Machine Translation using Deep Learning

Submitted in partial fulfillment of the requirements

of the degree of

Bachelor of Engineering

by

SHUBHAM AGRAWAL	**05**
ADITYA BHAGWAT	**10**
ROHIT NAIR	**45**
DAELYN OLIVEIRA	**47**

Supervisor:

Prof. Uday Nayak

UNIVERSITY OF MUMBAI

Machine Translation using Deep Learning

Submitted in partial fulfillment of the requirements

of the degree of

Bachelor of Engineering

by

SHUBHAM AGRAWAL	05
ADITYA BHAGWAT	10
ROHIT NAIR	45
DAELYN OLIVEIRA	47

Supervisor:

Prof. Uday Nayak

Department of Information Technology

Don Bosco Institute of Technology

2018-2019

AFFILIATED TO

UNIVERSITY OF MUMBAI

DON BOSCO INSTITUTE OF TECHNOLOGY
Vidyavihar Station Road, Mumbai - 400070

Department of Information Technology

CERTIFICATE

This is to certify that the project entitled **“MACHINE TRANSLATION USING DEEP LEARNING“** is a bonafide work of

SHUBHAM AGRAWAL	**ROLL NUMBER 05**
ADITYA BHAGWAT	**ROLL NUMBER 10**
ROHIT NAIR	**ROLL NUMBER 45**
DAELYN OLIVEIRA	**ROLL NUMBER 47**

submitted to the University of Mumbai in partial fulfillment of the requirement for the award of the degree of **Undergraduate** in **Bachelor of Information Technology**

Date: **22/04/2019**

(Prof. Uday Nayak)
Supervisor

(Prof. Prasad Padalkar)
HOD, IT Department

(Dr. Prasanna Nambiar)
Principal

DON BOSCO INSTITUTE OF TECHNOLOGY

Vidyavihar Station Road, Mumbai - 400070

Department of Information Technology

Project Report Approval for B.E.

This project report entitled **“MACHINE TRANSLATION USING DEEP LEARNING“** by **Shubham Agrawal, Aditya Bhagwat, Rohit Nair, Daelyn Oliveira** is approved for the degree of **Bachelor of Engineering in Information Technology**

(Examiner’s Name and Signature)

1. ______________________________

2. ______________________________

(Supervisor’s Name and Signature)

1. ______________________________

(Chairman)

1. ______________________________

Date: 22/04/2019

Place: Kurla(West)

DON BOSCO INSTITUTE OF TECHNOLOGY
Vidyavihar Station Road, Mumbai - 400070
Department of Information Technology

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea / data / fact / source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

SHUBHAM AGRAWAL	05	(————————)
ADITYA BHAGWAT	10	(————————)
ROHIT NAIR	45	(————————)
DAELYN OLIVEIRA	47	(————————)

Date:22/04/2019

ABSTRACT

Neural Machine Translation using Deep Learning is a newly proposed approach to machine translation.The issue with the current translation system is that it does not translate idioms and phrases appropriately. The proposed translation system attempts to successfully translate English text to its appropriate Hindi text translation and vice versa using Recurrent Neural Networks using a novel neural network architecture which will replicate the higher accuracy levels achieved. Our project uses an encoder-decoder Long Short Term Memory (LSTM) model. The evaluation of the output text is denoted by Bilingual Evaluation Understudy (BLEU) score indicating the quality of text which has been machine-translated from one natural language to another. Our project works by training and testing the model on bilingual pre-processed and cleansed dataset. The skeleton of the project is developed on Google Colaboratory Notebooks. The Proposed system will consist of a simple Interface with Input text field and Output text field. Our project uses Keras frontend and Tensorflow backend executed on the Google Colaboratory. Our project can be used by an English or Hindi amateur over the world wide web.

Keywords: *Machine Translation, Neural Networks, Neural Machine Translation, Deep Learning, English-Hindi MT.*

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Statement

One of the most important limitations of the modern day translators is the inability to handle the translations of idioms and phrases of the source language. Due to this limitation our goal is to implement a web application based on Neural Machine Translation system which successfully translates English text to Hindi text using a special type of Recurrent Neural Network(RNN) called as Long Short Term Memory(LSTM) for encoding and decoding (with an attention mechanism) to achieve better accuracy.

1.2 Scope of the Project

Implementing a fully functioning system based on Machine Translation, to successfully generate high quality and appropriate Translation from input. Scope of our project limits us to handle sentences with length more than specified(20 words), inability to handle proper nouns(name, place, animal, thing) excluding the ones already in the data set and tackling words not present in our data set. The Deliverables provided by our project would be Algorithm, Deep learning Machine Translation Model, Desktop UI. The features included are English to Hindi Text Translation. Understanding the input English Text entered by the user and successfully translating the English text input to Hindi Text using our model.

- Requirements Gathering Phase
- Requirements Analysis Phase
- System Design phase

1.3 Current Scenario

Sr. No	Name	Platform	Feature
1	Google Translate	Cross-platform (Web application)	Translate Word documents, PDFs, and other file types in Google Translate. Download offline languages. Live visual translations
2	Bing Translator	Cross-platform (Web application)	Microsoft's linguistically informed statistical MT system. Personal universal translator that enables up to 500 people to have live, multi-device, multi-language, in person translated conversations.
3	Yandex Translate	Cross-platform (Web application)	Statistical and neural machine translation. Photo text translation feature. Voice input feature included.
4	Anusaaraka	Cross-platform (Web application)	Rule-based, deep parser based, paninian framework based. All programs and language data are free and open-source
5	IBM	Cross-platform (Web application)	Both rule-based and statistical models developed by IBM Research. Neural Machine Translation models available through the Watson Language Translator API for developers.

Table 1.1: Current Scenario

1.4 Need for the Proposed System

To successfully translate the English Text input to Hindi text using deep learning instead of statistical machine translation(traditional translation systems). To play around with the hyperparameters, training data size, number of training steps and compare the accuracy of the results for different combinations so as to achieve the optimum translation accuracy.

1.5 Summary of the Results / Task completed

(a) Completed Literature survey

(b) Downloaded the data sets

(c) Cleansing of data sets

(d) Finding vocabulary size and maximum sentence length of english and hindi text from the parallel corpus.

(e) Training

(f) Validation

(g) Finding the BLEU scores of the model.

We learnt to build a neural network with hidden layer, using forward propagation and backpropagation. We understood the necessity of hyperparameters like batch size, optimizers, learning rate, number of epochs, size of the dataset, plotting to check the fitting, etc., and how the interdependencies among them work and use it to train deep neural networks. Our translation system handles idioms and phrases and generates desired outputs.

Chapter 2

Review of Literature

2.1 Summary of the investigation in the published papers

Experiments On Different Recurrent Neural Networks For English-Hindi Machine Translation

In this paper [1], experiments using different neural network architectures employing Gated Recurrent Units, Long Short Term Memory Units and Attention Mechanism and report the results for each architecture. The Bi-directional LSTMs generally show better performance for compound sentences and larger context windows. They describe the motivation behind the choice of RNNs in detail. When they employ bidirectional LSTMs, they end up with two hidden states - one in the forward direction and one in the backward direction. This allows the network to learn from the text. Bi-directional LSTMs generally work the best, especially when complemented with the attention mechanism. Pruning was also done to remove special characters and hyperlinks from the sentences. They observe that their model is able to produce grammatically fluent translations, as opposed to traditional approaches. A bi-directional LSTM model with attention mechanism shows improvement over normal RNN's in both these aspects. They demonstrated results using this approach on a linguistically distant language pair En ! Hi and showed a substantial improvement in translation quality. They conclude that RNN perform well for the task of English-Hindi Machine Translation. The bidirectional LSTM units perform best, specially on compound sentences.

Machine Translation Using Deep Learning : A Survey

In this paper [2], the term Machine Translation is used in the sense of translation of one language to another, without any human improvement. In this approach Neural Machine Translation technique will be used to translate Japanese language into English language. Tanaka corpus will be used in this approach. Japanese will be translated into English using improved Recurrent Neural Network(RNN). It contains 1,50,000 sentences pairs and database is available on the internet. In this approach they have tried to get accurate translation from Japanese to English. They have used small group of data to train the system. They will apply this architecture to the large amount of data with efficient processing unit. In future they will try to translate accurately Japanese to Hindi. Graphics processing unit (GPU) is very good option for parallel processing and fast computation as compare to the CPU. GPU provides a better energy efficiency and archives substantially higher performance over CPUs. The experiment performed by them in the following steps:

1. First of all collect Japanese-English vocabulary data.
2. Create RNN for Encoding texts with help of python.
3. Train the system with languge corpus.
4. Translate Japanese language into English language.

