Department of Electrical Engineering,
Indian Institute of Technology Madras

Neural Machine Translation for Low Resource Languages

B.Ramasubramanian (ee13b127)

Submitted in partial fulfilment of the requirements for the degree of
Bachelor of Technology

Thesis Certificate

This is to certify that the thesis titled **Neural Machine Translation for Low Resource Languages**, submitted by **B.Ramasubramanian (ee13b127)**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. S. Umesh
Research Guide
Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

Abstract

KEYWORDS: Attention Mechanism, Bidirectional LSTMs, BLEU, Embeddings, Encoder-Decoder, English-Hindi, LSTM, Neural Machine Translation, RNN-LM, Transliteration

Machine Translation (MT) is a task in Natural Language Processing (NLP), where automatic systems are used to translate the text from one language to another while preserving the meaning of source language. Statistical Machine Learning and phrase-based methods have been traditionally used for translation purposes. Neural machine translation approaches that have recently come up hold promise as some of them have outperformed their SMT counterparts. In this work, we analyze the well-documented English-French translation task. We then perform the English-Hindi translation task using an attention-based encoder-decoder model. We analyze the effect of different additions to this model and document them. We also perform a transliteration task, as well as implement an RNN language model, which is used for rescoring.

Acknowledgements

I would like to thank **Prof. S. Umesh** for his constant guidance and support during the course of the project. He was very approachable and always guided me whenever I dropped by his office. His inputs during the weekly meetings were very valuable and helped me be on track. The two courses that I did under him and this project have provided me a proper sense of direction and have motivated me to pursue masters in the field of Signal Processing and Deep Learning.

A special thanks to **Swaroop Kotni** for being a great project partner. I learnt a lot of things from him throughout the course of the project and I am grateful for his constant support, even when I wasn't able to get the desired results. I would also like to extend my gratitude to **Sandeep Kothinti** and the other lab-mates who were always available to solve my doubts and to discuss the possible future directions for the project from time-to-time. I would also like to thank **Prof. Mitesh Khapra** for providing access to the larger English-Hindi corpora.

Contents

Chapter 1

Introduction

1.1 Task description

1.1.1 Dataset description

The parallel corpora for the translation task was obtained from the Ninth Workshop on Statistical Machine Translation, ACL 2014. We got access to both the English-French and the English-Hindi corpora.

- Translation task
  - Training data

    The English-French dataset was from multiple sources, such as EuroParl parallel corpus, which is extracted form the proceedings of the European Parliament. EuroParl contains 2,007,723 sentences with 51,388,643 French words and 50,196,035 English words. It also contains a French single language dataset with 2,190,579 sentences and 54,202,850 words. On the whole, we had around 44 million sentences. It requires certain pre-processing steps such as tokenization, lowercasing, stripping the empty lines, removing the lines with XML tags etc.

    The English-Hindi parallel corpora is from HindEnCorp, a dataset collected by Charles university. It contained 287,000 sentences, with 3.6 mn English words and 3.97 mn Hindi words. The HindMonoCorp, which is basically a collection of general web and web news crawls, has 43.41 mn sentences of monolingual Hindi data, which has around 790.83 Hindi words. This corpus was a part of the WMT14 Shared Translation Task. Similar pre-processing steps, as was stated for the English-French dataset, are needed here also.

  – Development data

    At times, we split the training data as 90% training and 10% validation data. At other times, when the development set was provided, such as the news-test2013 dataset, we used those for validation.

- Transliteration task

  The dataset used for this task was the NEWS 2012 (Named Entities Workshop) shared task dataset, which has 14122, 997 and 1000 words in the training, validation and the test set respectively.

- Larger dataset for the translation task

  Recently, we obtained a dataset from the Workshop on Asian Translation (WAT 2016), which uses the IITB corpus. The training corpus is a mixed domain corpus. The development and test set are from the News domain and are exactly the same as the ones in WMT 2014. This training set has 1.5 mn sentences.

1.1.2 Difficulties in the task

- Non-monotonicity

  The first task that we performed was the English-French translation task. The alignment of words in a sentence between English and French is largely monotonic. Our transliteration task was also a monotonic one. But, English-Hindi translation is non-monotonic, and hence we couldn't simply adapt the work done for the other tasks to this. English is structurally classified as a Subject-Verb-Object (SVO) language with a poor morphology whereas Hindi is a morphologically rich, Subject-Object-Verb (SOV) language. These fundamental structural differences result in large distance verb and modifier movements across English-Hindi. As far as morphology is concerned, Hindi is more richer in terms of case-markers, inflection rich surface forms including verb forms etc. Hindi exhibits gender agreement and syncretism in inflections, which are not observed in English. As we mention later, a major jump in the BLEU score was obtained by using this non-monotonicity property.

- Limited literature

  Even though we could find considerable amount of literature for Statistical Machine Translation, the amount of work done on Neural Machine Translation was comparatively less, especially the work done on Hindi. Workshops such as WMT, WAT etc have started including Hindi in their translation task only from 2014 and 2016 respectively, and some of the information from these workshops is only accessible to the participants.

- Non-generalizable behaviour

  We suffered from this problem quite often. For example, introduction of peepholes improved the accuracy in case of the transliteration task, but either didn't affect or marginally reduced the BLEU score in case of the translation task. We figure that this could be due to the non-monotonicity property of the translation task. Even then, we weren't able to reproduce some of the results that we obtained on smaller amounts of data, when the whole dataset was used. This was probably because the smaller dataset wasn't a good representative and wasn't capturing all the variability.

- Limited training data

  As is well known, neural network models perform well only when there's 'sufficient' data available for training. We initially suffered from the problem of limited training data, as can be seen from the dataset description. Hence, it wasn't a surprise when the SMT gave better results than the NMT. This difficulty has now been overcome with the availability of a new and larger dataset.

1.2 Preliminaries

1.2.1 Xavier initialization

It helps signals reach deep into the network.

- If the weights in a network start too small, then the signal shrinks as it passes through each layer until its too tiny to be useful.
- If the weights in a network start too large, then the signal grows as it passes through each layer until its too massive to be useful.

