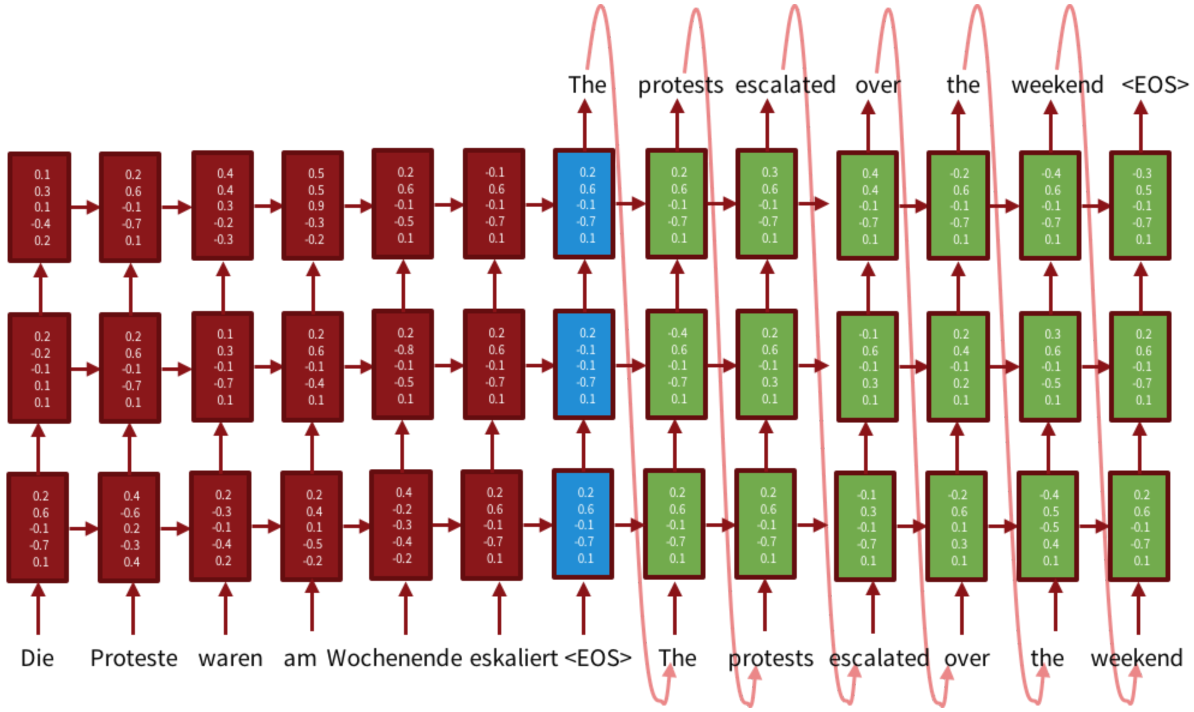
Neural Machine Translation - Hindi -> English



Ekta Sardana (2nd Year Grad, CS)

Ashish Vishwanath Shenoy (2nd Year Grad, CS)

ABSTRACT

In this project we build a deep neural network model to translate sentences from Hindi to English. We mainly draw upon previous work in the field of Neural Machine Translation(NMT) using Recurrent Neural Networks and LSTMs. Till now very little work has been published for translating Indian Languages using NMT. Most of the work revolves around translating from Japanese, Chinese, German to English. We show in this project that given enough parallel corpora of Hindi-English sentences, we can achieve a reasonable level of accuracy in translation by exploiting inherent language properties, Long Short Term Memory (LSTM), bidirectional RNNs with certain optimizations and attention mechanism. We also explore and experiment with optimization techniques such as ADAM, Dropout, Mini Batching and Beam Search in order to strengthen our NMT model. Below we also describe our experience of training the model with 'comparable' corpora vs 'parallel' corpora.

INTRODUCTION

"Machine Translation" is a term used to describe the process of using a software to translate text or speech from a source language to a target language. "Neural machine translation" or "neural sequence-to-sequence models" is a popular yet relatively new machine translation approach used primarily for handling human language. Departing from the traditional statistical approach of Machine Translation, NMT uses neural networks trained using deep learning techniques for translating sentences from one language to other. By generalizing the translation task as a sequence learning problem, these models have disrupted the domain of machine translation.