Achieving Open Vocabulary Neural Machine Translation With Hybrid Word-Character Model

In this paper [4] , neural machine translation (NMT) has used quite restricted vocabularies, perhaps with a subsequent method to patch in unknown words. This paper presents a novel word character solution to achieving open vocabulary neural machine translation. Publishers build hybrid systems that translate mostly at the word level and consult the character components for rare words. Their character-level recurrent neural networks compute source word representations and recover unknown target words when needed. The advantage of such a mixed approach is that it is much faster and easier to train than character-based ones. Also it never produces unknown words as in the case of word based models. In training word-based NMT, they follow Luong et al. (2015a) to use the global attention mechanism together with similar hyperparameters. In word-based NMT, their source and target sequences to have a maximum length of 50 each, words that go beyond the boundary are ignored. Due to memory constraint in GPUs, they limit their source and target sequences to a maximum length of 150 each, i.e., they backpropagate through at most 300 timesteps from the decoder to the encoder. They have proposed a novel hybrid architecture that combines the strength of both word- and character-based models. Word-level models are faster to train and offer high-quality translation; whereas, character-level models help achieve the goal of open vocabulary NMT. Their best hybrid model has surpassed the performance of both the best word-based NMT system and the best non-neural model to establish a new state-of-the-art result for English-Czech translation with 20.7 BLEU. Moreover, they have succeeded in replacing the standard ¡unk¿ replacement technique in NMT with their character level components, yielding an improvement of +2.111.4 BLEU points.

Neural Language Models for Machine Translation

In this paper [3], they studied that deep NLMs with three or four layers outperform those with fewer layers in terms of both the inability to deal with and the translation quality. They combine various techniques to successfully train deep NLMs that jointly condition on both the source and target contexts. They use DARPA BOLT program in the Chinese-English bitext, with 11.1 million parallel sentences. They reserve 585 sentences for validation, i.e., choosing hyperparameters, and 1124 sentences for testing. They train their model for 4 epochs. For both Chinese and English words the vocabularies are limited to the top 40000 frequent. All hidden layers have 512 units each and embeddings are of size 256. Their training speed on a single Tesla K40 GPU device is about 1000 target words per second and it generally takes about 10-14 days to fully train a model. In contrast, reranking with deep NLMs of three or four layers are clearly better, yielding average improvements of 1.0 TER / 1.0 BLEU points over the baseline and 0.5 TER / 0.5 BLEU points over the system reranked with the 1-layer model, all of which are statisfically significant according to the test described. In this paper, they have bridged the gap that past work did not show, that is, neural language models with more than two layers can help improve translation quality. Their results confirm the trend reported in (Luong et al., 2015) that source conditioned perplexity strongly correlates with MT performance.

2.2 Comparison between the tools / models

2.2.1 Tools

2.2.1.1 Tensorflow

For our proposed system we are making use of Tensorflow which is an interface for expressing machine learning algorithms and implement it. In a tensorflow system we can express the computation in a heterogenous system in a wide variety of ways ranging from small devices like mobile phones or tablets to large scale devices with hundreds or thousands of machines or computational devices. Tensorflow can be used to express wide variety of algorithms which includes training the model or algorithms for deep neural networks such as RNN or CNN and is used for research purposes and deploying the machine learning systems into various fields such as cognitive science, artificial intelligence and other fields such as OCR(Optical Character Recognition) speech recognition, NLP, robotics and computer vision.

Features of TensorFlow:

1. Open Source
2. Library for numerical Computation
3. Distributed
4. Extensible
5. GPU Support

2.2.1.2 Keras

Keras is a powerful easy-to-use Python library for developing and evaluating deep learning models. It warps the efficient numerical computation libraries Microsoft Cognitive Toolkit (CNTK) and TensorFlow and allows you to define and train neural network models in a few short lines of code. It was developed with a focus on enabling fast experimentation . Being able to go from idea to result with the least possible delay is key to doing good research.

Deep learning models are developed and evaluated using easy to use Python library ”Keras”. Wrapping the efficient numerical computation libraries CNTK and Tensorflow as well as defining and training neural networks in a few short lines. Keras helps you to efficiently run your algorithms in a short span of time. Keras helps you to do good research and helps you to implement the model with the least possible delay migrating from an idea

2.2.1.3 Pytorch

Very often, various python libraries are developed that have the potential of changing the landscape in the field of deep learning. One such library is PyTorch, which is a python based library built to provide flexibility as a deep

learning development platform. The workflow of PyTorch is very close to python's scientific computing library – numpy. PyTorch uses different back-ends for CPU, GPU and for various functional features instead of using a single back-end. Using different backends makes it very easy to deploy PyTorch on constrained systems. PyTorch is deeply integrated with the C++ code, and it shares some C++ backend with the deep learning framework, Torch which allows users to program in C/C++ by using an extension API based on cFFI for Python and compiled for CPU or GPU operation. This feature has extended the PyTorch usage for new and experimental use cases which makes them a preferable choice for research use.
Benefits of using Pytorch to build deep learning models are:

- Easy to use API - It is one of the more easily learnable python libraries.
- Python support – As mentioned above, PyTorch integrates smoothly with the python data science stack. It is so similar to numpy that one might not even notice the difference.
- Dynamic computation graphs – Instead of predefined graphs with specific functionalities, PyTorch provides a framework for us to build computational graphs , and even change them during runtime. This is beneficial in situations where we don't know how much memory is going to be required for creating a neural network.

2.2.2 Recurrent Neural Network (RNN)

RNN or Recurrent Neural Network is a class of artificial neural network which uses the output of hidden state produced by previous input and current input to produce current output. In other words, it uses memory for producing the desired output. RNN remembers what it knows from previous input using a simple loop. This loop takes the information from previous timestamp and adds it to the input of current timestamp.

This allows it to exhibit dynamic temporal behavior for a time sequence. Unlike feed forward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition. Recurrent Neural Network comes into the picture when any model needs context to be able to provide the output based on the input. Sometimes the context is the single most important thing for the model to predict the most appropriate output.

The following are the few applications of the RNN:

1. Next word prediction.
2. Music composition.
3. Image captioning
4. Speech recognition
5. Time series anomaly detection

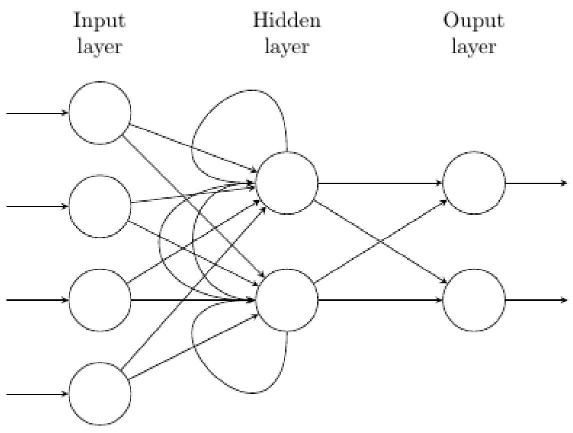


Figure 2.1: Recurrent Neural Network

2.3 Algorithm(s) with example

2.3.1 Statistical Machine Translation for Text-to-text

Statistical machine translation (SMT) is a machine translation where the text are translated using parameters which are derived using bilingual text corpora. The statistical approach contrasts with the rule-based approaches to machine translation as well as with example-based machine translation. Training of millions of words training data is needed for high quality SMT results which is costly compared to modern way of approach. So SMT used to translate with a limited amount of words which created results in a less optimal way.

In Statistical machine translation (SMT) it learns to translate the human translations by analzying existing translations which is also known as bilingual text corpora. It differs from the Rules Based Machine Translation (RBMT) which uses an approach of word based translation which uses more computational power and has less efficiency, modern SMT systems are based on phrased based translation and rearranges the translations using overlap phrases. Phrase-based translation aims to reduce the restrictions of word-based translation by allowing it to translate the whole word where even the length of the word may differ.

The shortcomings of SMT are as follows:

- Corpus creation are costly.
- Errors are difficult to detect and fix.
- Results may have less fluency of words that masks translation problems.

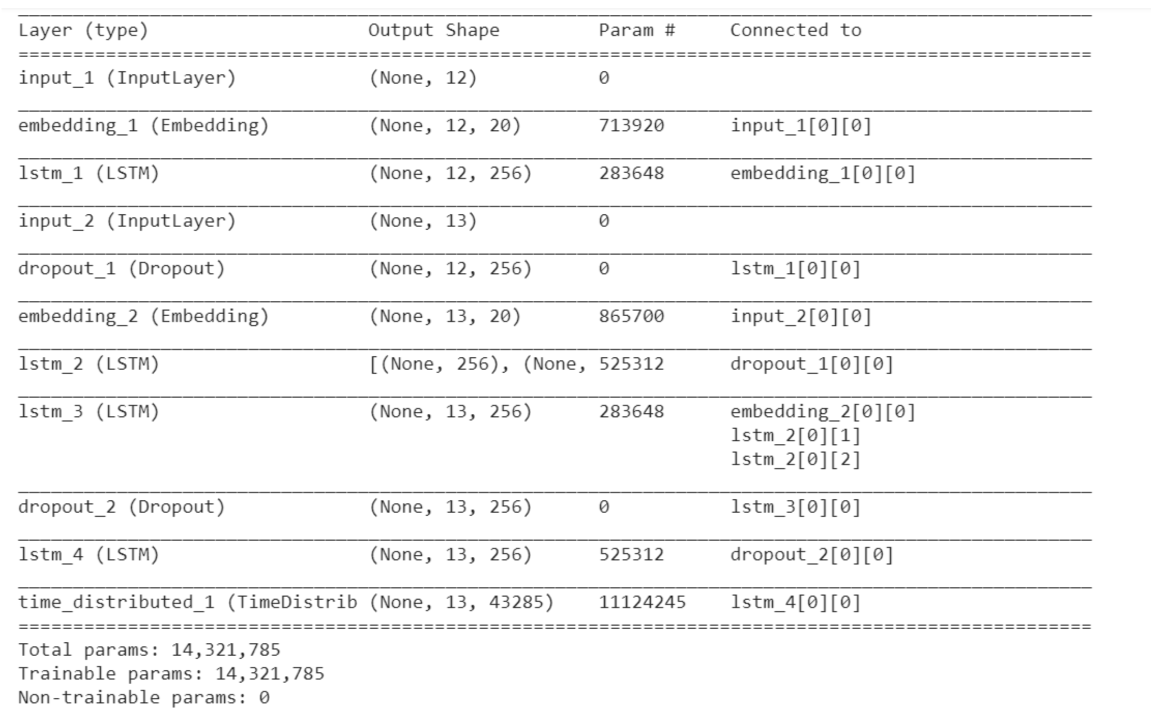
2.3.2 Model 2 : Recurrent Neural Network for Text-to-text

In feed forward neural network length of input layer is fixed to the length is the input sentence. So, to deal with these types of variable-length input and output, RNN comes into action. Whenever a single sample is fed into a feed-forward neural network, the network's internal state, or the activation of the hidden units, is computed from scratch and is not influenced by the state computed from the previous sample. The RNN's thus help in converting the input sequence to a fixed size feature vector that encodes primarily the information which is crucial for translation from the input sentence, and ignores the irrelevant information.

