. In this, we feed the correct target output word from the original dataset to the next decoder unit instead of the predicted word during training. This helps the network learn from what the prediction should actually have been.

The loss is computed between the predicted word and the actual target output word from the dataset we are training on. This loss is the loss value for a batch. Gradient descent is performed on the encoder and decoder networks to change the trainable weights of the two networks based on the loss that has been obtained for a batch. Further, all the batches are trained for a certain number of epochs. The loss for each epoch is computed as the average of all the batch losses for that epoch. Ideally, with each epoch, the training loss reduces because of gradient descent.

4.6 Prediction

This process is exactly the same to the training process. However, instead of performing teacher forcing by feeding the correct outputs to subsequent decoder units, we actually feed the previous decoder unit's output to a decoder unit. Once we are confident about the training process, the model can perform predictions. If training has been done on enough data, the predictions can be expected to be good.

Further, since this is not training, we neither compute batch loss nor perform gradient descent based on the batch loss. To analyse the performance of our machine translation model, we use a metric called BLEU score.

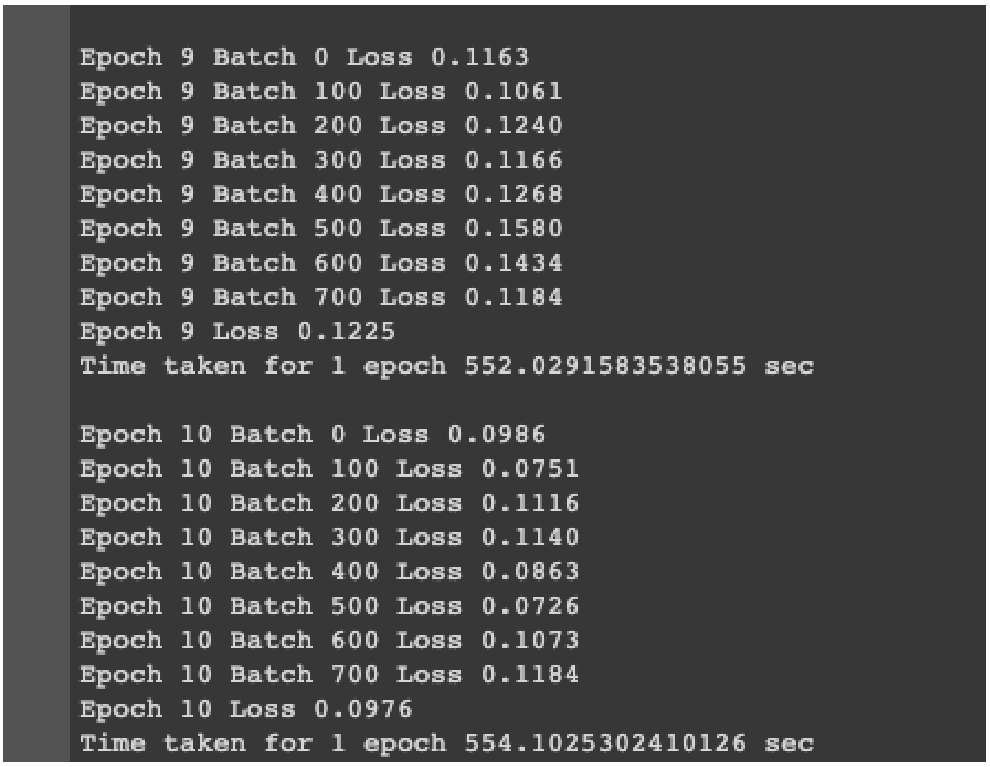
4.6.1 BLEU Score: A metric to evaluate the performance of machine translations

To put it simply, BLEU scores compare machine translations with (professional) human translations and assign the machine translation model a score. The idea is that the higher the BLEU score, the closer the translation model to a hypothetical human translation model. Most academic papers, including Bahdanau et. al and Sutskever et. al [7]that we have referred to use this metric to validate the performance of their models. In the section below, we discuss our model's results on real data.