Xavier initialization makes sure the weights are just right, keeping the signal in a reasonable range of values through many layers. The statistical explanation of Xavier initialization follows:

Suppose we have an input **X** with n components and a linear neuron with random weights **W** that spits out a number **Y**. Then,

$$Y = W_1X_1 + W_2X_2 + \ + W_nX_n$$
$$Var(W_iX_i) = E[X_i]^2Var(Wi) + E[W_i]^2Var(X_i) + Var(W_i)Var(i_i)$$

If the inputs and weights both have mean **0**, that simplifies to

$$Var(W_iX_i) = Var(W_i)Var(X_i)$$

Moreover, if X_i and W_i are all independent and identically distributed,

$$Var(Y) = Var(W_1X_1 + W_2X_2 + \ + W_nX_n) = nVar(W_i)Var(X_i)$$

If we want the variance of the input and output to be the same, that means $nVar(W_i)$ should be 1, which means the variance of the weights should be

$$Var(W_i) = 1/n = 1/n_{in} = 2/(n_{in} + n_{out})$$

The last formula can be derived by working through the same steps for the backpropogated signal, and realizing that $n_{in} = n_{out}$.

1.2.2 Dropouts

Dropouts are mainly used as regularization operators to avoid overfitting in LSTMs. Dropout operator is only applied to the non-recurrent connections. Borrowing the notations from a standard LSTM, we can write:

$$\begin{bmatrix} i \\ f \\ o \\ g \end{bmatrix} = \begin{bmatrix} sigm \\ sigm \\ sigm \\ tanh \end{bmatrix} T_{2n,4n} \begin{bmatrix} D(h_t^{l-1}) \\ h_{t-1}^l \end{bmatrix}$$

where D is the dropout operator, and

$$c_t^l = f \odot c_{t-1}^l + i \odot g$$
$$h_t^l = o \odot tanh(c_t^l)$$

The dropout operator corrupts the information carried by the units, forcing them to perform their intermediate computations more robustly. At the same time, all the information from the units isn't erased. It is especially important that the units remember events that occurred many timesteps in the past. By not using dropout on the recurrent connections, the LSTM can benefit from dropout regularization without sacrificing its valuable memorization ability

1.2.3 Optimizers

Here we use the Stochastic Gradient Descent (SGD) optimizer as the starting point and then explain the other optimizers

- Stochastic Gradient Descent
  SGD performs a parameter update for each training example $x^{(i)}$ and label $y^{(i)}$. It is usually much faster than batch gradient descent. SGD performs frequent updates with a high variance that cause the objective function to fluctuate heavily. SGD's fluctuation, on the one hand, enables it to jump to new and potentially better local minima. On the other hand, this ultimately complicates convergence to the exact minimum, as SGD will keep overshooting.

$$\theta = \theta - \eta \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

- Momentum
  SGD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in

one dimension than in another. Momentum helps accelerate SGD in the relevant direction and dampens oscillations. It does this by adding a fraction γ of the update vector of the past time step to the current update vector.

$$v_t = \gamma v_{t\text{-}1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

- Nesterov Accelerated gradient It effectively looks ahead by calculating the gradient not w.r.t. to the current parameters θ but w.r.t. the approximate future position of the parameters:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$

- Adagrad
  It adapts the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameters. But, Adagrad accumulates the squared gradients in the denominator, which keeps growing during training. This in turn causes the learning rate to shrink and eventually become infinitesimally small.

$$v_t = v_{t-1} + (\nabla_\theta J(\theta))^2$$
$$\theta_t = \theta_{t-1} - \tfrac{\eta}{\sqrt{v_{t-1} + \epsilon}}(\nabla_\theta J(\theta))$$

- RMSProp
  Due to Adagrad's weakness mentioned above, RMSProp was introduced, which avoids the decay of the denominator.

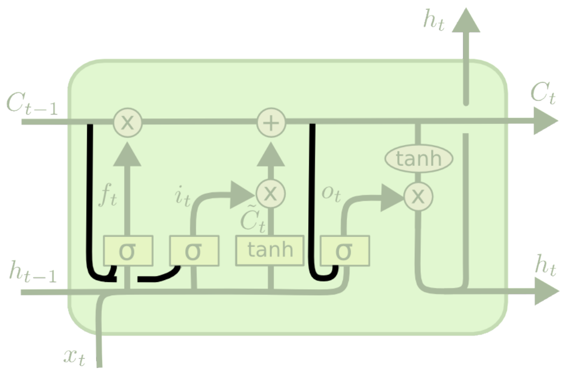
$$v_t = \beta v_{t-1} + (1 - \beta)(\nabla_\theta J(\theta))^2$$

- Adam
  Adaptive Moment Estimation uses a cumulative history of the gradients apart from doing whatever RMSProp does. m_t and v_t are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively, hence the name of the method. As m_t and v_t are initialized as vectors of zeros, they are biased towards zero, especially during the initial time steps, and especially when the decay rates are small (i.e. β_1 and β_2 are close to 1).

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)(\nabla_\theta J(\theta)$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla_\theta J(\theta))^2$$
$$\theta_t = \theta_{t-1} - \tfrac{\eta}{\sqrt{v_{t-1} + \epsilon}} m_{t-1}$$

1.2.4 Peephole connections

Assuming that the reader is aware of LSTM connections, we now describe a variant of that which was introduced by Schmidhuber and Gers (2000): peephole connections. During learning, no error signals are propagated back from gates via peephole connections. Peephole connections are treated like regular connections to gates (e.g. from the input) except for update timing. The peephole connections allow all gates to inspect the current cell state even when the output gate is closed. This information can be essential for finding well-working network solutions. Peepholes can either be added to all gates (as shown in the figure below) or to some of the gates.



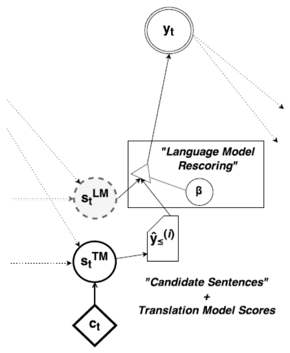
$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \ + \ b_o\right)$$

1.2.5 Shallow Fusion

The idea of shallow fusion was proposed by Gulchere et al. At each time step, the translation model proposes a set of candidate words. The candidates are then scored according to the weighted sum of the scores given by the translation model and the language model. The score of the new word is computed by

$$logp(y_t = k) = logp_{TM}(y_t = k) + \beta logp_{LM}(y_t = k)$$

where β is a hyper-parameter that needs to be tuned to maximize the translation performance on a development set.


y_t
"Language Model Rescoring"
β
s_t^{LM}
$\hat{y}_{\leq}^{(i)}$
s_t^{TM}
"Candidate Sentences" + Translation Model Scores
c_t

Chapter 2

Literature Survey

2.1 Related work

Though a good amount of literature exists w.r.t translation, here we discuss only the work related to English-Hindi translation. The English-French model has been described in detail later.