Recurrent Neural Networks (RNN) are a powerful and robust type of neural networks and belong to the most promising algorithms out there at the moment because they are the only ones with an internal memory. Because of their internal memory, RNN's are able to remember important things about the input they received, which enables them to be very precise in predicting what's coming next. In a RNN, the information cycles through a loop. When it has to make a decision, it considers the current input and also what it has learned from the inputs it received previously. Long Short Term Memory (LSTM) units are a type of RNNs which are very good at preserving information through time-steps over a period of time. In an LSTM we have three gates:-

1. Input gate: This gate determine whether or not to let new input in
2. Forget gate: This gate deletes the information because it isn't important

3. Output gate: This gate gives the output of the current time step.



```
________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
================================================================================================
input_1 (InputLayer)            (None, 12)           0
________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 12, 20)       713920      input_1[0][0]
________________________________________________________________________________________________
lstm_1 (LSTM)                   (None, 12, 256)      283648      embedding_1[0][0]
________________________________________________________________________________________________
input_2 (InputLayer)            (None, 13)           0
________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 12, 256)      0           lstm_1[0][0]
________________________________________________________________________________________________
embedding_2 (Embedding)         (None, 13, 20)       865700      input_2[0][0]
________________________________________________________________________________________________
lstm_2 (LSTM)                   [(None, 256), (None, 525312      dropout_1[0][0]
________________________________________________________________________________________________
lstm_3 (LSTM)                   (None, 13, 256)      283648      embedding_2[0][0]
                                                                 lstm_2[0][1]
                                                                 lstm_2[0][2]
________________________________________________________________________________________________
dropout_2 (Dropout)             (None, 13, 256)      0           lstm_3[0][0]
________________________________________________________________________________________________
lstm_4 (LSTM)                   (None, 13, 256)      525312      dropout_2[0][0]
________________________________________________________________________________________________
time_distributed_1 (TimeDistrib (None, 13, 43285)    11124245    lstm_4[0][0]
================================================================================================
Total params: 14,321,785
Trainable params: 14,321,785
Non-trainable params: 0
```

Figure 2.2: Model Summary

Model Architecture

- Text Vectorization (One-hot encoding)
- InputLayer : Input :(None,12) , Output :(None,12)
- Embedding : Input :(None,12) , Output :(None,12,20)
- LSTM : Input :(None,12,20) , Output :[(None,256), (None,256), (None,256)]
- InputLayer : Input : (None,13) , Output : (None,13)
- Embedding : Input :(None,13) , Output :(None,13,20)
- LSTM : Input : [(None,13,20), (None,256), (None,256)] , Output :(None, 13, 256)
- TimeDistributed : Input : (None, 13, 256) , Output : (None, 13, 43213)

2.3.3 Gradient descent optimization algorithms

SGD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in one dimension than in another, which are common around local optima.

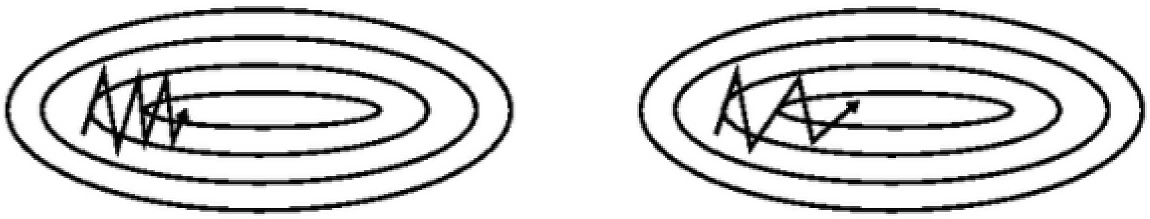


Figure 2.3: Left: SGD without momentum. **Right:** SGD with momentum.

In these scenarios, SGD oscillates across the slopes of the ravine while only making hesitant progress along the bottom towards the local optimum as in 2.3(Left).

Momentum

Momentum is a method that helps accelerate SGD in the relevant direction and dampens oscillations as can be seen in 2.4(Right). It does this by adding a fraction of the update vector of the past time step to the current update vector:

$$\upsilon_t = \beta\upsilon_{t-1} + (1-\beta)\nabla_\theta J(\theta)$$

$$\theta = \theta - \alpha\upsilon_t$$

2.3.4 Adam

Adaptive Moment Estimation(Adam) [11] is another method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients υ_t like Adadelta [**?**] and RMSprop[1], Adam also keeps an exponentially decaying average of past gradients m_t similar to momentum. Whereas momentum can be seen as a ball running down a slope, Adam behaves like a heavy ball with friction, which thus prefers flat minima in the error surface. We compute the decaying averages of past and past squared gradients m_t and υ_t respectively as follows:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)g_t$$

$$\upsilon_t = \beta_2\upsilon_{t-1} + (1-\beta_2){g_t}^2$$

and m_t and υ_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 υ_t are initialized as vectors of 0's, the authors of Adam observe that they are biased towards zero, especially during the initial

[1]`https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`

time steps, and especially when the decay rates are small (i.e. β_1 and β_2 are close to 1).
They counteract these biases by computing bias-corrected first and second moment estimates:

$$\hat{m}_t = \frac{m_t}{1-{\beta_1}^t}$$

$$\hat{\upsilon}_t = \frac{\upsilon_t}{1-{\beta_2}^t}$$

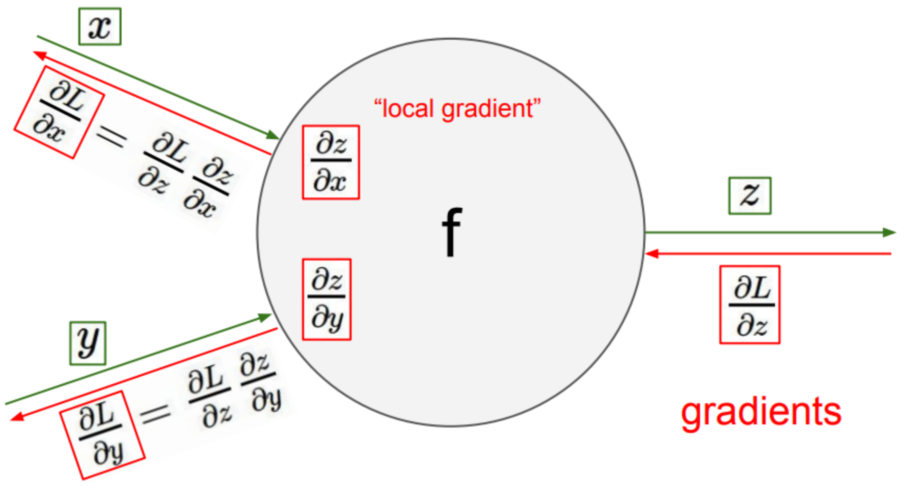
They then use these to update the parameters just as we have seen in Adadelta and RMSprop, which yields the Adam update rule:

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\hat{\upsilon}_t}+\varepsilon}\hat{m}_t$$

The authors propose default values of 0.9 for β_1, 0.999 for β_2, and 10^{-8} for ε. They show empirically that Adam works well in practice and compares favorably to other adaptive learning-method algorithms.

2.3.5 Backpropagation

Backpropagation is a method used in artificial neural networks to calculate a gradient that is needed in the calculation of the weights to be used in the network. Backpropagation is shorthand for ”the backward propagation of errors,” since an error is computed at the output and distributed backwards throughout the network’s layers. It is commonly used to train deep neural networks.



x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradients

Figure 2.4: Backpropagation

Backpropagation is a generalization of the delta rule to multi-layered feedforward networks, made possible by using the chain rule to iteratively compute gradients for each layer.

In the context of learning, backpropagation is commonly used by the gradient descent optimization algorithm to adjust the weight of neurons by calculating the gradient of the loss function.

2.3.6 Beam search

Beam Search is where we take top k predictions, k being the beam width, feed them again in the model and then sort them using the probabilities returned by the model. So, the list will always contain the top k predictions. In the end, we take the one with the highest probability and go through it till we encounter ¡end¿ or reach the maximum caption length.