Why Hindi? Hindi is the national and official language of India. 425 million people speak Hindi as their first language while more than 12 million people as their second (Britannica, 2014). Outside India, some communities in South Africa, Mauritius, Bangladesh, Yemen, and Uganda also communicate in Hindi language.

The motivation to use Neural Machine Translation approach to translation and more specifically try it on Indian languages, are as follows :

1. NMTs require only a fraction of the memory needed by traditional statistical machine translation techniques.[1]

2. NMTs out-perform traditional state of the art statistical machine translation techniques.[2]
3. Neural Machine translation is often one of the main driving tasks behind the development of new models, and thus these models tend to be tailored to MT first, then applied to other tasks.
4. Very little work done in the field of Indian Languages translation using NMT.
5. Availability of a new parallel corpora of Hindi-English sentences made available by Institute of Formal and Applied Linguistics.
6. Additionally, NMT approaches can be combined with word-alignment approaches to address rare-word problems prevalent in statistical machine translations.
7. Recurrent Neural Networks can capture long distance dependencies using Long Short Term Memory. Sentences such as “**He** doesn't have very much confidence in **himself”** or **“She** doesn't have very much confidence in **herself”** would require such optimizations.

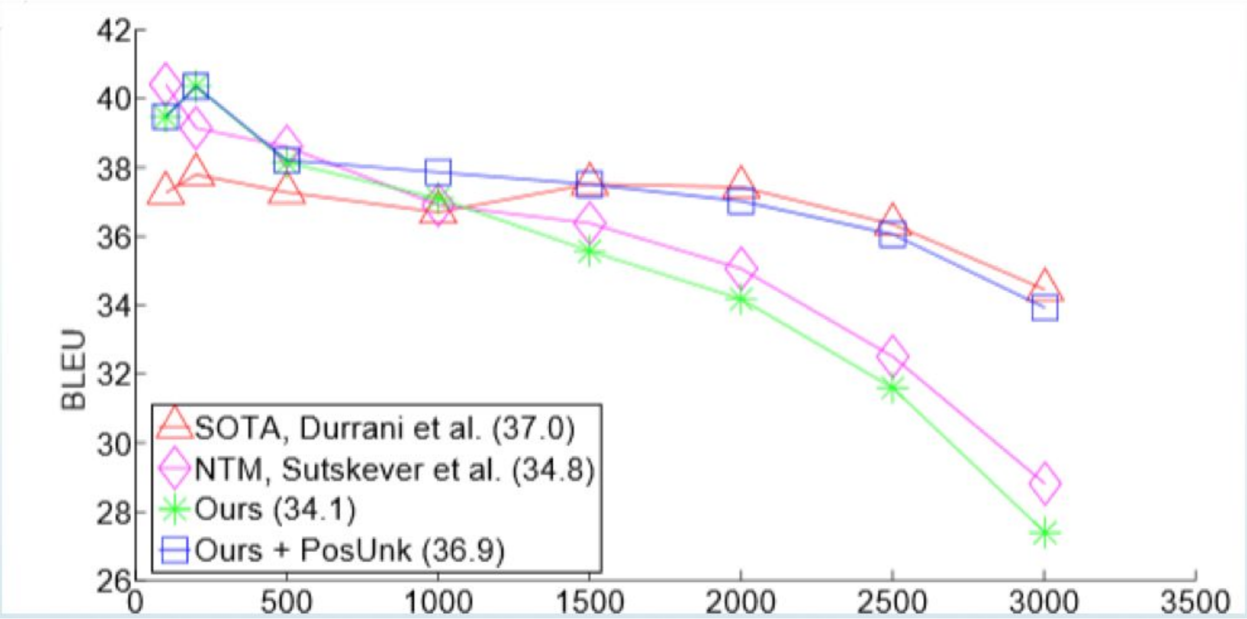


Figure 1. Performance of NMT[2]. Number of training examples on x-axis and BLEU[3] score on y-axis.

Over the course of this project we built a Neural Machine Translation Model using bidirectional RNNs augmented with LSTM trained on over 10000 english to hindi parallel sentences. We show below that for Hindi-English translations, bidirectional

RNNs outperform regular RNNs and how ADAM and attention mechanisms play a very important role in the speed of convergence.

PROBLEM DEFINITION AND ALGORITHM

Below we explain the main tasks and goals of this project along with details about the algorithms used.