The first work that we describe here was a part of the work submitted to WMT2014 shared task by IIT Bombay. The core components of their translation system are **phrase based (Hindi-English) and factored (English-Hindi) Statistical Machine Translation systems**. They use number, case and Tree Adjoining Grammar information to improve English-Hindi translation, primarily by generating morphological inflections correctly. They show improvements to the translation systems using pre-processing and post-processing components. To overcome the structural divergence between English and Hindi, they preorder the source side sentence to conform to the target language word order. To overcome the difficulties faced due to a small parallel corpus, they translate out-of-vocabulary words and transliterate named entities in a post-processing stage. Their work also investigates ranking of translations from multiple systems to select the best translation. We used this work as a baseline for our task, but due to limited training data, their SMT system outperformed our NMT system.

People have looked at supertags, which are elementary trees of Lexicalized Tree Adjoining Grammar (LTAG) (Joshi and Schabes, 1991). They provide syntactic as well as dependency information at the word level by imposing complex constraints in a local context. These elementary trees are combined in some manner to form a parse tree, due to which, supertagging is also known as An approach to almost parsing(Bangalore and Joshi, 1999). A supertag can also be viewed as fragments of parse trees associated with each lexical item. Some works have also explored verb chunk reordering and noun inflection rules.

Another related work comes from IIIT Allahabad. They explore a hybrid approach for English-Hindi translation. A combined approach of **phrase based statistical machine translation (SMT), ex-**

ample based MT (EBMT) and rule based MT (RBMT) is proposed to develop a novel hybrid data driven MT system capable of outperforming the baseline SMT, EBMT and RBMT systems from which it is derived. In short, the proposed hybrid MT process is guided by the rule based MT after getting a set of partial candidate translations provided by EBMT and SMT subsystems. It is known that EBMT systems are capable of outperforming the phrase-based SMT systems and RBMT approach has the strength of generating structurally and morphologically more accurate results. The hybrid approach that this work proposes increases the fluency, accuracy and grammatical precision which improve the quality of a machine translation system. They have also compared the proposed hybrid machine translation (HTM) model with renowned translators i.e. Google, BING and Babylonian and showed that the proposed model works better on sentences with ambiguity as well as comprised of idioms than others.

A related work by A Gehlot, V Sharma et al. looks at a **transfer based Machine translation system**. Their system takes an input text file and checks its structure through parsing. Reordering rules are used to generate the text in the target language. They claim it to better than corpus based MTS because corpus Based MTS require large amount of word aligned data for translation that is not available for many languages while transfer Based MTS requires only knowledge of both the languages(source language and target language) to make transfer rules. They are able to get correct translation for simple assertive sentences and almost correct for complex and compound sentences.

There also exists some literature on **using effective selection in multi-model SMT.** K. Sachdeva, R. Srivastava et al. from IIIT Hyderabad have shown that multi-model systems perform better than the individual models. They have described a Hindi to English statistical machine translation system and improved over the baseline using multiple translation models. They have considered phrase based as well as hierarchical models and enhanced over both these baselines using a regression model. Their system is trained over textual as well as syntactic features extracted from source and target of the aforementioned translations. Their system shows significant improvement over the baseline systems for both automatic as well as human evaluations. Their proposed methodology is quite generic and they claim that it can be easily extended to other language pairs as well. Other transfer based approaches, usually for English to other languages, include those by CDAC Pune (MANTRA, MATRA), IISc Bangalore (SHAKTI) and IIIT Hyderabad.

Most of the work on English-Hindi translation is based on SMT based approaches. The following work by E.Gupta and R.Gupta from IITK explores an NMT-based approach. They start off with the aim of developing a technique to produce a **high quality parallel sentence-aligned corpus from existing subject/document-aligned comparable corpora**, since limited parallel corpus is a major obstacle for most NMT based tasks. They have tried to build a comparable corpus from non-sentence-aligned and untranslated bilingual topic-aligned documents. They train a weak translator using limited parallel corpus. The weak translator and an aligning heuristic (ex: Hunalign) are used to create additional parallel corpus. They then retrain a neural translator on the generated bigger parallel corpus.

Chapter 3

Neural Machine Translation Model

Let us first recall the sequence-to-sequence LSTM model. The Long Short-Term Memory model of [8] is defined as follows. Let x_t, h_t, and m_t be the input, control state, and memory state at timestep t. Given a sequence of inputs $(x_1, \ldots, x_T)$, the LSTM computes the h-sequence $(h_1, \ldots, h_T)$ and the m-sequence $(m_1, \ldots, m_T)$ as follows.

$$\begin{aligned}
i_t &= \operatorname{sigm}(W_1 x_t + W_2 h_{t-1}) \\
i'_t &= \tanh(W_3 x_t + W_4 h_{t-1}) \\
f_t &= \operatorname{sigm}(W_5 x_t + W_6 h_{t-1}) \\
o_t &= \operatorname{sigm}(W_7 x_t + W_8 h_{t-1}) \\
m_t &= m_{t-1} \odot f_t + i_t \odot i'_t \\
h_t &= m_t \odot o_t
\end{aligned}$$

The operator $\odot$ denotes element-wise multiplication, the matrices $W_1, \ldots, W_8$ and the vector h_0 are the parameters of the model, and all the nonlinearities are computed element-wise.

In a deep LSTM, each subsequent layer uses the h-sequence of the previous layer for its input sequence x. The deep LSTM defines a distribution over output sequences given an input sequence:

$$\begin{aligned}
P(B|A) &= \prod_{t=1}^{T_B} P(B_t | A_1, \ldots, A_{T_A}, B_1, \ldots, B_{t-1}) \\
&\equiv \prod_{t=1}^{T_B} \operatorname{softmax}(W_{\mathrm{o}} \cdot h_{T_A+t})^\top \delta_{B_t},
\end{aligned}$$

The above equation assumes a deep LSTM whose input sequence is $x = (A_1, \ldots, A_{T_A}, B_1, \ldots, B_{T_B})$, so h_t denotes t-th element of the h-sequence of the topmost LSTM. The matrix W_{o} consists of the vector representations of each output symbol and the symbol δ_b is a Kronecker delta with a dimension