Chapter 3

Analysis and Design

3.1 Methodology / Procedure adopted

A typical life cycle was divided into two phases:

Phase 1:

The fundamentals of Deep learning in context to the model was the main task in the first phase that we intended to learn and understand . In this step we intended to get to know all of the available resources and technologies that can be used in our project for the model to be implemented. Literature survey and downloading of the datasets needed to train the model was part of this phase.

Phase 2:

We shift from the details and theory of deep learning algorithms to implementations, this involves an Agile like model. We build and improvise until we get an optimally performing model. In an iterative fashion we take one module at a time. At each step of project development requirements of customers are checked upon. A rapid delivery of the products is done. Importance is given to the stake holder's and customer's requirements.

When shifting from one module to other these phases can overlap. Literature survey is carried out throughout the project as per the requirements. We have weekly assignments where we try to read about some technology or resources relevant to our project. Also a weekly meeting is conducted either in person or on a shared conference call. We use GitHub, Google Docs for collaboration. By setting milestones and checking them at each stage the progress of the project is measured

3.2 Analysis

Requirement gathering of our system was done in the initial stages. We came to a conclusion that we would use Google CoLab to implement our model as it has its own GPU which can be used to train the model using the datasets.

Some of the requirements were modified and looked into as suggested by our guide and the panel.

3.2.1 Performance Requirements

Response Time
The application should take approximately 10-20 seconds to recognize the English text and give the appropriate Hindi text translation output.

Capacity
The system is capable of handling english sentence length of maximum 12 words.

3.2.2 Software / System Requirement Specification - IEEE format

User Interfaces
Our project consists of a simple GUI of web application with Input text field and an Output text field where user can translate the English text to Hindi text.

Hardware Interfaces
GPU: Nvidia GEFORCE 1080 Ti The project is going to be implemented on Google CoLab which has its own GPU that can be used to train the model using the datasets.

Software Interfaces
Front End: Keras
Back End: Tensorflow
Programming Language: Python 3.x

TensorFlow is an open source software library for high performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. Originally developed by researchers and engineers from the Google Brain team within Google's AI organization, it comes with strong support for machine learning and deep learning and the flexible numerical computation core is used across many other scientific domains.

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

Use Keras if you need a deep learning library that:

1. Allows for easy and fast prototyping (through user friendliness, modularity and extensibility)
2. Supports both convolutional networks and recurrent networks, as well as combinations of the two.
3. Runs seamlessly on CPU and GPU.

Flask is a web framework. This means flask provides with tools, libraries and technologies that allow to build a web application. This web application can be some web pages, a blog, a wiki or go as big as a web-based calendar application or a commercial website.

Flask is part of the categories of the micro-framework. Micro-framework are normally framework with little to no dependencies to external libraries. This has pros and cons. Pros would be that the framework is light, there are little dependency to update and watch for security bugs, cons is that some time you will have to do more work by yourself or increase yourself the list of dependencies by adding plugins.

Communications Interfaces

The requirements associated with communications functions required by our project are cloud storage where the training datasets will be uploaded from Google CoLab and verifying the text with the files automatically through web application.

3.3 Proposed System

Architecture/Design Model

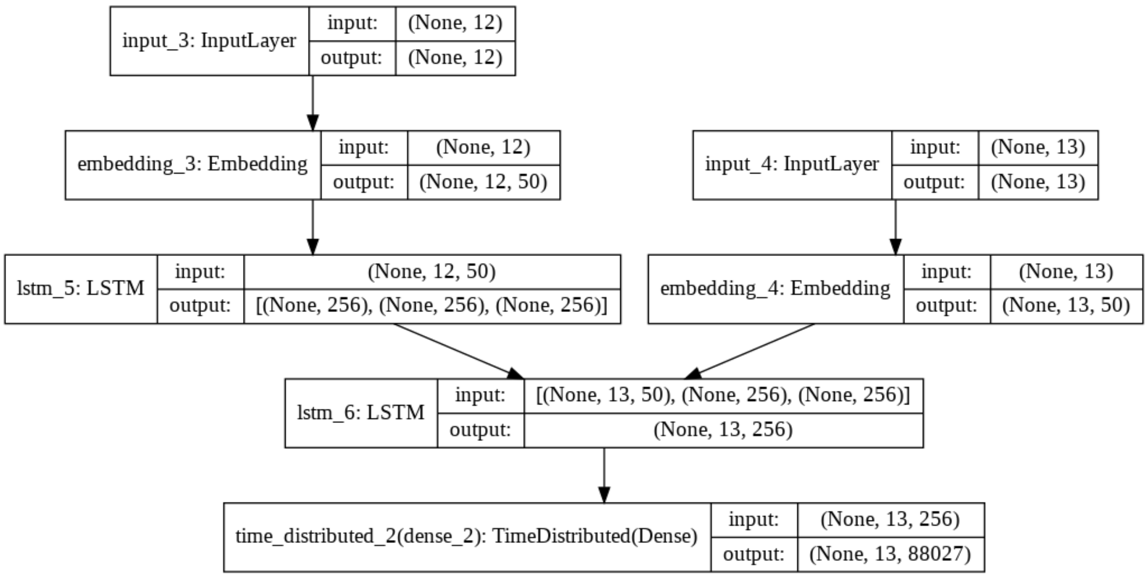

input_3: InputLayer | input: (None, 12) | output: (None, 12)
embedding_3: Embedding | input: (None, 12) | output: (None, 12, 50)
input_4: InputLayer | input: (None, 13) | output: (None, 13)
lstm_5: LSTM | input: (None, 12, 50) | output: [(None, 256), (None, 256), (None, 256)]
embedding_4: Embedding | input: (None, 13) | output: (None, 13, 50)
lstm_6: LSTM | input: [(None, 13, 50), (None, 256), (None, 256)] | output: (None, 13, 256)
time_distributed_2(dense_2): TimeDistributed(Dense) | input: (None, 13, 256) | output: (None, 13, 88027)

Figure 3.1: Architecture/Block Diagram

In Figure 3.1 we have created a novel architecture for the proposed system to translate from English to Hindi text. In the first Input layer an English text is taken as an input and is passed to the embedding layer. The embedding layer creates an array of the word vectors for each of the English words which is being mapped from the English tokenizer . The maximum length of the English sentence is 12 which is being embedded into a 50 length vector representation. The output from the embedding layer is being passed to LSTM layer where the sentence is being encoded. There are 256 units used in the encoding layer is which is being passed to the decoder. At the same time the English encoded sentence is passed to the to the decoder there is an another layer being passed to the decoder. We take a second input for the system which is an Hindi text. The second input text is being passed to an embedding layer where the all unique words are mapped to the Hindi vocabulary and we get an array of all the words mapping it to Hindi tokenizer.

The network takes three inputs, X_t is the input of the current time step, h_t-1 is the output from the previous LSTM unit and C_t-1 is the "memory" of the previous unit, which is the most important input. h_t is the output of the current network. C_t is the memory of the current unit. This single unit makes decision by considering the current input, previous output and previous memory. And it generates a new output and alters its memory. On the LSTM diagram, the top

