SimplTran

English to Hindi Translator

Irfan Mohammed

Master of Science in
Computer Science

University of Massachusetts
Lowell

Abstract— **Natural Language Processing (NLP) has come a long way from the days of Rule-Based Translation. Today's leading models utilize Deep Neural Networks (DNN) and use Sequence to Sequence translation. This technique chunks sentences of one language, encodes them, runs them through a DNN, and decodes them into a another language. Hindi is the root of many Indian languages and is still used today. However, many leading translation tools do not implement the ability to do English-Hindi translations. The goal of this project it to utilize Google's TensorFlow Python module to create a Sequence to Sequence LSTM to translate to and from English and Sanskrit. The largest difficulty of the project is not necessarily the model but rather the data. Public English-Hindi translations are scarce. Acquiring accurate translations from the model using scarce data will be difficult. It will require minor changes to the model, as well as the acquisition of more English-Hindi translations.**

Keywords—Machine Learningt;Hindi; Translation; NLP

I. INTRODUCTION

Natural language processing is a popular machine learning technique these days for translating texts from one language to another. Currently, tools are available to translate most common or popular languages such as German, French, Danish, and Spanish. However, a translation tool does not exist for Hindi. Hindi is an ancient language that is one of the 22 official languages in the Eighth Schedule of the Indian Constitution. It is a very important language for the Hindu religion and various other religions as it is used as a basic language for communication and constitutes for the major part of India speaking and writing it as it still is used in texts and chants to this day. Also, according to the 2001 census of India, Hindi has about Tweleve million speakers (as a first, second, or third language), so an effective, automated written translator of full sentences would be widely useful. Therefore, the aim of SimplTran is to train an English-Hindi sentence translation model.
Among the many different languages already translated, Hindi is one language that is still missing effective tools for full translation.
There are numerous online dictionaries for English-Hindi word-by-word translation, but there is no full-fledged translator which takes the context and grammar of sentences into account.
Machine Translation is a growing yet well-developed discipline of data science, with a variety of methodologies and models that have been tried and evaluated on a variety of languages. It would be tough and interesting to apply some of these to Hindi translation particularly.

Hindi has non-Latin alphabets, but also an intermediate “Transliteration” language - called IAST - which is made up of Latin characters. This could be used to translate texts using existing models.
The model we'll use is called Encoder-Decoder LSTM, with LSTM standing for Long-Short Term Memory. The LSTM is a variant of the recurrent neural network. It operates by deducing the meaning of each word in a phrase from the meaning of preceding words. The context of the sentence can assist you figure out what each word in the sentence should be.
Loops are used in recurrent neural networks to allow information to remain in future events. Long-term and short-term dependencies can be learned via LSTM. An encoder-decoder and two submodels round out the model. The encoder is in charge of generalizing and summarizing meanings across several languages. The decoder, on the other hand, is in charge of predicting an output sequence within the supplied language, one character at a time.

II. Related works

Various authors [5] have discussed that Hindi is a viable language for Natural Language Processing as compared to other languages since its grammar and syntax is easier to understand. Words like “it is” and “I” do not have separate words in Hindi but can be easily decoded. Due to inflections, it reduces a four-letter English word to 2 letters in Hindi making it easier to decode and reducing storage space.

Upon research, two methods were found for the process of translating English to Hindi. The first comes from Young-Suk Lee who describes Statistical Machine Translation (SMT): an older but still effective method of translation. This method used Morphological Syntactic Features, also called ‘part-of-speech’ tags in conjunction with SMT which takes the language and uses their part-of-speech tags to create features for the statistical analysis of the text. The paper further explores prefix and suffix information in its mapping to make the model more powerful [2].

The other method comes from P Bahadur, A Jain, and D.S.Chauhan. They explain a general algorithm of translating between English and Hindi. A method of top-down processing can be used which starts with the English text, then breaks it up into tokens and phrases, continues to parse them, takes that output and parses it into Hindi grammar, and finally convert that phrase into Hindi. This exact process can be reversed in order to create Hindi-English translation as well, but additional tokens need to be added with the English. [3].