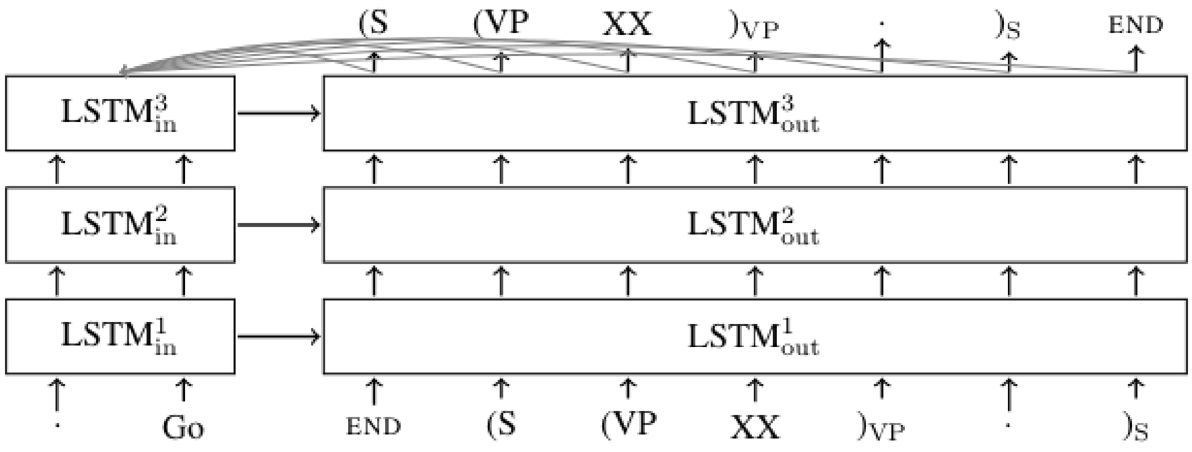


Figure 3.1: Encoder- Decoder Model

for each output symbol, so $\text{softmax}(W_o \cdot h_{T_A+t})^\top \delta_{B_t}$ is precisely the B_t'th element of the distribution defined by the softmax. Every output sequence terminates with a special end-of-sequence token which is necessary in order to define a distribution over sequences of variable lengths. We use two different sets of LSTM parameters, one for the input sequence and one for the output sequence, as shown in Figure 3.1. Stochastic gradient descent is used to maximize the training objective which is the average over the training set of the log probability of the correct output sequence given the input sequence.

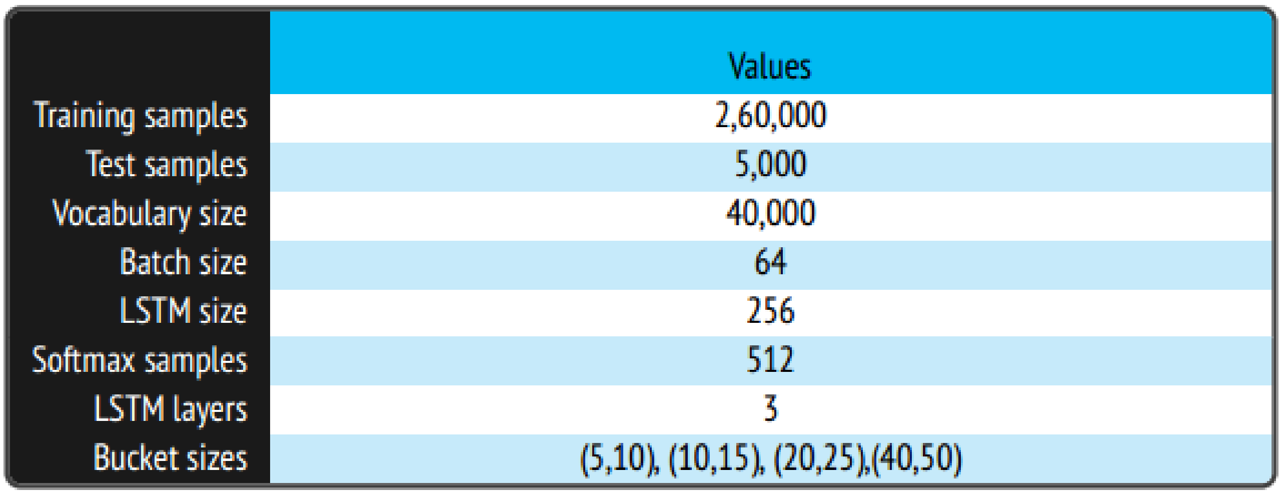
3.1 Attention Mechanism

An important extension of the sequence-to-sequence model is the addition of an attention mechanism. We adapted the attention model from [2] which, to produce each output symbol B_t, uses an attention mechanism over the encoder LSTM states. Similar to our sequence-to-sequence model described in the previous section, we use two separate LSTMs (one to encode the sequence of input words A_i, and another one to produce or decode the output symbols B_i). The encoder hidden states are denoted $(h_1, \ldots, h_{T_A})$ and we denote the hidden states of the decoder by $(d_1, \ldots, d_{T_B}) := (h_{T_A+1}, \ldots, h_{T_A+T_B})$.

To compute the attention vector at each output time t over the input words $(1, \ldots, T_A)$ we define:

$$\begin{aligned} u_i^t &= v^T \tanh(W_1' h_i + W_2' d_t) \\ a_i^t &= \text{softmax}(u_i^t) \\ d_t' &= \sum_{i=1}^{T_A} a_i^t h_i \end{aligned}$$

The vector v and matrices W_1', W_2' are learnable parameters of the model. The vector u^t has length T_A



	Values
Training samples	2,60,000
Test samples	5,000
Vocabulary size	40,000
Batch size	64
LSTM size	256
Softmax samples	512
LSTM layers	3
Bucket sizes	(5,10), (10,15), (20,25),(40,50)

Figure 3.2: Model Parameters

and its i-th item contains a score of how much attention should be given to the i-th hidden encoder state h_i. These scores are normalized by softmax to create the attention mask a^t over encoder hidden states. In all our experiments, we use the same hidden dimensionality (256) at the encoder and the decoder, so v is a vector and W_1' and W_2' are square matrices. Lastly, we concatenate d_t' with d_t, which becomes the new hidden state from which we make predictions, and which is fed to the next time step in our recurrent model.

3.2 Improvisations

The stock code from TensorFlow which was used for training English-French translation model gave us a BLEU score of **0.33**. The following changes were made to the code:

3.2.1 Xavier Initialization

Xavier initialization was employed to initialize the parameters, and the BLEU score increased to **2.45**

3.2.2 Dropouts

Dropouts reduce the number of variables to learn and hence would theoretically yield better results for low resource trainings. Upon implementation, the results agreed with the theoretical expectations by yielding BLEU scores of 2.61 and 2.66 for introducing input and (input and output) dropout layers respectively.