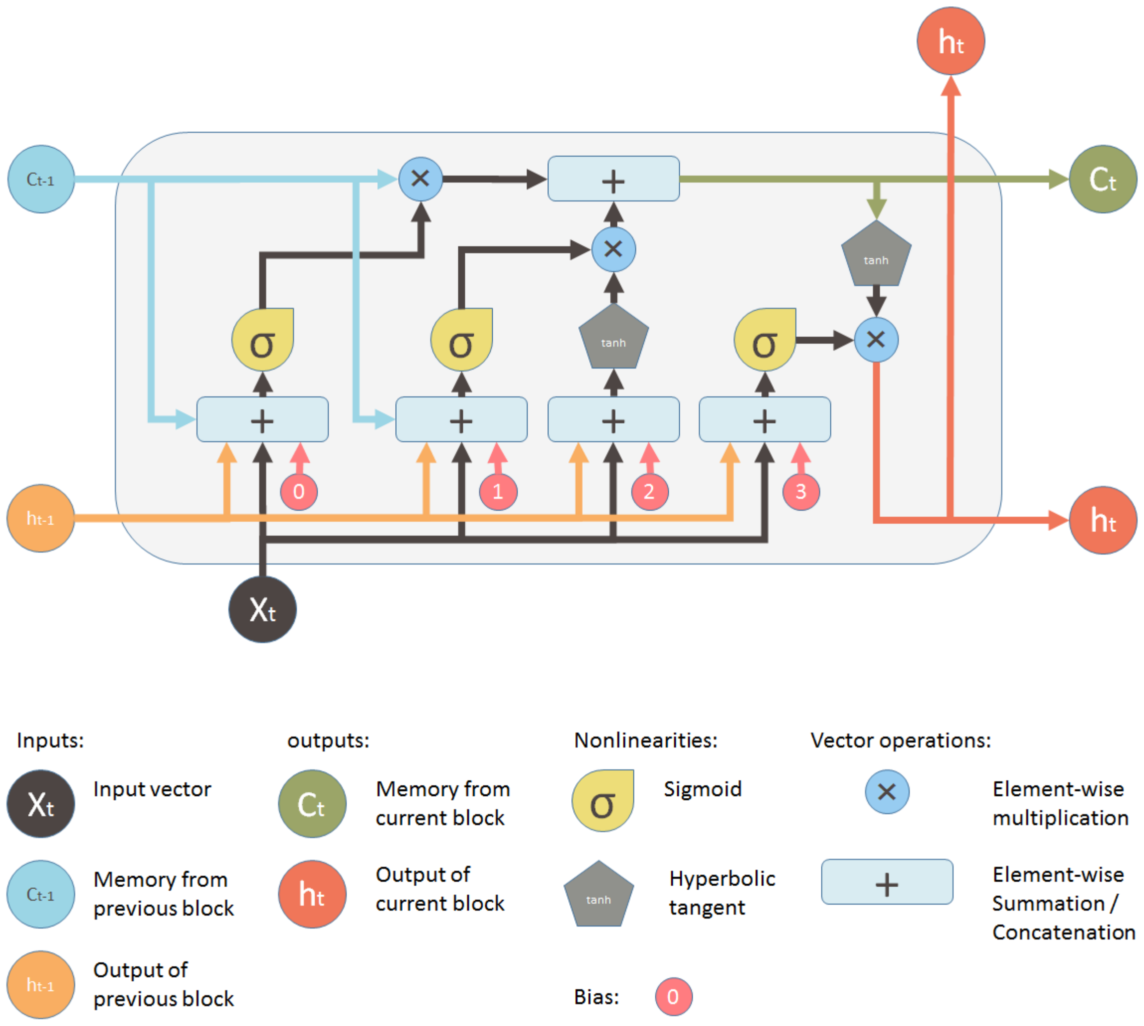

ht
Ct-1
Ct
tanh
σ
tanh
0
1
2
3
ht-1
ht
Xt
Inputs:
Xt Input vector
Ct-1 Memory from previous block
ht-1 Output of previous block
outputs:
Ct Memory from current block
ht Output of current block
Nonlinearities:
σ Sigmoid
tanh Hyperbolic tangent
Bias: 0
Vector operations:
× Element-wise multiplication
+ Element-wise Summation / Concatenation

Figure 3.2: Working of LSTM

“pipe” is the memory pipe. The input is the old memory (a vector). The first valve is called the forget valve. Now the second valve is called the new memory valve. Both are controlled by a one layer simple neural network that takes the same inputs as the forget valve. And finally, we need to generate the output for this LSTM unit. This step has an output valve that is controlled by the new memory, the previous output h_t-1, the input X_t and a bias vector. This valve controls how much new memory should output to the next LSTM unit.

The embedding layer is being is passed to the decoder along with the English encoded sentence. On the decoder layer when English encoded sentence is being decoded each of the English sentence is being mapped to its corresponding Hindi sentence through which the training of the model is efficient. The system learns better in this way. The output from the decoder layer is being passed on the Time Distributed Dense layer which is more of a wrapper layer in which the Hindi sentence as an output in the form of a matrix where the rows of the matrix represents the maximum length of the sentence and the columns represent the vocabulary size of Hindi

The drawback of LSTM can be overcome by attention mechanism, attention is simply a vector, normally the outputs of dense layer using softmax function. Before Attention mechanism, translation relies on reading a complete sentence and compress all information into a fixed-length vector, as you can imagine, a sentence with hundreds of words represented by several words will surely lead to information loss or inadequate translation, etc.

3.3.1 Design Details

1. Sequence Diagram

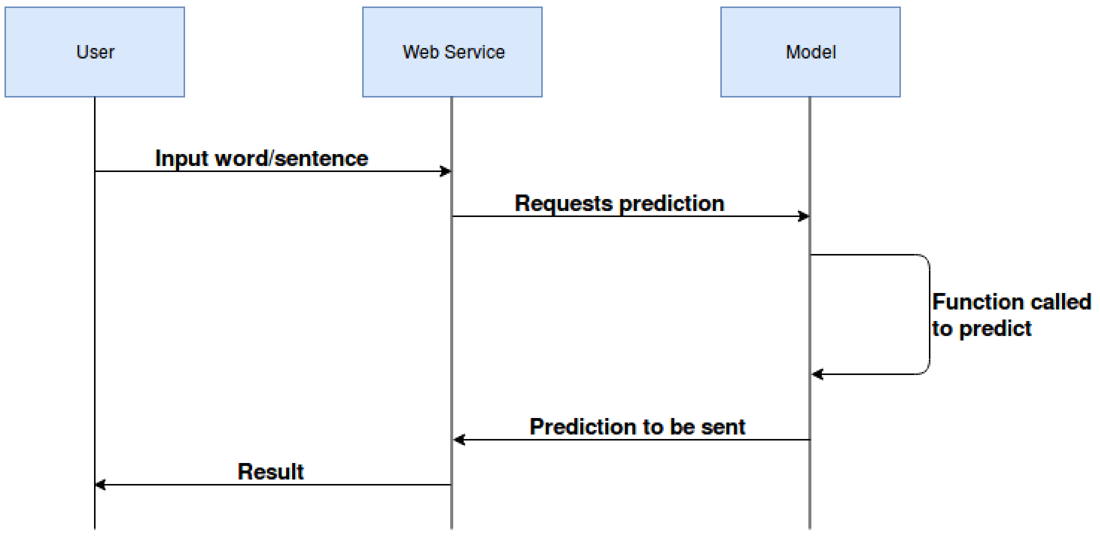


Figure 3.3: Sequence Diagram

The Encoder-Decoder architecture with recurrent neural networks has become an effective and standard approach for both neural machine translation (NMT) and sequence-to-sequence prediction in general. The key benefits of the approach are the ability to train a single end-to-end model directly on source and target sentences and the ability to handle variable length input and output sequences of text.

in Figure 3.3 The translation task was processed one sentence at a time, and an end-of-sequence token was added to the end of output sequences during training to signify the end of the translated sequence. This allowed the model to be capable of predicting variable length output sequences.

The word embeddings are first computed and given as input to the one hot word vectors, I.e. English text input. Then the English chunks are translated to Hindi text. Depending on the length of our input sentence this process is repeated. Finally the Hindi words are rearranged in a grammatical manner so that the output sentence makes sense and is if not perfect atleast a close translation of the input sentence. Our input sentence could be of different lengths for which we must have a stop token which is basically an extra ‘word’ in our training data e.g a full stop which will indicate that we have reached the end of a sentence. The decides RNN node will start producing output vectors once we reach the stop token.

2. Encoder Decoder

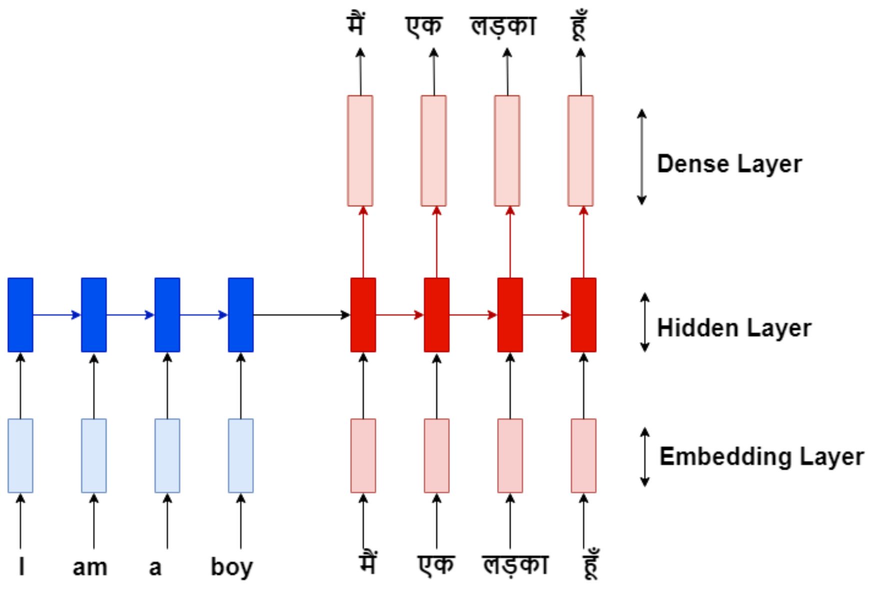


Figure 3.4: Encoder Decoder

The Figure 3.4 represents the model shown in this architecture : one is for reading and encoding the input sentence into fixed length vector, other is for outputting the predicted sequence by decoding the fixed length vector.This gives the architecture its name of Encoder-Decoder LSTM.

The Encoder-Decoder LSTM was developed for natural language processing problems. It illustrated the state-of-art performance in the text translation area. The use of fixed sized internal representation in the heart of the model which reads the input and output sequences to and from is the innovation of this architecture. Thus the method may be referred to as sequence embedding. Each word of the English sentence is passed on to an embedding layer where the each sentence is represented in a form of vectors. The embedding layer is then passed to the hidden layer.

The hidden layer is like a black box where both encoding and decoding of the sentence is done. In this layer both English and Hindi sentences are passed to the encoding layer where English the sentences are encoded. In the second part of the hidden layer i.e decoding layer each of the encoded English sentences are decoded and the corresponding Hindi sentences are mapped to the English sentences so that the system learns and understands better. The LSTM take cares of all the weights while encoding and decoding the sentences.