III. Datasets and features

The first dataset we are using is from Hindi_English_trunkated_corpus.csv . It contains a full verse by verse translation of the New Testament into Hindi. This dataset was found on one of the links provided by the Professor for doing the particular project. The following dataset contains of the following features

- Total No.of Sentences
  - English : 127606
  - Hindi: 127605
- Total No of Words
  - English: 1908266
  - Hindi: 2399762
- Average Word per sentence:
  - English: 14
  - Hindi: 18
- Vocabulary size
  - English:70502
  - Hindi: 81198

The data that comes from the Hindi query originally came in as unicode format, which we change to escape sequences (e.g., “\u028f”) using UTF-8 encoding. Each individual character is encoded this way and full words are combinations of various encodings. From here, we began the step of preprocessing the data. Our idea for preprocessing was to take the written text and convert each sentence of the text into an array of items. For example, the sentence: “The chicken crossed the street.” becomes [“The”, “chicken”, “crossed”, “the”, “street”, “.”].

I also had to go through and separate all the punctuation in the text since written language usually places words adjacent to ‘.’, ‘,’, ‘!’, and

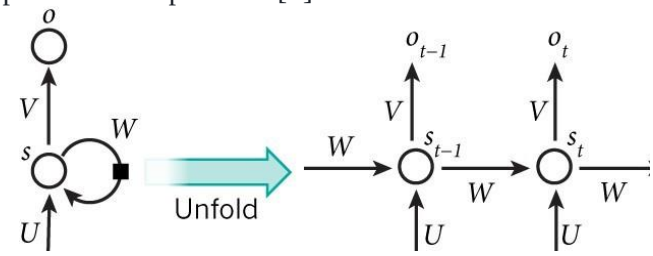
other such symbol. Once I had preprocessed this data in the English text, I applied the same preprocessing onto the unicode-escaped Hindi file using regex.

This was the only data set I have decided to use as it was pretty large and also would have taken a lot of time to complete as due to having the time restraint I decided to reduce the project data to around 10,000 samples from each of the languages to train and validate on which means there are about 10,000 samples of English and 10,000 samples of the Hindi language on my current working data set.

For the method we have implemented a recurrent neural network with an LSTM layer.

A. *Background*

Recurrent Neural Networks are artificial neural networks that form cyclic connections between different units and are predominantly used for processing sequential knowledge [7]. In traditional neural networks, it is assumed that the input and output are independent of each other. However, this leads to complications as for instance to predict the next letter in the alphabet, the previous one has to be known. RNNs are recurrent as they perform the same task for every element in the sequence, i.e., the output is dependent on previous computation [8].



Long short-term memory(LSTM) is a type of RNN which are used to learn long term dependencies. LSTMs help maintain the error that can be backpropgated[2] through time and layers [14]. They were developed to solve the issue of vanishing and exploding gradients in regular RNNs, allowing for training on longer sequences of inputs. Additionally, they can decide what they keep and remove from their cell state with the help of gates which are generally composed of the sigmoid function.

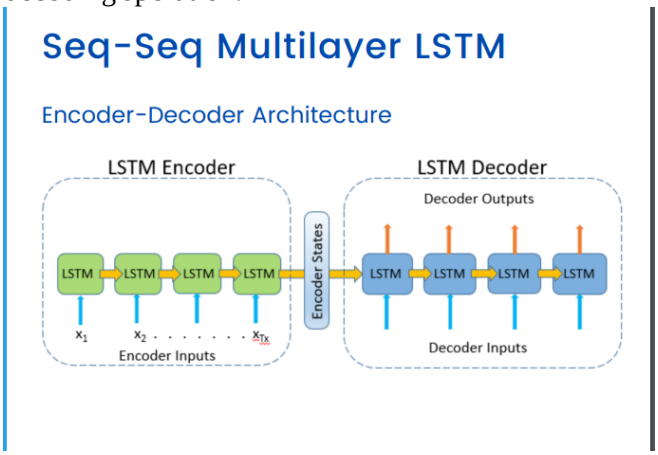
The equations for an LSTM are as follows from [9]:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$
$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$
$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma(W_c x_t + U_c h$$

Here, "f" controls which part of the previous state to weigh heavily, "i" controls which part of the input to weigh heavily, and "o" controls which part of the outputs to weigh heavily. "c" is the cell state, and "h" is the output, while W, U, and b are parameter vectors and matrices. σ is the sigmoid function.

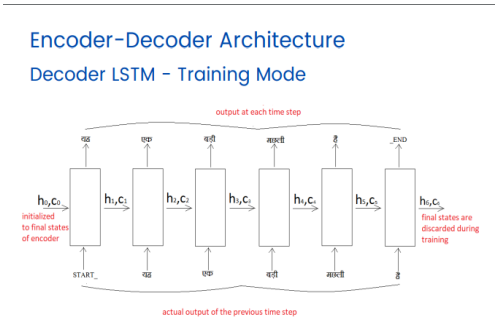
We make use of the "seq2seq" pattern for RNNs. This pattern describes a specific way of executing the neural network. While some RNN execution patterns will accept input and give output with each iteration, the seq2seq pattern splits execution into first a set of iterations for taking input, then followed by a set of iterations for generating output.