3.2.3 NOT Reversing the inputs

In the English-French model, the inputs were reversed and fed for training because doing so established short term dependencies between the input and the output sentences. This in turn helped the model to train better and yield better results. However, such short term dependencies cannot be established in the English-Hindi translation model by reversing the inputs. This is because English-Hindi translation is a non-monotonic task, as opposed to the monotonic English-French translation task. Hence, we **did not** reverse the input sentences and as expected, we got better scores. The BLEU score improved from **2.66** to **3.82**.

3.2.4 Peepholes

Peepholes were introduced in the LSTM cells to check whether they would help the model during training. The motivation behind this was the fact that they helped in improving the accuracy in the transliteration task. But, the BLEU score reduced from **3.82** to **3.59**. Hence, we did not include peepholes in the final model.

3.2.5 Transliteration

The goal of transliteration is to write a word in one language using the closest corresponding letters of a different alphabet or language. More formally, the goal of transliteration is to transform a name (a string of characters) in one language (and corresponding script) to the target language (and corresponding script) while ensuring phonemic equivalence preserving and conforming to the phonology and conventions of the target language. We built a sequence-to-sequence model for transliteration of words between English and Hindi languages. We achieved an accuracy (as defined in section 4.5) of about 48%. This task could help in tackling the problem of _UNK tags in the outputs (as has been discussed in section 5.2.8). Since our translation model jointly learns to align and translate, we can utilize the alignment details and find out the English word aligned with an unknown tag in the Hindi output sentence and replace it with the transliteration of the English word. This would aid us in correctly translating proper nouns which would usually not occur in vocabulary. However, the transliteration model remains to be integrated with the NMT model and is left for future work.

3.2.6 Shallow Fusion

An RNN language model was trained to improve the syntactic and semantic proficiency of the generated Hindi sentence. The model was trained on a pilot subset of the available monocorpus to ensure the end-to-end working of the code. The built RNN model was used to rescore the outputs from the

NMT model. The model also performed better than the baseline established in the lab in terms of perplexity during training (achieving a validation set perplexity of 105 as comapred to the baseline of 219). Although, a concrete result cannot be reported, we believe that when the RNN model is trained on the entire available monocorpus, the system would output much more refined and accurate sentences (since the 4-gram accuracy is really low now and an RNN-LM can fix that)

Chapter 4

Documentation

This chapter delves deeper into the work done during the course of the project. It also acts as a documentation of the work done by us, from where someone working on this project in the future can carry it forward.

4.1 Code Organization

The whole project was carried out under two usernames viz. batch5 and kjjswaroop. Most of the work was done in kjjswaroop while batch5/kjjswaroop folder was used to train models on bigger datasets. The code, saved models and the corpora used in the project can be found in the following directories.

- **/ speech/ kjjswaroop**
  - **translate** : contains code for the neural translation model.
    - **translate_hinen.py** : Main file containing the TensorFlow graph for the NMT model.
    - **seq2seq_model.py** : Describes the NMT model. All changes to the model have to be done in this file.
    - **data_utils.py** : Contains utility functions required for preprocessing corpora before feeding them into the model.
  - **ptb_print** : contains code for the RNN language model
    - **ptb_word_lm.py** : contains the RNN and TensorFlow graph for the language model.
    - **reader.py** : Contains utility functions that read and process corpora to feed them into the model.
  - **BLEU**: compares the candidate text files (outputs of the NMT model) with the reference text file and calculates BLEU score.

    * **calculatebleu_betterblue.py** : code to calculate BLEU score by calculating the number of 1,2,3,4-gram matchings and taking a geometric mean of them.
  - **nmt_data** : Each of the subdirectories in nmt_data contains processed corpus, vocabulary, test files and model saved checkpoints for a particular model configuration, evident in the name of the subdirectory.
  - **transliteration** :
    * **translate.py** : Main file containing the TensorFlow graph for the NMT model.
    * **seq2seq_model.py** : Describes the transliteration model. All changes to the model have to be done in this file.
    * **attention.py** : contains the bidirectional RNN implementations. Edit this file to change the structure of RNNs.
    * **data_utils.py** : Contains utility functions required for preprocessing corpora before feeding them into the model.
  - **Queue_dir** : contains the perl scripts to run any job on either CPUs or GPUs in the lab network.

/ speech /batch5 /kjjswaroop/ contains similar folder organization. The code has been forked into this directory to train various models on larger datasets due to privilege/resource limitations in kjjswaroop account. The NMT model can be run on IITB corpus here. The RNN Hindi language model can be run on the available 10GB monocorpus.

4.2 Data Processing

4.2.1 Hindi text normalization

Hindi text in all the corpuses were normalized before using them for training translation/language models. Usually, the normalizer classes do the following:

- Some characters have multiple Unicode code points. The normalizer chooses a single standard representation
- Some control characters are deleted
- While typing using the Latin keyboard, certain typical mistakes occur which are corrected by the module
- Performs some common normalization, which includes:

  - Byte order mark, word joiner, etc. removal
  - ZERO_WIDTH_NON_JOINER and ZERO_WIDTH_JOINER removal
  - ZERO_WIDTH_SPACE and NO_BREAK_SPACE replaced by spaces

In addition to basic normalization, Hindi specific normalization is carried out according to the following norms:

- Replace the composite characters containing nuktas by their decomposed form
- replace pipe character '|' by poorna virama character
- replace colon ':' by visarga if the colon follows a character in this script

4.3 Neural Machine Translation Model (NMT) Model

4.3.1 translate_hinen.py

This file contains the TensorFlow graphs for training and decoding the NMT model. Training the NMT follows the below scheme:

- Create model.
- Read data into buckets and compute their sizes.
- A bucket scale is a list of increasing numbers from 0 to 1 that we'll use to select a bucket. Length of [scale[i], scale[i+1]] is proportional to the size if i-th training bucket, as used later.
- while True:
  - Choose a bucket according to data distribution. We pick a random number in [0, 1] and use the corresponding interval in train_buckets_scale.
  - Get a batch and make a step.
  - Once in a while, we save checkpoint, print statistics, and run evals.
  - Decrease learning rate if no improvement was seen over last 3 times.
  - Save checkpoint and zero timer and loss.
  - Run evals on development set and print their perplexity.

The pseudo algorithm for decoding a new sentence using trained models is given as follows:

- Create model and load parameters
- Load vocabularies
- Decode from standard input
- while sentence:
  - Get token-ids for the input sentence
  - Which bucket does it belong to?
  - Get a 1-element batch to feed the sentence to the model
  - Get output logits (probabilistic interpretation) for the sentence
  - This is a greedy decoder - outputs are just argmaxes of output_logits
  - If there is an EOS symbol in outputs, cut them at that point
  - Print out Hindi sentence corresponding to outputs

4.3.2 seq2seq_model.py

This file contains the model used in the NMT system. We have employed an encoder decoder model with attention mechanism for English-Hindi translation.