[7]Sequence to Sequence Learning with Neural Networks - https://arxiv.org/pdf/1409.3215.pdf

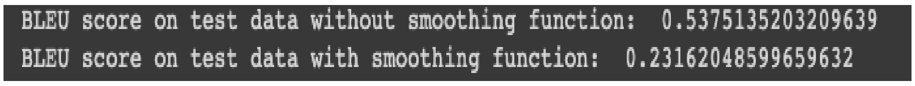
5 Results

Out of the 84000 data points in the dataset, due to computational limitations, we trained only on 48000 data points for ten epochs. The average loss for the first epoch was 0.7076. After training for ten epochs, it went down to 0.0976. The total training time was about 90 minutes.



```
Epoch 9 Batch 0 Loss 0.1163
Epoch 9 Batch 100 Loss 0.1061
Epoch 9 Batch 200 Loss 0.1240
Epoch 9 Batch 300 Loss 0.1166
Epoch 9 Batch 400 Loss 0.1268
Epoch 9 Batch 500 Loss 0.1580
Epoch 9 Batch 600 Loss 0.1434
Epoch 9 Batch 700 Loss 0.1184
Epoch 9 Loss 0.1225
Time taken for 1 epoch 552.0291583538055 sec

Epoch 10 Batch 0 Loss 0.0986
Epoch 10 Batch 100 Loss 0.0751
Epoch 10 Batch 200 Loss 0.1116
Epoch 10 Batch 300 Loss 0.1140
Epoch 10 Batch 400 Loss 0.0863
Epoch 10 Batch 500 Loss 0.0726
Epoch 10 Batch 600 Loss 0.1073
Epoch 10 Batch 700 Loss 0.1184
Epoch 10 Loss 0.0976
Time taken for 1 epoch 554.1025302410126 sec
```

The BLEU score we obtained before and after applying a smoothing function is given below:

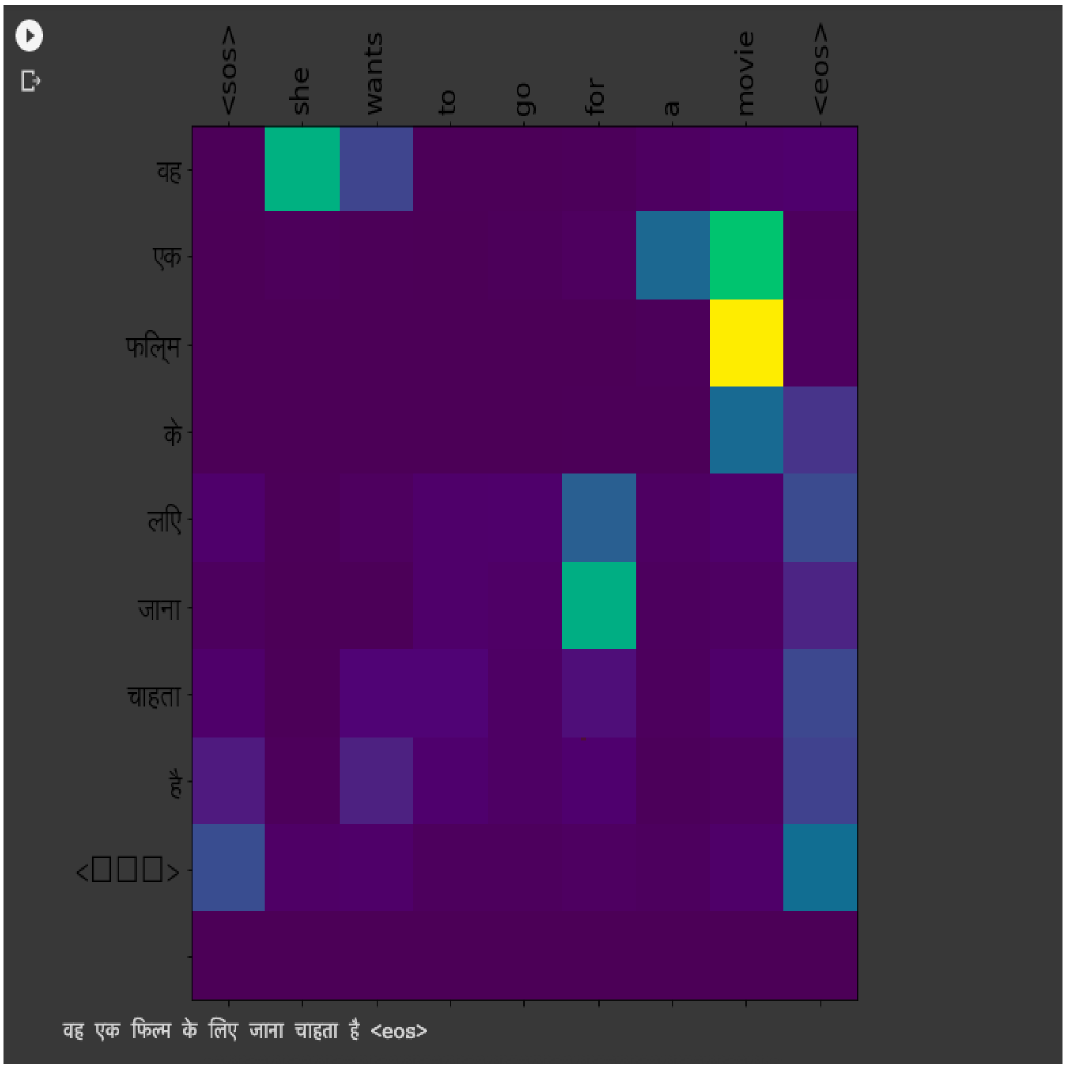


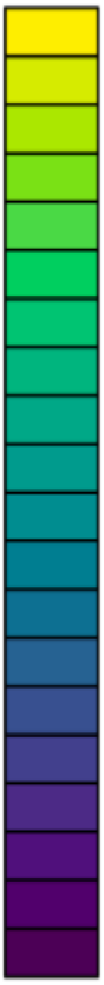
```
BLEU score on test data without smoothing function:  0.5375135203209639
BLEU score on test data with smoothing function:  0.23162048599659632
```

The reason we used smoothing functions was inspired from Chen and Cherry [8] who write - "All smoothing techniques improved sentence-level correlations over no smoothing." We can assume the true BLEU score could be anywhere between 0.2361 and 0.5375. For a reference, the score obtained by Sutskever et.al and was between 0.33 and 0.36.

[8]A Systematic Comparison of Smoothing Techniques for Sentence-Level BLEU - http://acl2014.org/acl2014/W14-33/pdf/W14-3346.pdf

Some results have been shown below:





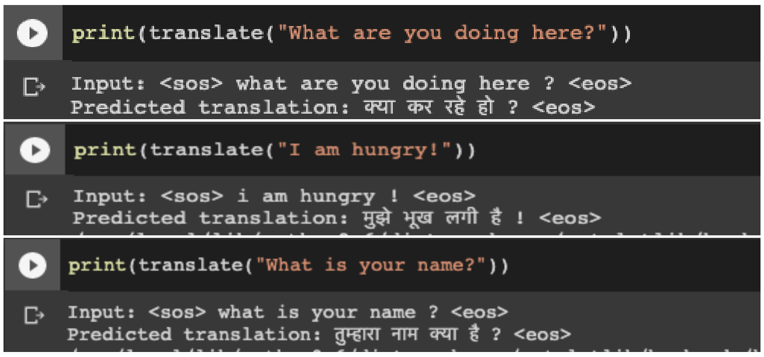
As we can observe from this attention plot (lowest activation value - dark purple to the highest activation value - bright yellow, as shown in the scale here), the word "vah" (meaning, "that", "he" or "she or "it") in Hindi gets most of its meaning for the English word "she" and "film" (meaning "film" / "movie") in Hindi gets most of its meaning from the English word "movie" and so on. This is interesting and helps us in getting a sense of which of the output words take the most meaning from which of the input words.