I. Task Definition

Our main goals over the course of the project involved accomplishing the following :

1. Train a NMT model to translate Hindi Sentences to English.
2. Run the model on previously unseen Hindi sentences and evaluate the accuracy of the translated sentence.

The main tasks that were identified and executed are as follows :

1. Accumulate training data, clean it up and parse it. : Apart from the HindEnCorp dataset. These are parallel sentences which mean the same thing, but are not necessarily word to word translations. We wrote parsers and web crawlers to fetch comparable corpora from wikipedia Hindi translations of a few English pages.
2. Train a basic Recurrent Neural Network(RNN) encoder-decoder model to a reasonable accuracy.
3. Improve upon the baseline RNN model by mainly using optimizations in three components :
    a. Preparation of Corpus :
        i. Try different tokenization techniques : Allowing only alphanumeric, allowing punctuations.
        ii. Maximum number of unknowns.
    b. Optimize the encoder :
        i. Use ADAM for speeding up convergence.
        ii. Use dropout to avoid overfitting.
        iii. Use step decaying learning rate.
        iv. Use attention based RNNs.
        v. Use bidirectional RNNs.
        vi. Use mini batching

    vii. Vary the number of layers and hidden units

  c. Optimize the decoder :
    i. Vary the beam sizes
    ii. Use residual connections.
    iii. Vary the number of layers and hidden units

II. Algorithm Definition

We used Recurrent Neural Networks with Long Short Term Memory. Recurrent neural networks are a variety of neural network that makes it possible to model these long-distance dependencies. The idea is simply to add a connection that references the previous hidden state when calculating the current hidden state.

EXPERIMENTAL EVALUATION

I. Our Stack and RNN Structure

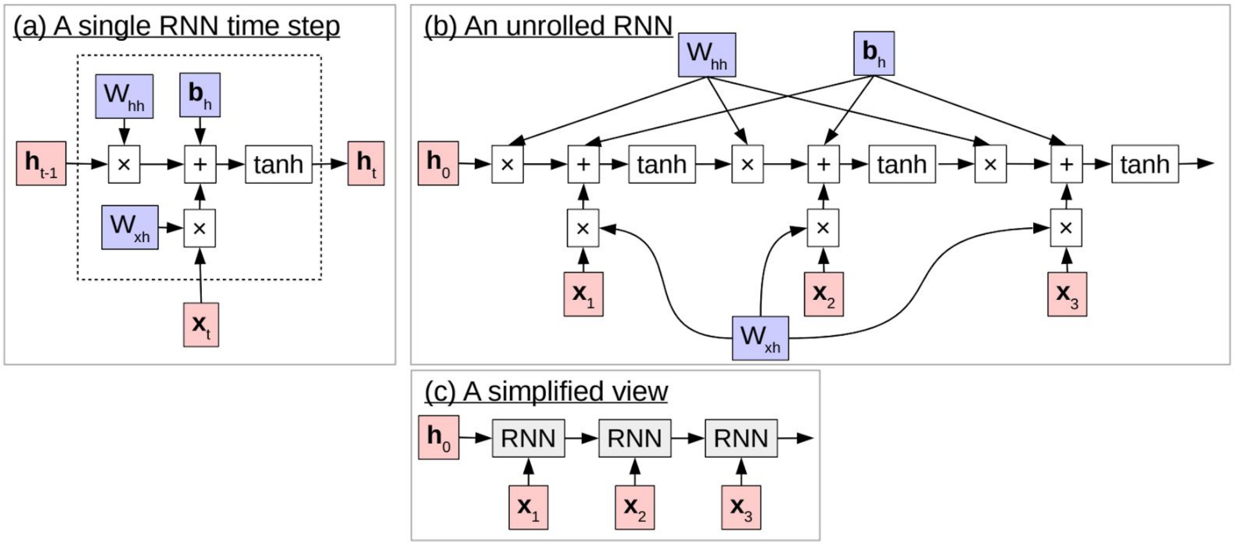
Our infrastructure consisted of :

- 1 AWS EC2 Instance: 1 Nvidia GPU and 8 CPUs
- OpenNMT library used to write the python encoders and decoders.
- PyTorch - internally used by the OpenNMT library.