- Create Model with the following arguments:
  - **source_vocab_size** : size of the source vocabulary
  - **target_vocab_size**: size of the target vocabulary
  - **buckets**: a list of pairs (I, O), where I specifies maximum input length that will be processed in that bucket, and O specifies maximum output length. Training instances that have inputs longer than I or outputs longer than O will be pushed to the next bucket and padded accordingly. Buckets are used to speed up training. Otherwise, one long sentence in a batch will cause all the other sentences to have unnecessarily large number of pads at the end, thus wasting computations. Instead, if all the sentences in a batch are picked from the same bucket, the overall number of pads can be reduced, thus increasing the efficiency.
  - **size**: number of units in each layer of the model
  - **num_layers**: number of layers in the model
  - **max_gradient_norm**: gradients will be clipped to maximally this norm
  - **batch_size**: the size of the batches used during training
  - **learning_rate**: learning rate to start with

  - **learning_rate_decay_factor**: decay learning rate by this much when needed
  - **use_lstm**: if true, we use LSTM cells instead of GRU cells
  - **num_samples**: number of samples for sampled softmax
  - **forward_only**: if set, we do not construct the backward pass in the model

- Create the internal multi-layer cell for our RNN
- Create the seq2seq function: we use embedding for the input and attention
- Create placeholders for inputs
- Our targets are decoder inputs shifted by one
- Calculate training outputs and losses
- Perform gradients and SGD update operation for training the model
- Save the model at checkpoints

The file also contains functions to execute one complete step of the model and to create batches for training from the datasets.

4.3.3 data_utils.py

This file contains all the utility functions required to pre process data. It handles the following tasks:

- **Create vocabulary**: The function runs through the entire corpus and forms a vocabulary by listing the N (an input parameter to the function) most frequently occurring words in the corpus.
- **Initialize vocabulary**: If the vocabulary already exists in the specified folder, it simply loads the vocabulary file. Else, it raises an error.
- **Sentence To Token IDs** : Convert all the sentences in the corpus into token IDs based on the vocabulary. If a word is not found in the dictionary, it replaces the word with _UNK ID.

4.4 RNN LM model for Hindi language

The code for the RNN language model is available in the **/speech/batch5/kjjswaroop/ptb_print**.

4.4.1 temp_rescore.py

This file contains the TensorFlow graph and the model. The code contains four different models: Small, Medium, Large and Test. Based on the size of training data available, one can choose the type of model. The test model, as the name says, should be used to test the code.

- Create LSTM cell of required dimensions and depth
- Add dropout wrapper to the LSTM cell while training the model
- Calculate the cell state and outputs
- Calculate output logits by using output projections
- Jointly train the parameters of the LSTM cell and the output projections by estimating loss function
- Update cell state, logits and loss

The code also contains rescore() function which evaluates the probability of occurrence of an input sentence.

4.4.2 reader.py

The file contains data utility functions required during pre-processing the corpus for training. The code implements the following functions:

- **Build vocabulary**: The function iterates through the whole corpus and forms a vocabulary of the N most frequently occurring words in the corpus.
- **File To Word IDs** : Convert all the sentences in the corpus into token IDs based on the vocabulary. If a word is not found in the dictionary, it replaces the word with _UNK ID.

4.5 Transliteration

We trained a sequence-to-sequence model using the encoder-decoder architecture. We used a vocabulary size of 60 for English and 130 for Hindi. The overall structure of the network is as follows:

- INEMBED: A feedforward layer of size $|V_s|$ X inembsize, where V_s is the source language vocabulary (i.e. set of characters) and inembsize is the output size of the layer (Inembsize = 256).
- ENCODER: bidirectional LSTM layer with encsize outputs in either direction (encsize = 256).
- DECODER: LSTM layer with decsize outputs (decsize = 512).
- SOFTMAX: softmax layer for classification, decsize inputs and V_t outputs, where V_t is the target language vocabulary (i.e. set of characters).
- OUTEMBED: A feedforward layer of size $|V_t|$ X outembsize, where outembsize is the output size of the layer (outembsize = 256). This layer is used for obtaining the embedding of the output character and feeding it to the input of the decoder for the next time step.
- ATTENTION: Consists of i) a single feedforward network whose inputs are the previous decoder state, previous decoder outputs embedding and the annotation vector (encoder output) under consideration. It outputs a single attention score. (ii) a softmax layer over the attention scores from each annotation vector to convert it to a probability value. The attention weights from the softmax layer are used to linearly interpolate the annotation vectors. The resulting context vector will be an input to the decoder along with the decoder output in the previous timestep.

The objective function is to minimize the negative log-likelihood. The primary evaluation metric that we used for this task was Accuracy (exact match). This means that the output is considered correct only if it matches the reference transliteration exactly.

As can be inferred from the above description, the folder structure and the code organization of **kjjswaroop/transliteration** are similar to **kjjswaroop/translation** folder. However, the sequence-to-sequence model has been edited by replacing simple RNNs with bi-directional RNNs.

4.6 BLEU Score evaluation

BLEU (bilingual evaluation understudy) is an algorithm for evaluating the quality of text which has been machine-translated from one natural language to another. Quality is considered to be the correspondence between a machine's output and that of a human: "the closer a machine translation is to a professional human translation, the better it is". BLEU was one of the first metrics to achieve a high correlation with human judgments of quality, and remains one of the most popular automated and inexpensive metrics. Scores are calculated for individual translated segments generally sentences by comparing them with a set of good quality reference translations. Those scores are then averaged over the whole corpus to reach an estimate of the translation's overall quality. Intelligibility or grammatical correctness are not taken into account. BLEU is designed to approximate human judgment at a corpus level, and performs badly if used to evaluate the quality of individual sentences. BLEU scores are calculated using the file **kjjswaroop/Bleu/calculatebleu_betterbleu.py**. A brevity penalty ensures that a high-scoring candidate translation matches the reference translations in length, in word choice, and in word order. It prevents very short candidates from receiving too high a score. Let r be the total length of the reference corpus, and c be the total length of the translation corpus. If $c \leqslant r$, the brevity penalty applies, defined to be $e^{(1-r/c)}$. The code counts n-gram matchings between the candidate sentence and the reference sentence for n=1,2,3,4 and maintains the counts in a dictionary. Once all the n-gram counts are calculated, a geometric mean of all the accuracies (number of correct n-grams/total number of n-grams) multiplied with the brevity penalty gives the BLEU score.