According to the above figure, blue represents encoder and red represents decoder, the last part of the encoder-decoder model is the dense layer which finally gives the output for the English sentence. It represents the output in form of a matrix where rows are length of Hindi sentence and columns are Hindi vocab size

Chapter 4

Implementation And Testing

4.1 Implementation plan for Semester 8

Gantt Chart

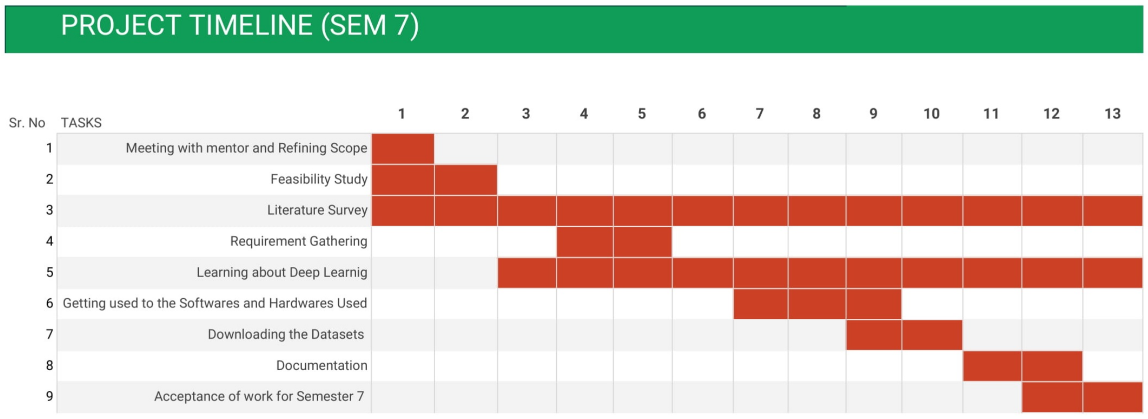
A project timeline (sometimes called a Gantt Chart) visualizes project activities. The scale of a project timeline is a calendar. The duration of activities is represented by bars or process arrows, whereas singular events are represented by milestones.

Implementation Plan for semester VIII

The development of the project was split over two semesters. The first semester was mainly utilized to understand the various machine translation papers and literature surveys were conducted to identify different designs for the model as well different working models and the different ways to implement it and the different evaluation methodologies [6].

Using the information obtained in this process, the entire scope of the project was fine tuned to focus on translating from English to Hindi text as well defining the various hyper parameters for the model. A novel architecture was created for the translation along with an encoder decoder model and finally calculating the Bilingual Evaluation Understudy(BLEU) [9] for the model to check the accuracy of the model. Followed by this, a web interface was created for the users to translate the text they wanted and the model will translate the text from English to Hindi.

4.1.1 Implementation plan for semester VII

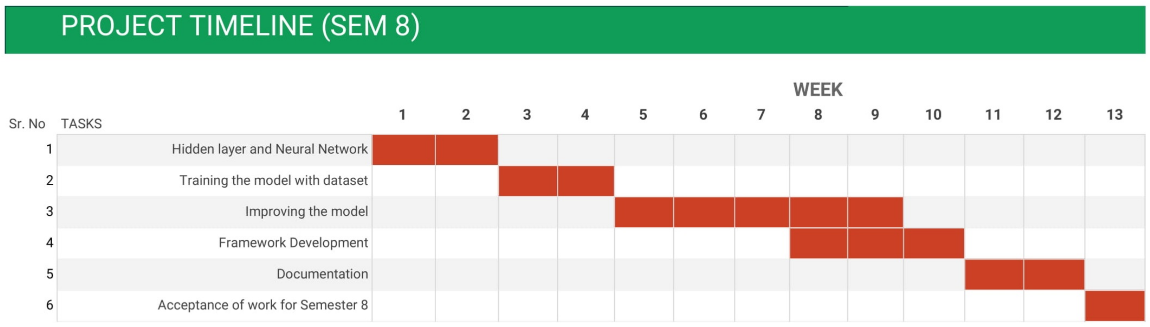


PROJECT TIMELINE (SEM 7)

Sr. No	TASKS	1	2	3	4	5	6	7	8	9	10	11	12	13
1	Meeting with mentor and Refining Scope	🟥												
2	Feasibility Study	🟥	🟥											
3	Literature Survey	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
4	Requirement Gathering				🟥	🟥								
5	Learning about Deep Learnig			🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
6	Getting used to the Softwares and Hardwares Used							🟥	🟥	🟥				
7	Downloading the Datasets									🟥	🟥			
8	Documentation											🟥	🟥	
9	Acceptance of work for Semester 7												🟥	🟥

Figure 4.1: Timeline Chart for Semester 7

4.1.2 Implementation plan for semester VIII



PROJECT TIMELINE (SEM 8)

		WEEK												
Sr. No	TASKS	1	2	3	4	5	6	7	8	9	10	11	12	13
1	Hidden layer and Neural Network	🟥	🟥											
2	Training the model with dataset			🟥	🟥									
3	Improving the model					🟥	🟥	🟥	🟥	🟥				
4	Framework Development								🟥	🟥	🟥			
5	Documentation											🟥	🟥	
6	Acceptance of work for Semester 8													🟥

Figure 4.2: Timeline Chart for Semester 8

4.2 Task Completion

4.2.1 Identify implemented methods for machine translation

Our first goal was to study about machine translation systems and the to get knowledge about deep learning to implement the efficient translation systems. We accomplished this by studying various and different systems as well as learning courses. We also researched about the domain by reading a few published papers.

4.2.2 Feasibility study

The models studied showed varying results in terms of accuracy. The various models defined use different model architecture and layers. Some of the models studied include unidirectional RNN, embedding architecture, bidirectional RNNs and encoder-decoder mechanism. In a RNN we use source-target pairs and try to create a neural network from the vocabulary and dictionary developed. RNNs basically arrange the weights and show the most nearest translation of the input word. As a development to RNNs this embedding model uses input-output vector embeddings as opposed to the pairs. An embedding is a vector representation of the words considered as the vocabulary, that is close to similar words in n-dimensional space wherein the n represents the size of the embedding vectors. Bidirectional RNNs, as the name goes provide forward as well as backward network which when used along with proper memory units stores temporal data slices which helps in predicting what will precede a certain set of input as well as what must succeed it. The encoder decoder model uses one hot word embeddings which are encoded in a block and mapped to its desired output in another block whose output is fed into the decoder to generate the appropriate output. Such models use special memory units like LSTM (Long Short Term Memory each of them improving the accuracy in its own fashion.

4.2.3 Data set

Training data set is the largest of three of them, while test data functions as seal of approval and you don't need to use till the end of the development. Our data set consists of wide variety of sentences including almost all of the words of oxford dictionary as well as corpus from different websites.

4.2.4 Training

The training data set is the actual data set used to train the model for performing various actions. This is the actual data the ongoing development process models learn with various API and algorithm to train the machine to work automatically.

4.2.5 Testing

This is the data typically used to provide an unbiased evaluation of the final that are completed and fit on the training data set. Actually, such data is used for testing the model whether it is responding or working appropriately or not.

4.2.6 User Interface

UI is designed using Flask framework. There is an input field which takes English text as input and an output field which gives the appropriate Hindi translation as output also a translate button.

4.2.7 Software components

- Google CoLab
- Flask

4.3 Model Implementation

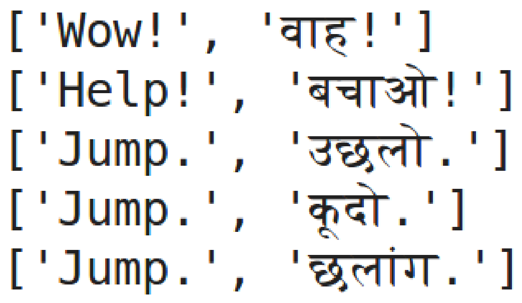
4.3.1 Data Exploration

For training our model we are using English-Hindi Corpus. We downloaded the open source datasets from different websites like:

Sr. No	No. of pairs	Description
1	2,831	ManyThings.org is one of the popular website which provides bilingual datasets for almost every majorly spoken language around the world. Downloaded dataset is the clean dataset which doesn't contain any punctuation.
2	435,620	Academia [10] is a growing collection of translated texts from the web. The community provides with a publicly available parallel corpus.
3	280	To make our translator able to translate the actual meaning of english idioms phrases in hindi we also add the english-hindi idioms and phrases dataset from the website.
Total	438,731	

Table 4.1: Dataset

After this, we combine all the datasets to make our final dataset which contains 438,731 pairs of English-Hindi words and sentences, one pair per line with a tab separating the language. In our project, we are loading the dataset in the memory with encoding UTF-8.

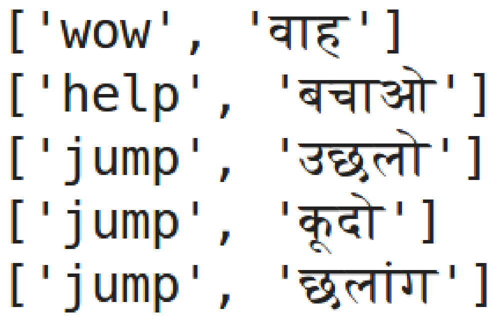


```
['Wow!', 'वाह!']
['Help!', 'बचाओ!']
['Jump.', 'उच्छलो.']
['Jump.', 'कूदो.']
['Jump.', 'छलांग.']
```

Figure 4.3: Before Cleaning

Each line contains a single pair of phrases, first English and then Hindi, separated by a tab character. We have to split the loaded text first by line and then by phrase. Then we clean each sentence. The specific cleaning operations we will perform are as follows:

1. Remove all punctuation from English and Hindi sentences.
2. All English sentences must be lowercase.
3. All the words in the English sentences must be alphabetic.
4. Remove any remaining tokens that are not alphabetic.



```
['wow', 'वाह']
['help', 'बचाओ']
['jump', 'उछलो']
['jump', 'कूदो']
['jump', 'छलांग']
```

Figure 4.4: After Cleaning

We also restrict the English and Hindi sentences in the dataset to a max length of 12 words only. After doing all this operation we left only 212,488 pairs of English-Hindi sentences. And load this clean dataset into the memory. After cleaning operation we create 3 pickle files which are : train.pkl val.pkl test.pkl

Running the example creates three new files:

Train.pkl:
The sample of data used to fit the model. The dataset that we use to train the model i.e weights and biases in the case of Neural Network. The model sees and learns from this data. We used 210,488 pairs of English-Hindi to train the model.