The baseline architecture of our RNN model was as follows :

```
NMTModel (
  (encoder): Encoder (
    (word_lut): Embedding(30550, 500, padding_idx=0)
    (rnn): LSTM(500, 500, num_layers=2, dropout=0.3)
  )
  (decoder): Decoder (
    (word_lut): Embedding(32071, 500, padding_idx=0)
    (rnn): StackedLSTM (
      (dropout): Dropout (p = 0.3)
      (layers): ModuleList (
        (0): LSTMCell(1000, 500)
        (1): LSTMCell(500, 500)
      )
    )
    (attn): GlobalAttention (
      (linear_in): Linear (500 -> 500)
      (sm): Softmax ()
      (linear_out): Linear (1000 -> 500)
      (tanh): Tanh ()
    )
    (dropout): Dropout (p = 0.3)
  )
  (generator): Sequential (
    (0): Linear (500 -> 32071)
    (1): LogSoftmax ()
  )
)
batch_size=64,learning_rate=1.0,learning_rate_decay=0.8,
optim='sgd'
```

Below is an illustration of a single RNN cell and the simplified version of an encoder with a basic RNN.


(a) A single RNN time step

(b) An unrolled RNN

(c) A simplified view

Figure 2. Structure of a Recurrent Neural Network Model. Source : [4]

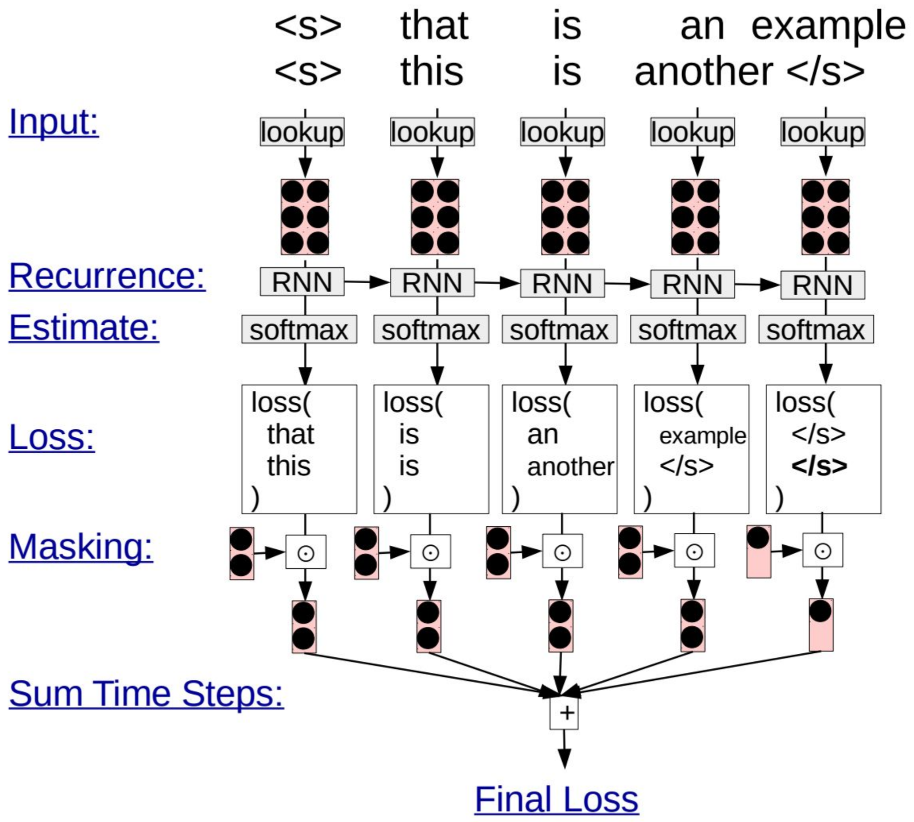


Figure 3. A mini batch example in a RNN. Source : [4]