6 Conclusions

In this project, we looked at attention based encoder decoder model for the task of language translation between English and Hindi. Our model successfully produced several accurate translations with a BLEU score of 0.5375 on test data. Some areas of improvement in our model could be, training over a larger and more noisy dataset, using the BEAM search method for prediction of outputs as opposed to the greedy method, using an evaluation criteria other than BLEU (Since, simplistic ngram matching technique of BLEU is often incapable of differentiating between acceptable and unacceptable translations[9]). We observed that our model was not able to translate unseen words like proper nouns. It was also interesting to see how the model seemed to be skewed towards using male pronouns. Even if the input had explicitly female pronouns, like "she" and "her" in English, the model translated it into a sentence with male pronouns in Hindi. This is, without a doubt, a result of a skewed dataset with respect to gender pronouns. Therefore, we would like to extend this work to building machine translation models that use a more diverse data with respect to gender.

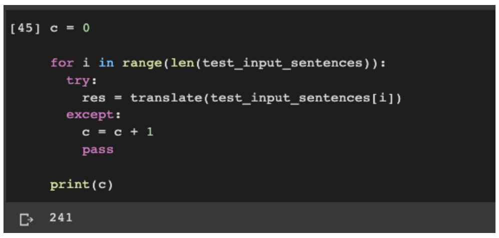
7 Additional Results

These are some translations we obtained while running the model.



```
print(translate("What are you doing here?"))

Input: <sos> what are you doing here ? <eos>
Predicted translation: क्या कर रहे हो ? <eos>

print(translate("I am hungry!"))

Input: <sos> i am hungry ! <eos>
Predicted translation: मुझे भूख लगी है ! <eos>

print(translate("What is your name?"))

Input: <sos> what is your name ? <eos>
Predicted translation: तुम्हारा नाम क्या है ? <eos>
```

Out of our test dataset, 241 sentences were not included in the BLEU score, since they contained words that were not present in the trained model and thus could not be translated.



```
[45] c = 0

     for i in range(len(test_input_sentences)):
       try:
         res = translate(test_input_sentences[i])
       except:
         c = c + 1
         pass

     print(c)

241
```

[9]Some Issues in Automatic Evaluation of English-Hindi MT: More Blues for BLEU Ananthakrishnan R, Pushpak Bhattacharyya, M Sasikumar, Ritesh M Shah

8 Acknowledgements

We would like to sincerely thank Professor Ravi Kothari for his exceptional teaching and for being so accommodating during such times. We are very grateful to Chanda who has offered us timely advice and help on all the concepts, assignments, and projects.

We want to acknowledge how helpful Tensorflow.org's tutorials have been in understanding the architecture, and most importantly, the problem better. We also want to acknowledge the Indic Languages Multilingual Parallel Corpus for Indic Languages Multilingual Tasks developed by Kyoto University and the National Institute for Information and Communications Technology, Japan for making Indic datasets available to the public for free. Further, we would like to acknowledge Bahdanau et. al. for their remarkable work on attention.

9 References

1. Brownlee, Jason. "Attention in Long Short-Term Memory Recurrent Neural Networks." Machine Learning Mastery, 14 Aug. 2019, machinelearningmastery.com/attention-long-short-term-memory-recurrent-neural-networks/.

2. Cho, et al. "Neural Machine Translation by Jointly Learning to Align and Translate." ArXiv.org, 19 May 2016, arxiv.org/abs/1409.0473.

3. Goodfellow, Ian, et al. Deep Learning. MIT Press, 2017.

4. Graham. "Neural Machine Translation and Sequence-to-Sequence Models: A Tutorial." ArXiv.org, 5 Mar. 2017, arxiv.org/abs/1703.01619v1.

5. Ilya, et al. "Sequence to Sequence Learning with Neural Networks." ArXiv.org, 14 Dec. 2014, arxiv.org/abs/1409.3215.

6. Luong, et al. "Effective Approaches to Attention-Based Neural Machine Translation." ArXiv.org, 20 Sept. 2015, arxiv.org/abs/1508.04025.

7. Mittal, Aditi. "Understanding RNN and LSTM." Medium, Towards Data Science, 12 Oct. 2019, towardsdatascience.com/understanding-rnn-and-lstm-f7cdf6dfc14e.

8. Some Issues in Automatic Evaluation of English-Hindi MT ... core.ac.uk/download/pdf/23798335.pdf.