4.7 Shell scripts

We have used shell scripts to run all our codes on the lab resources. Following are the shell scripts used throughout the project:

- **/batch5/kjjswaroop**
  - **translate_script.sh**: This script is used to decode test data and calculate the BLEU score by comparing the candidate file with the reference file.

```
export PATH=/usr/local/cuda-8.0/bin${PATH:+:${PATH}}
export
   LD\_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda-8.0/lib64
python translate/normalize.py
python translate/translate_hinen.py --decode
   --train_dir=data_iitb_corpus --data_dir=data_iitb_corpus
   --from_train_data=IITB.en-hi.en
   --to_train_data=IITB_norm.en-hi.hi
   --from_dev_data=dev_test_tokenized/dev.en
   --to_dev_data=dev_test_tokenized/dev_norm.hi
   --from_vocab_size=40000 --to_vocab_size=40000 --num_layers=3
   --size=256
python Bleu/calculatebleu_betterblue.py ˜output_dev.txt
   test_norm_hi.txt
```

  - **testcript_rescore.sh**: This script is used to train the RNN model.

```
export PATH=/usr/local/cuda-8.0/bin${PATH:+:${PATH}}
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda-8.0/lib64
python ptb_print/temp_rescore.py --model=small
   --data_path=/speech/batch5/kjjswaroop/
   --save_path=/speech/batch5/kjjswaroop/rnn_saved_models
   --rnn_dir=/speech/batch5/kjjswaroop/rnn_saved_models
```

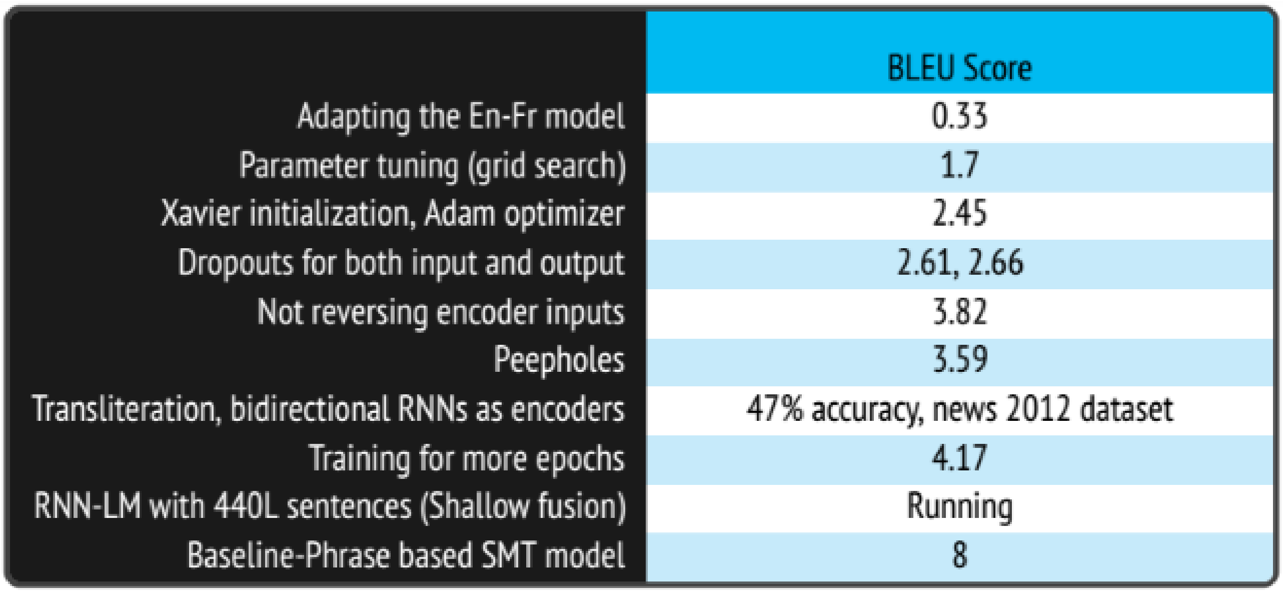
In **/speech/kjjswaroop/**, there are similar shell scripts; **train_script.sh** can be used to train the NMT model.

Chapter 5

Conclusion

5.1 Summary

The results achieved during the course of the project are tabulated below.



	BLEU Score
Adapting the En-Fr model	0.33
Parameter tuning (grid search)	1.7
Xavier initialization, Adam optimizer	2.45
Dropouts for both input and output	2.61, 2.66
Not reversing encoder inputs	3.82
Peepholes	3.59
Transliteration, bidirectional RNNs as encoders	47% accuracy, news 2012 dataset
Training for more epochs	4.17
RNN-LM with 440L sentences (Shallow fusion)	Running
Baseline-Phrase based SMT model	8

Figure 5.1: Experiment Results

5.2 Future Work

5.2.1 Problem of repeated tokens

We observed that our model suffered from a problem where the last few tokens kept repeating without being able to identify the EOS (end-of-sentence) tag. The problem was very prominent when we did the transliteration task, where its effect on accuracy was much more pronounced than in the translation task. We haven't been able to find a reason for that behaviour and hence leave it as future work.

5.2.2 Parameter tuning

It goes without saying that parameter tuning plays a major part in any neural network framework. We managed to get good results from most models, but there's always room for improvement, especially at places wherein we tried several different models in a short span of time (such as RNN-LM), which restricted the amount of time we could put into playing with the parameters.

5.2.3 Weight sharing

In our model, the encoder and decoder use different set of parameters, which is also the common practice. But since the training data is limited, weight sharing might help in better learning as the number of parameters to be learnt is less. But at the same time, there's no logical reasoning as to why different languages should share weights, and hence this wasn't tried.

5.2.4 Larger dataset

Recently, we acquired a parallel corpus that has 7 times as much data as our original one. This dataset was a part of WAT 2016 (Workshop on Asian Translation). It's a well known fact that the larger the training data, the better the performance of neural networks. We ran it for as many epochs as the smaller dataset and it gave a slightly better BLEU score. We realize that a learning rate decay factor close to 1 could be used and the model could be trained for many more epochs. The model is currently under training.

5.2.5 RNN-LM training on whole corpus

The RNN-LM training is very slow. As was pointed out by Mikolov, the very first version of RNN-LM trained by Bengio et al. without importance sampling took 113 days to train. Our model is very slow in reading the words and tokenizing them, even though we split the task across 40 CPUs. We have already shown that our RNN-LM performs better than the baseline on a standard dataset (as mentioned already), but that training set was much smaller.