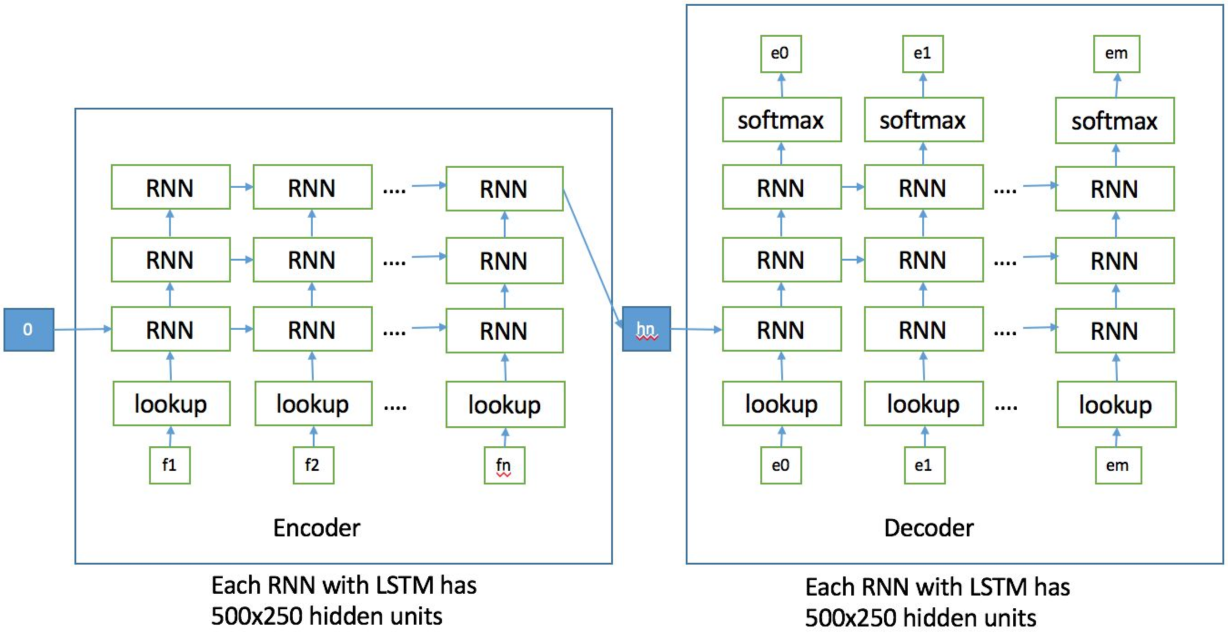

0
RNN
lookup
f1
f2
fn
....
Encoder
Each RNN with LSTM has 500x250 hidden units
hn
e0
e1
em
softmax
Decoder
Each RNN with LSTM has 500x250 hidden units

Figure 4. Our Encoder and Decoder structure.

II. Methodology

The procedure used to test the trained model is similar to the traditional way of evaluating learned machine learning models.

The main training datasets we used was a parallel corpora provided by the Institute of Formal and Applied Linguistics [5] called HindEnCorp version 0.5. It contains 273 thousand sentences and about 3.8 million tokens in each language. The source for this dataset is a combination of open source tool translations, TED talks, dictionary examples etc. A detailed list of sources can be found at [6].

The next set of dataset which we obtained was by writing web crawlers for wikipedia pages and splitting sentences. This was for purely experimental purposes. The main idea behind doing this was to avoid having word to word translated sentences and more naturally flowing sentences.

We used 30000 parallel sentences to train, test and validate our model.

We split our dataset in the ratio of 60:20:20 and used them as training data, validation data and testing data respectively.

We used the validation data to compare our trained models and tune our hyperparameters and other various components as described in the above section.

Testing data was used to generate the translations after finalizing the choice of the model parameters and training it.

Evaluating an NMT model essentially boils down to how accurately it models the actual source language. There are several different ways to do this but the most commonly used way of defining the accuracy is the log likelihood of the model with respect to the validation or test data.

$$log\, P(\varepsilon_{test}; \theta) = \Sigma\, log\, P(E; \theta)$$

The likelihood of the parameters θ with respect to this data is equal to the probability that the model assigns to the data[4].

Another measure of language model commonly used is called perplexity. It is defined as the exponent of the average negative log likelihood per word.

$$ppl(E_{test};\ \theta) = e - (log\, P(E_{test}; \theta))/length(E_{test})$$

An intuitive explanation of the perplexity is “how confused is the model about its decision?”[4]. More accurately, it expresses the value “if we randomly picked words from the probability distribution calculated by the language model at each time step, on average how many words would it have to pick to get the correct one?”

Apart from the **log likelihood** and **perplexity**, we additionally collected the **time taken**, **average source tokens processed per second** and **average target tokens processed per second** for each batch in epoch.

We mainly evaluated the above statistics by building a **learning curve**.