Our model is specifically an "encoder/decoder" model. This is a special category of seq2seq RNNs used for machine translation. The way the model runs is like any other seq2seq model, with a few additions. First, the model runs a set of iterations for the "encoding" operation. A phrase from the source language is passed into the inputs of the network, and with each iteration of the network, the recursive LSTM layer is keeping and updating a vector which represents an internal "understanding" of the input phrase so far. When the network has taken in the entire input phrase, a special input value signals the model to switch over to the decoding operation. One word at a time, the network outputs a translated word (based on the LSTM-layer internal vector). With each iteration, the output of the network is also passed back in as an input to the network, so the model can keep track of how far it is through translation. A special output signals the end of the decoding operation.

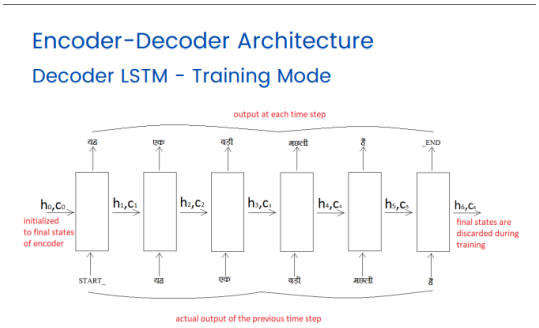


Finally, our model also takes advantage of an “attention” mechanism. With each iteration of the decoding operation, the current LSTM-layer internal vector is compared with those generated in each iteration of the encoding operation. This vector of comparisons, called the “context vector”, is combined with the current LSTM-layer state to produce an “attention vector”, which is used to produce the network output, and is given to the next iteration of the decoder [6].

The Encoder-Decoder architecture, Encoder plays the same role in both training and inference phase Important components of Encoder LSTM xi: Input sequence at time step i, represented as vectors using Internal State: hi & ci: Represent what the LSTM has read through each time step i. [Thought Vector] h0, c0: is initialized to 0 hk, ck: final step known as encoding of xi yi: Output sequence at step i, discarded in our encoder structure.

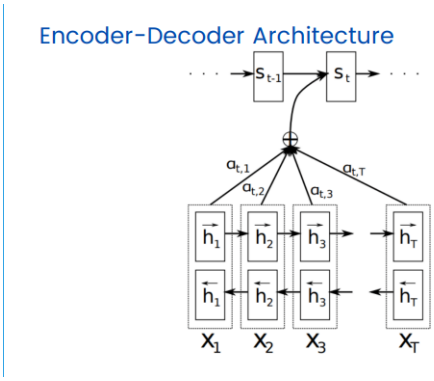


Decoder will generate the entire output sequence given the thought vectors, word by word. The initial states of the decoder are set to the final states of the encoder. The initial input to the decoder is always the START_ token. At each time step, we preserve the states of the decoder. The predicted output is fed as input in the next time step The loop breaks when the decoder predicts the _END

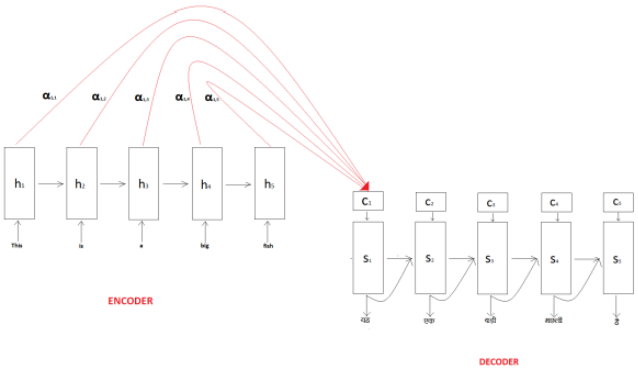


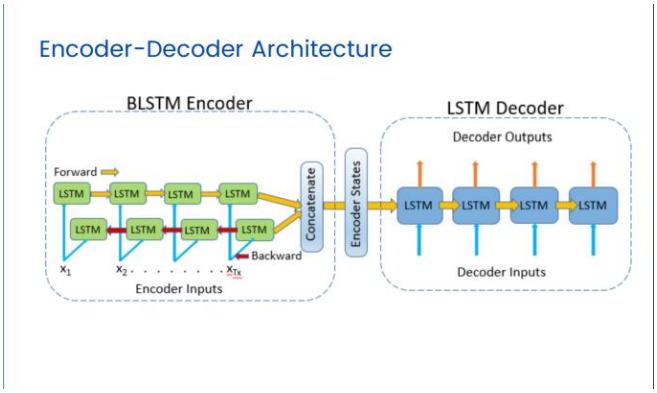
2 recurrent components: Forward & Backward The forward component computes the hidden and cell states like a standard LSTM The backward component computes the hidden and cell states by taking the input sequence in a reversechronological order This creates a way for the network to see the future and learn its weight accordingly, allowing to capture some dependencies. For correct encoding, the forward component have to be concatenated with those of the backward component respectively.