5.2.6 Bi-directional RNNs for translation

Bi-directional RNNs gave marginal improvements in the transliteration task, which motivated us to implement it for the translation task as well. But the concatenation of the cell state (memory state) and the hidden state couldn't be performed as we expected and we ran into a few other coding errors, which we couldn't fix despite help from several others. Hence, this problem remains one of future curiosity.

5.2.7 Parse trees and transliteration

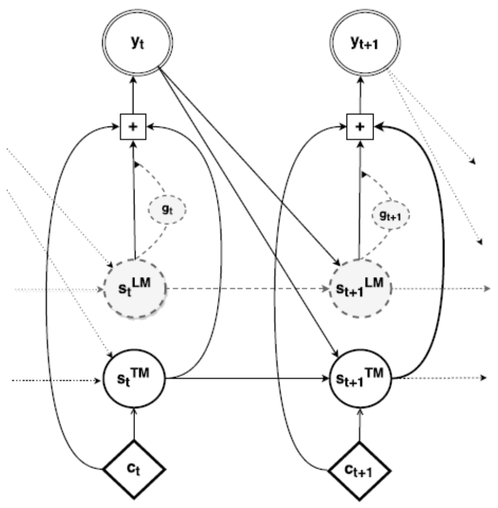
A parse tree is an ordered, rooted tree that represents the syntactic structure of a string according to some context-free grammar. Syntax models the knowledge people unconsciously have about the grammar of their native language. Parsing extracts syntax from a sentence. A basic English sentence has a subject (noun), a head (verb) and an object (noun). One can run a grammar backwards to find possible structures for a sentence. Parsing can be viewed as a search problem, a hidden data problem. Parsing can be top-down or bottom-up. The former is goal-driven, to find the ways of parsing a given sentence. The latter is data-driven, where you look for small pieces that you know how to assemble, and work your way up to larger pieces, but this suffers from the problem of creating local structure that doesn't benefit the global structure. There are several standard parsing algorithms, for example Cocke-Kasami-Younger (CKY), which is like Viterbi but for trees. Since we have already done the transliteration task, one can apply parse trees and get the POS labels (part-of-speech) for both the languages. One can then find those parts in the English sentence for which the output Hindi token was UNK, and replace those by their transliteration. This idea works because usually the words that turn out to be unknown are proper nouns and transliterating them makes sense.

5.2.8 Deep fusion

In deep fusion, the RNNLM and the decoder of the NMT are integrated by concatenating their hidden states next to each other. The model is then finetuned to use the hidden states from both of these models when computing the output probability of the next word. The hidden layer of the deep output takes as input the hidden state of the RNNLM in addition to that of the NMT, the previous word and the context such that

$$p(y_t|y_{<t}, x) \propto exp(y_t^T(W_o f_o(s_t^{LM}, s_t^{TM}, y_{t-1}, c_t) + bo))$$

The structure learned by the LM from the monolingual corpora isn't overwritten since only the parameters used for the output are finetuned. A decoder with a controller mechanism could also be used to dynamically weight the different models depending on the word being translated. In our work, shallow fusion gave marginally better results when used for rescoring. Gulchere et al. have shown that deep fusion performs much better, which is a motivation to try this out in the future.



Bibliography

[1] Ilya Sutskever, Oriol Vinyals, Quoc V. Le., " Sequence to Sequence Learning with Neural Networks"

[2] Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate"

[3] Sebastien Jean, Kyunghyun Cho, Roland Memisevic, Yoshua Bengio, "On Using Very Large Target Vocabulary for Neural Machine Translation."

[4] Piyush Dungarwal, Rajen Chatterjee, Abhijit Mishra, Anoop Kunchukuttan, Ritesh Shah, Pushpak Bhattacharyya, "The IIT Bombay HindiEnglish Translation System at WMT 2014"

[5] Xavier Glorot,Yoshua Bengio, "Understanding the difficulty of training deep feedforward neural networks"

[6] Caglar Gulcehre, Orhan Firat, Kelvin Xu, Kyunghyun Cho, Loic Barrault, Yoshua Bengio, "On Using Monolingual Corpora in Neural Machine Translation"

[7] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio, "Learning phrase representations using RNN encoder-decoder"

[8] Sepp Hochreiter and Jrgen Schmidhube, "Long short-term memory"

[9] Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, Geoffrey Hinton, "Grammar as a Foreign Language"

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

[11] Wang Ling, Isabel Trancoso, Chris Dyer, Alan W Black, "Character-based Neural Machine Translation"

[12] Rico Sennrich, Barry Haddow, Alexandra Birch, "Improving Neural Machine Translation Models with Monolingual Data"

[13] M. Auli, M. Galley, C. Quirk, and G. Zweig, "Joint language and translation modeling with recurrent neural networks"

[14] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, ”A neural probabilistic language model”

[15] Y. Bengio, P. Simard, and P. Frasconi, ”Learning long-term dependencies with gradient descent is difficult”

[16] R. Pascanu, T. Mikolov, and Y. Bengio, ”On the difficulty of training recurrent neural networks”

[17] K. Papineni, S. Roukos, T. Ward, and W. J. Zhu, ”BLEU: a method for automatic evaluation of machine translation”

[18] Thang Luong, Ilya Sutskever, Quoc V Le, Oriol Vinyals, and Wojciech Zaremba, ”Addressing the rare word problem in neural machine translation”

[19] M. Sundermeyer, R. Schluter, and H. Ney, ”LSTM neural networks for language modeling”

[20] Cho, K., van Merrienboer, B., Bahdanau, D., and Bengio, Y.,” On the properties of neural machine translation: Encoder-Decoder approaches”

[21] Hochreiter, S. and Schmidhuber, J., ”Long short-term memory”

[22] Kalchbrenner, N. and Blunsom, P., ”Recurrent continuous translation models”

[23] Christian Buck, Kenneth Heafield, and Bas van Ooyen, ”N-gram counts and language models from the common crawl”

[24] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean, ”Efficient estimation of word representations in vector space”

[25] Schuster, M. and Paliwal, K. K., ”Bidirectional recurrent neural networks”

[26] Omkar Dhariya, Shrikant Malviya and Uma Shanker Tiwary, ”A Hybrid Approach For Hindi-English Machine Translation”

[27] Understanding LSTM networks – colah’s blog

[28] An overview of gradient descent optimization algorithms, Sebastien Ruder

[29] Deep Learning slides of Prof. Mitesh Khapra, CSE, IIT-M