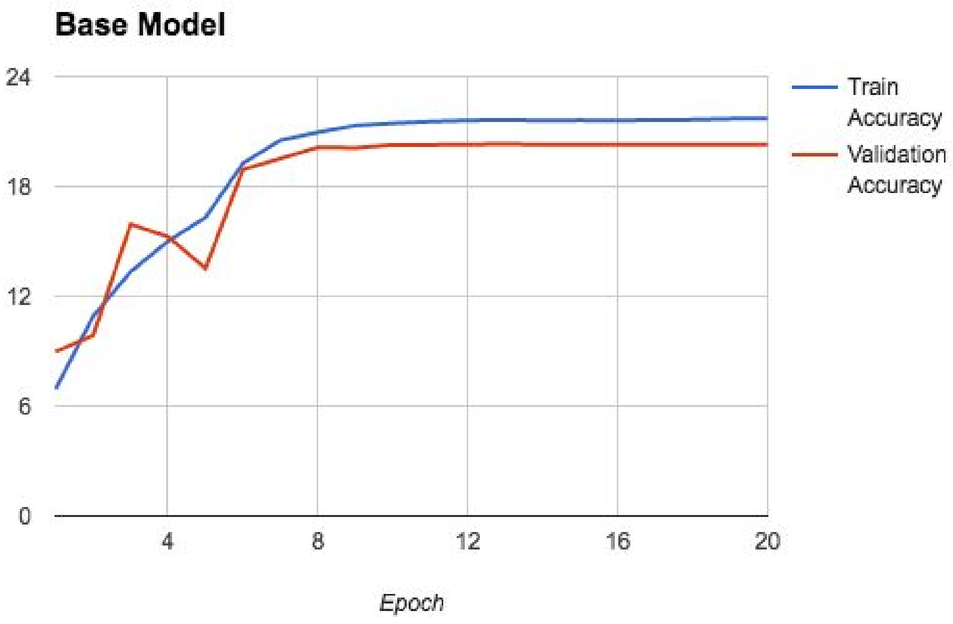
The baseline we used for evaluating our eventual models was the basic default RNN structure provided by OpenNMT.

The variables of the model are as follows :

- Number of layers and hidden units in the RNN.
- Maximum sequence length.
- Batch size used in mini batching.
- Initial Learning rate.
- Beam size.
- Dropout percentage.

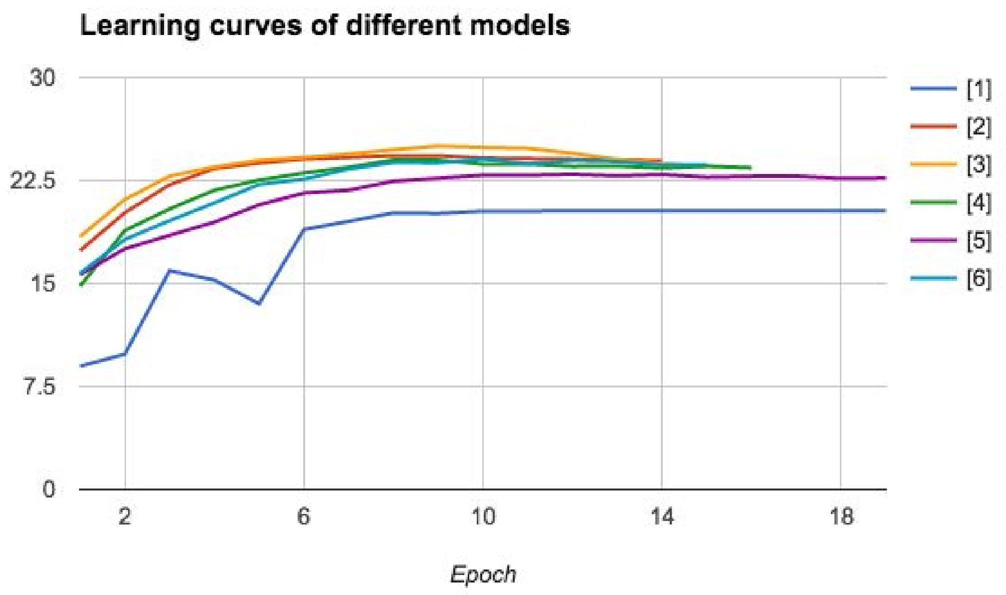
III. Results

1. Learning curve for basic RNN structure



Base Model	Training	Validation
Accuracy	21.7122@20	20.3011@10
Perplexity	322.513@20	709.488@4