Attention Model:



Encoder works similar to the Multilayer LSTM model Decoder ’ s hidden state is computed with a context vector, the previous output and the previous hidden state It also has a seperate context vector c for each target word.



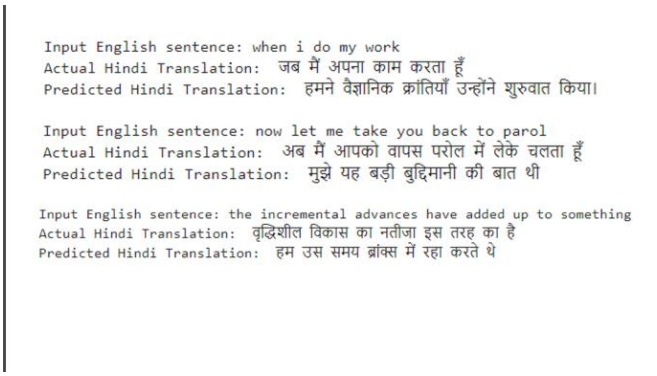


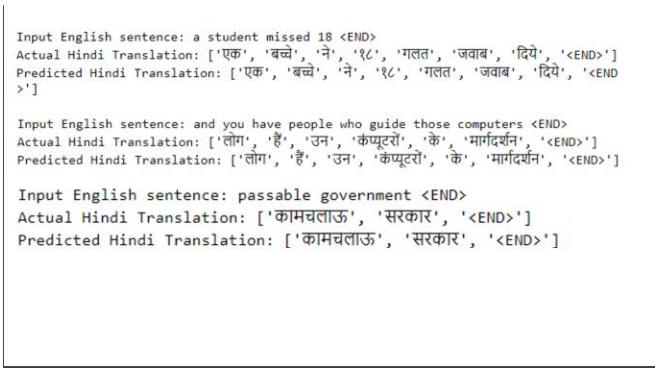
IV.RESULTS

Initially the losses rapidly fall and then there is a spike in our losses after which loss begins to fall again.

These are the following results I have obtained from running my models,the translation we receive for our current model. I notice that it initially translates correctly however, it unnecessarily repeats words for an extended period.

Architecture	No. of HiddenLSTM	BLEU Score
Multilayer LSTM	300	0.3389
	500	0.4915
Bidirectional LSTM	300	0.4350
	500	0.4995
Attention	256	1.0





V.DISCUSSION

The project still requires a fair amount of work. Although my model works for now, there are still a few flaws or features we can add to the current model. Currently, the model needs to be retrained every time the code is compiled and ran. Ideally, we would save the state of the most up to date model and reload it when needed for additional training and testing. Training the model with new and more diverse datasets may also strengthen the model. Diverse training sets could include a wider variety of language translations, in addition varying sources of data. Hopefully, this may help our model handle future Hindi translations.

The Data does come out to be perfect but I would like to add Image transcribing which would be more useful and can help in the project to work and a better evaluation can be made of.

REFERENCES

[1] Priscila Aleixo, Thiago Alexandre Salgueiro Pardo, Finding Related Sentences in Multiple Documents for Multidocument Discourse Parsing of Brazilian Portuguese Texts

[2] Tensorflow documentation, https://www.tensorflow.org/tutorials/seq2seq

[3] Ian Goodfellow and Yoshua Bengio and Aaron Courville. "*Deep Learning.*" MIT press (2016)

[4] Britz, Denny. "*Recurrent Neural Networks Tutorial, Part 1 – Introduction to RNNs.*" WildML. N.p., 08 July 2016. Web. 30 July 2017.

[5] Hochreiter, Sepp, and J ürgen Schmidhuber. *"Long Short Term Memory."* Neural

[6] Papineni, Kishore, et al. "Bleu: a Method for Automatic Evaluation of Machine Translation." *Aclweb.org*, IBM T. J. Watson Research Center, July 2002, www.aclweb.org/anthology/P02-1040.pdf.

Computation, 1997. Web.