Val.pkl:
The validation set is used to evaluate a given model, but this is for frequent evaluation.The data sample which is used by the model for training the dataset helps the model to have an unbiased evaluation of the model along with adjusting the model's hyperparameters. The accuracy of the model increases by using a validation dataset on the training dataset which is incorporated during the model

configuration to evaluate the model. We used 2000 pairs of English-Hindi to validate the model.

Test.pkl:
The Test dataset provides the gold standard used to evaluate the model. Once a model is completely trained, it can be used. The test dataset shows the accuracy of the prediction for the given input for the model which is trained earlier. We used 2000 pairs of English-Hindi to test the model.

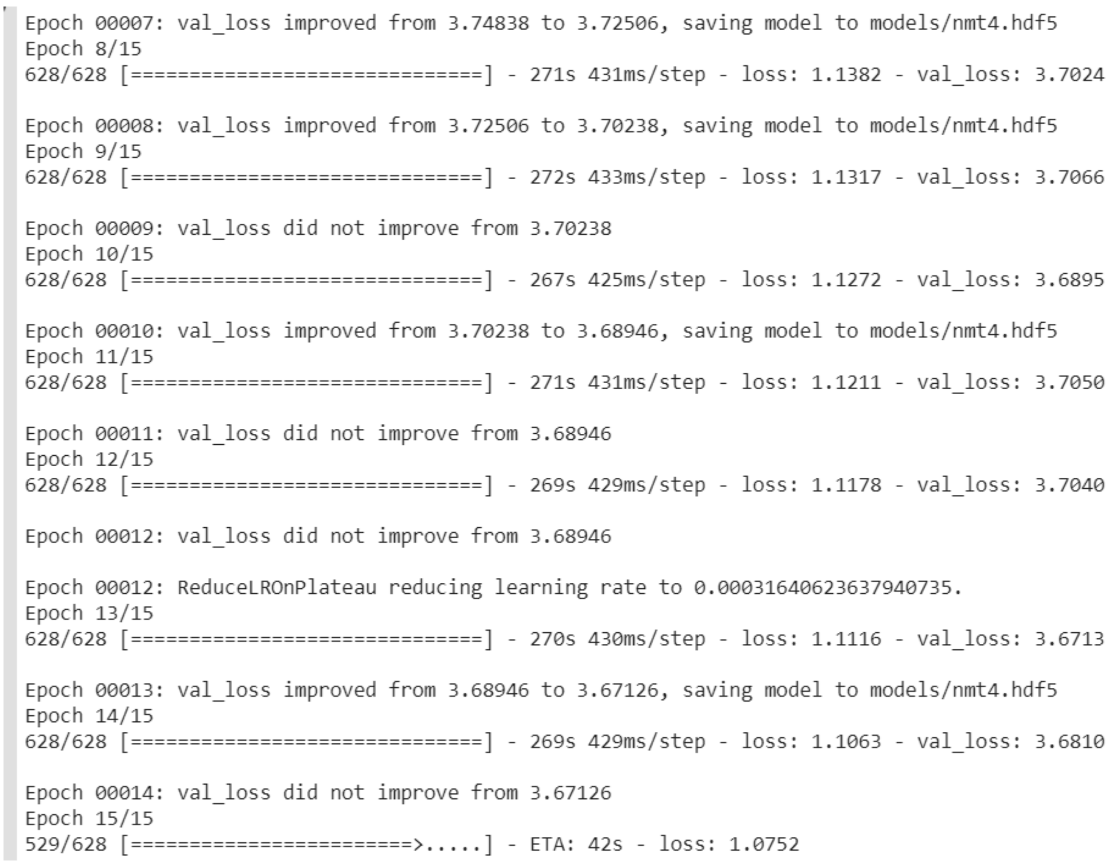
4.3.2 Model Training

In order to find the vocabulary of a language , we tokenize the English sentences from the train split so as to find the maximum word index and increment it by one. The process of the tokenization is a text tokenization utility class. This class allows to vectorize a text corpus by turning each text into a vector where the sentences are split into words and each word now being treated as a separate entity. Vocabulary size helps to find the amount of vectorization required.

Another important step prior to training is to find the maximum length sentence in terms of the number of words for both the languages. This helps to set an upper limit on the size of the vectors to be created in the future for training. This is done by ordering the length of every sentence of Hindi(and English) and then locating the maximum.

It is necessary to one-hot encode the tokenizers developed before moving ahead. After this encoding, the blank spaces in the tokenizers are handled by replacing them with a special character known to define the end of the statement of the tokenizer at hand. Pad_sequences is the keras.preprocessing class used to pad the sequences from the 'post'.

1. **ReduceLrOnPlateau**:-If the model is near a minimum and the learning rate is too high, then the model will circle around that minimum without ever reaching it. This callback will allow us to reduce the learning rate when the validation loss stops improving, allowing us to reach the optimal point.



```
Epoch 00007: val_loss improved from 3.74838 to 3.72506, saving model to models/nmt4.hdf5
Epoch 8/15
628/628 [==============================] - 271s 431ms/step - loss: 1.1382 - val_loss: 3.7024

Epoch 00008: val_loss improved from 3.72506 to 3.70238, saving model to models/nmt4.hdf5
Epoch 9/15
628/628 [==============================] - 272s 433ms/step - loss: 1.1317 - val_loss: 3.7066

Epoch 00009: val_loss did not improve from 3.70238
Epoch 10/15
628/628 [==============================] - 267s 425ms/step - loss: 1.1272 - val_loss: 3.6895

Epoch 00010: val_loss improved from 3.70238 to 3.68946, saving model to models/nmt4.hdf5
Epoch 11/15
628/628 [==============================] - 271s 431ms/step - loss: 1.1211 - val_loss: 3.7050

Epoch 00011: val_loss did not improve from 3.68946
Epoch 12/15
628/628 [==============================] - 269s 429ms/step - loss: 1.1178 - val_loss: 3.7040

Epoch 00012: val_loss did not improve from 3.68946

Epoch 00012: ReduceLROnPlateau reducing learning rate to 0.00031640623637940735.
Epoch 13/15
628/628 [==============================] - 270s 430ms/step - loss: 1.1116 - val_loss: 3.6713

Epoch 00013: val_loss improved from 3.68946 to 3.67126, saving model to models/nmt4.hdf5
Epoch 14/15
628/628 [==============================] - 269s 429ms/step - loss: 1.1063 - val_loss: 3.6810

Epoch 00014: val_loss did not improve from 3.67126
Epoch 15/15
529/628 [========================>.....] - ETA: 42s - loss: 1.0752
```

Figure 4.5: Training

2. **ModelCheckpoint**:-Generally, we will want to use the model that has the lowest loss on the validation set. This callback will save the model each time the validation loss improves.

3. **EarlyStopping**:-We will want to stop training when the validation loss stops improving. Otherwise, we risk over fitting. This monitor will detect when the validation loss stops improving, and will stop the training process when that occurs.

The function define_model takes the pre established source vocabulary, target vocabulary, English and Hindi max lengths of the sentences and the no of units(Neurons) in the model as the input. Input() layer acts as a staging layer for its next layer that is the Embedding() layer which converts the src_vocab into dense vectors of fixed size. The output of these embedded fixed size vectors are input for the LSTM() layer which returns the last output in the output sequences. In order to prevent the model fit to get overfitted, we apply a Dropout() [8] to the output of the previous LSTM(Long-Short Term Memory) layer. A dual encoder-decoder model is developed by staging and mapping the

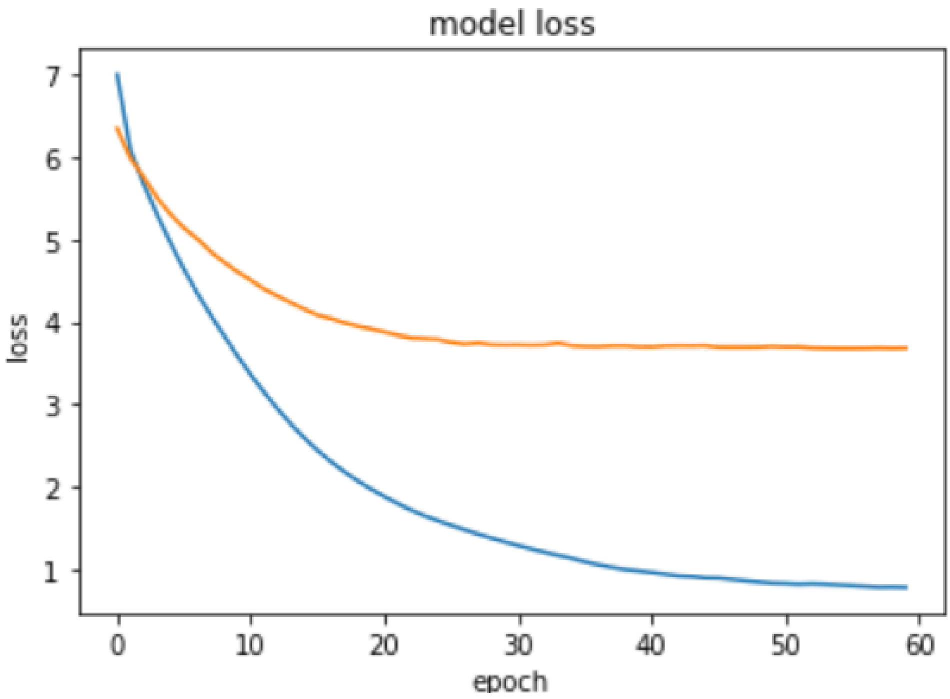


Figure 4.6: Plotting Loss for training

outputs from the LSTM. This is then followed by a Dense() layer of the regular deep connected neural networks is achieved with an activation function of ‘softmax’. This dense layer is followed by a TimeDistributed() wrapper layer used to convert the dimensions of the fixed output vectors.