2. Learning curve for RNN with ADAM
3. Learning curve for RNN with ADAM, bi-directional encoders.
4. Learning curve for RNN with ADAM, bi-directional encoders and number of layers as 3 in both encoder and decoder
5. Learning curve for RNN with ADAM, bi-directional encoders, number of layers as 3 in both encoder and decoder, dropout as 0.5
6. Learning curve for RNN with ADAM, bi-directional encoders, number of layers as 3 in both encoder and decoder, dropout as 0.5 and number of hidden units as 600



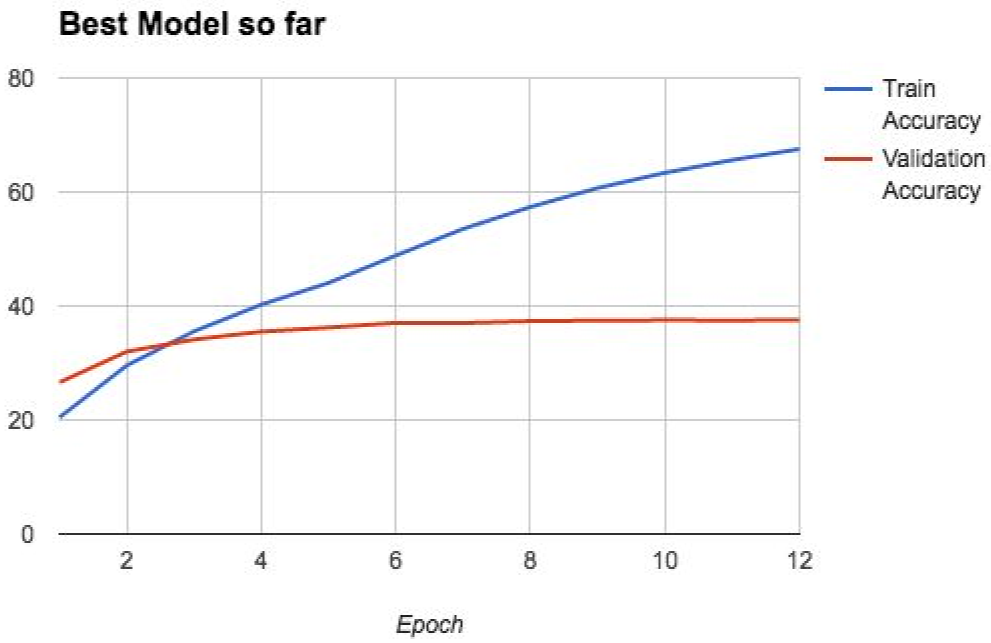
Note: Numbers in the legend correspond to the above experiments

Model	[1]	[2]	[3]	[4]	[5]	[6]

Best Training Perplexity	321.838@19	8.03305@14	7.91158@13	9.43013@16	27.6476@19	15.4117@15
Best Validation Perplexity	516.657 @ 8	477.487@4	469.119@4	499.117@5	548.208@6	472.098@6
Training Accuracy	21.7122@20	58.7494@14	56.9305@13	55.8606@16	40.2696@19	47.398@15
Validation Accuracy	20.3011@18	24.3206@8	25.0173@9	24.013@4	22.9627@12	23.8902@13

As it can seen from the learning curves of different models and table above, model 3 which is RNN with ADAM and bidirectional encoders outperform other models, we train a new model with same parameters but larger dataset(around 70000 training instances)

7. Learning Curve of Best Model which uses model 3 from above and 70000 training instances.



Best Model so far	**Training**	**Validation**

Accuracy	67.5341@12	37.5135@10
Perplexity	4.33129@12	131.725@4

8. Few translations using the best model.

Sentence: मैं मजाक नहीं कर रही हूँ.

True: i'm not kidding.

Predicted: i'm not kidding.

Sentence: इस पसंद द्वारा निर्दिष्ट किए गए स्थान पर जाएँ

True: go to the location specified by this bookmark

Predicted: go to the location specified by this bookmark

Sentence: जनसंखाय घनत्व प्रति वर्ग कि.मी. १२६ है।

True: the population density is 126 persons per sq.km.

Predicted: the density density is 126 persons per sq.km.

Sentence: तो मैंने उसी पल में निश्चय कर लिया

True: so i made up my mind at that moment

Predicted: so i took up at the same moment

Sentence: एक रात मुझे नींद नहीं आ रही थी,

True: i was finding it hard to fall asleep one night,

Predicted: i was going to wake asleep one night,

IV. Discussion

Using **Mini Batching** significantly speeded up our training. The time taken per epoch reduced from x to y. This was expected as gradient updates take time for such a big network and batching updates speeds up this process.

Using **ADAM** over SGD and dropping the learning rate to 0.001 while using ADAM helped achieving higher accuracies on validation data early. Also, perplexity starts dropping significantly faster when using ADAM than standard SGD.

Bidirectional Encoders helped in improving the accuracy a bit. The basic idea is that one RNN encodes the word from left to right and other RNN from right to left and then both representations are concatenated for each word.

Attentional models fare better than the ones without. The main idea behind attention models is that when we want to generate a particular target word e_i, that we will want to focus on a particular source word f_j, or a couple words around the word. In order to express this, attention calculates a "context vector" c_i that is used as input to the softmax in addition to the decoder state [7].

Increasing the number of layers in encoder and decoder and adding more hidden units resulted in overfitting to the training data due to smaller data set(17,000 training instances) and then increasing dropout rate didn't help in improving the accuracy.

INDIVIDUAL CONTRIBUTION

Following are the contributions of each individual in this project :

Ashish Shenoy	Obtaining training data.
	Writing parsers and crawlers for the training data.
	Setting up OpenNMT library.
	Writing the preprocessing and encoder python script.
	Tuning the hyperparameters and running the experiments
	Preparing the report.

Ekta Sardana	Setting up the AWS EC2 instance.
	Writing the decoder model python script.
	Automating and running the training scripts.
	Tuning the hyperparameters and running the experiments.
	Literature survey and project proposal.
	Preparing the report

RELATED WORK

Although there is a lot of work in the area of Neural Machine Translation, related work in this specific case of Hindi to English translation and vice versa is limited. Google recently started using NMT models for their google translate[8]. Their model consisted of a deep LSTM network with 8 encoder and decoder layers using residual connections and attention mechanism.

Using the different optimization techniques described above we were able to improve the test accuracy of our model from **20.3011** to **25.0173** on just 17000 training sentences. Using 69000 training sentences we were able to achieve **37.5135** accuracy. Hence if a larger corpus and larger number of training examples are used the result can be improved considerably.

FUTURE WORK

One main feature we did not change from our default RNN structure was increasing the number of encoders and decoders. This was due to our hardware and time constraints. An extension to this work can be to experiment with larger numbers of encoders and decoders. Another area to explore would be how ensembles will perform on this data set. An interesting extension to this work could also be to use Convolutional Neural Networks for decoding and combine them with the LSTM encoder.

CONCLUSION

Neural Machine Translation (NMT) Models are a game changer in the domain of machine translation. It performs significantly better than traditional statistical approaches and the fact that we could train a decent translation engine in a short amount of time using just 1 GPU and limited dataset implies that this is a promising approach to achieve higher precision and speed in training and learning new translation models.

BIBLIOGRAPHY

[1] Multi-Task Learning for Multiple Language Translation, Daxiang Dong, Hua Wu, Wei He, Dianhai Yu and Haifeng Wang Baidu Inc, Beijing, China, [https://www.aclweb.org/anthology/P/P15/P15-1166.pdf]

[2] Effective Approaches to Attention-based Neural Machine Translation, T Luong et al, ACL 2015 [http://aclweb.org/anthology/D15-1166]

[3] https://en.wikipedia.org/wiki/BLEU

[4] Neural Machine Translation and Sequence-to-sequence Models: A Tutorial, Graham Neubig [https://arxiv.org/pdf/1703.01619.pdf]

[5] https://ufal.mff.cuni.cz/hindencorp

[6] http://ufallab.ms.mff.cuni.cz/~bojar/hindencorp/

[7] https://github.com/neubig/nmt-tips

[8] https://research.google.com/pubs/pub45610.html

[9] Visualizing and understanding recurrent networks, 2015, Andrej Karpathy, Justin Johnson, and Li Fei-Fei [https://arxiv.org/pdf/1506.02078.pdf]

[10] Long short-term memory. Neural computation, Sepp Hochreiter and Jurgen Schmidhuber [http://www.bioinf.jku.at/publications/older/2604.pdf]