Finally, the training phase begins when the fit_generator() is called and the model gets trained batch-by-batch by the data batches created previously to avoid RAM crashing. Number of Epochs to be ran, callbacks and the validation data are the major parameters to be passed for the training.

4.4 Coding Standard

We follow the Python PEP8 style guidelines. which can be summarized as below.

4.4.1 Indentation, line-length code wrapping

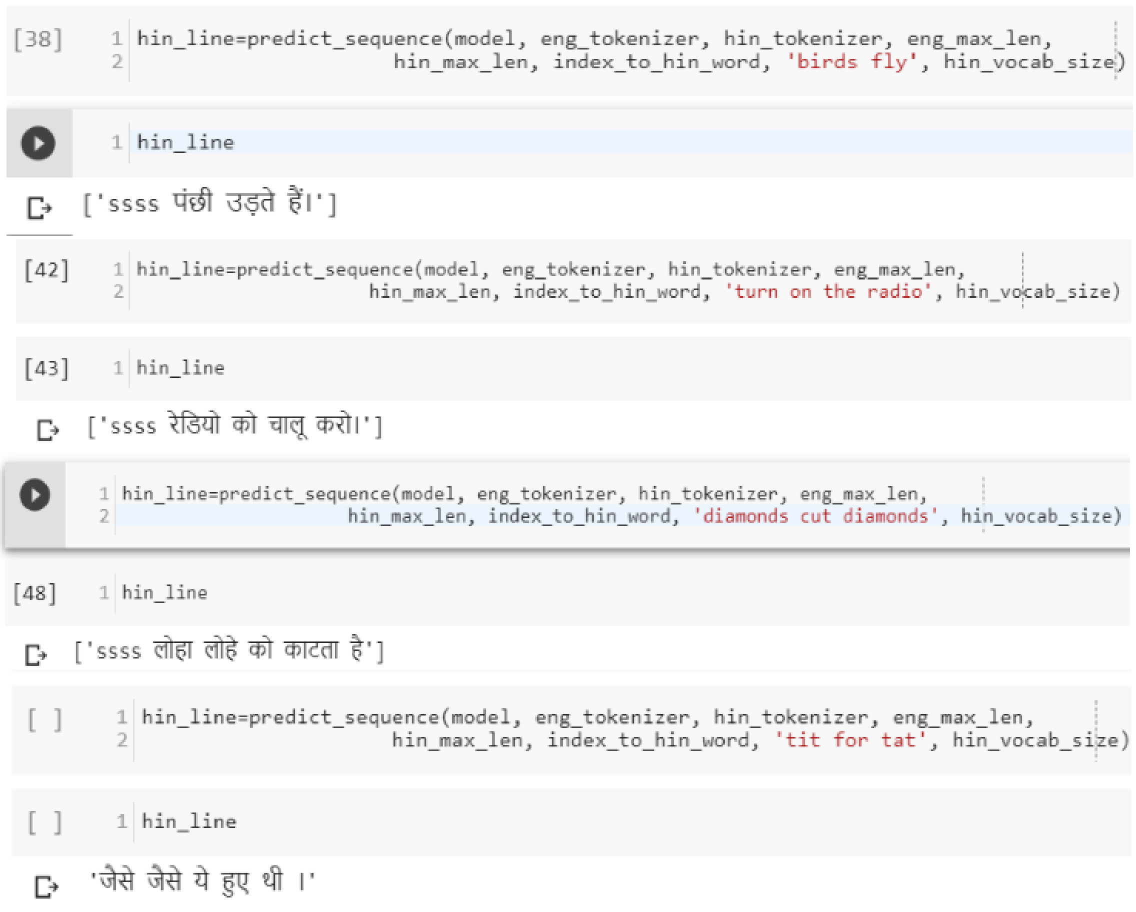
1. Always use 4 spaces for indentation
2. Write in ASCII in Python 2 and UTF-8 in Python 3
3. Always indent wrapped code for readability

4.4.2 Naming Convention

1. method, function and variables names in lowercase with underscores
2. Private methods and properties start with double underscore
3. “Protected” methods and properties start with single underscore
4. For a reserved word, add a to the end (e.g. class)
5. Always use self for the first argument to instance methods
6. Always use cls for the first argument to class methods

4.5 Testing

We manually tested our model for data from the test set. The results obtained are



```
[38]  hin_line=predict_sequence(model, eng_tokenizer, hin_tokenizer, eng_max_len,
                          hin_max_len, index_to_hin_word, 'birds fly', hin_vocab_size)

      hin_line

['ssss पंछी उड़ते हैं।']

[42]  hin_line=predict_sequence(model, eng_tokenizer, hin_tokenizer, eng_max_len,
                          hin_max_len, index_to_hin_word, 'turn on the radio', hin_vocab_size)

[43]  hin_line

['ssss रेडियो को चालू करो।']

      hin_line=predict_sequence(model, eng_tokenizer, hin_tokenizer, eng_max_len,
                          hin_max_len, index_to_hin_word, 'diamonds cut diamonds', hin_vocab_size)

[48]  hin_line

['ssss लोहा लोहे को काटता है']

[ ]   hin_line=predict_sequence(model, eng_tokenizer, hin_tokenizer, eng_max_len,
                          hin_max_len, index_to_hin_word, 'tit for tat', hin_vocab_size)

[ ]   hin_line

'जैसे जैसे ये हुए थी ।'
```

Figure 4.7: Testing Results

Chapter 5

Results and Discussion

5.1 Discussion

The project aims to build a machine translator which would translate from English to Hindi text. This project will efficiently aid the user by analyzing the input given by him and predicting the corresponding translation. The system is trained on Google Colab having its own GPU which requires no hardware costs to train the model. The user interface is made using Flask which a python micro framework.

Machine translation is a tool that can help businesses and individuals in many ways. While machine translation is unlikely to totally replace human beings in any application where quality is really important, there are a growing number of cases that show how effective and useful machine translation can be post-editing.

Leveraging GPU for performing computation intensive tasks can give results significantly faster than using CPU only. Many high level frameworks make it possible to easily develop and deploy more complex model. Using TensorFlow and Keras we can further work on other modules and develop them.

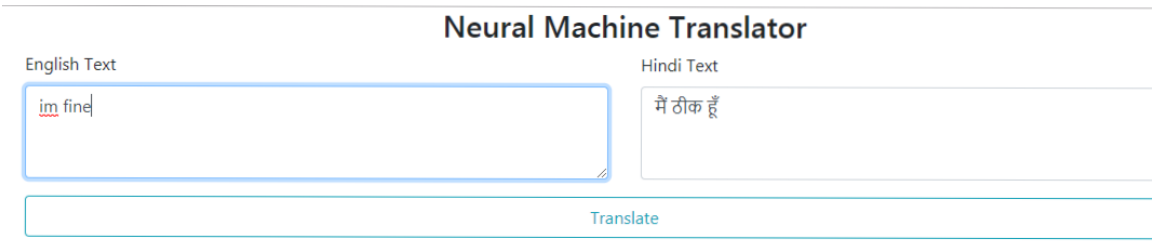
Machine translation is useful in many areas, including:

- Highly repetitive content where productivity gains from machine translation can dramatically exceed what is possible with just using translation memories alone (e.g. automotive manuals)
- Content that is similar to translation memories but not exactly the same (e.g. government policy documents.)
- Content that does not need to be perfect but just approximately understandable (e.g. any website for a quick review.)

- Knowledge content that facilitates and enhances the global spread of critical knowledge (e.g. customer support.)
- Content that would normally be too expensive or too slow to translate with a human only translation approach (e.g. many projects that have insufficient budget for a human only approach.)
- Real-time communications where it would not be practical for a human to translate (e.g. chat and email.)

5.1.1 Results

We have implemented the model that can successfully from English to Hindi. Dividing the data into batch size of 64 and then training helped us to train the model without the use of external hardware. We increased our knowledge of working in python language and also his knowledge of different deep learning architectures. The image shows the interface of our system for the users to translate the text.


Neural Machine Translator

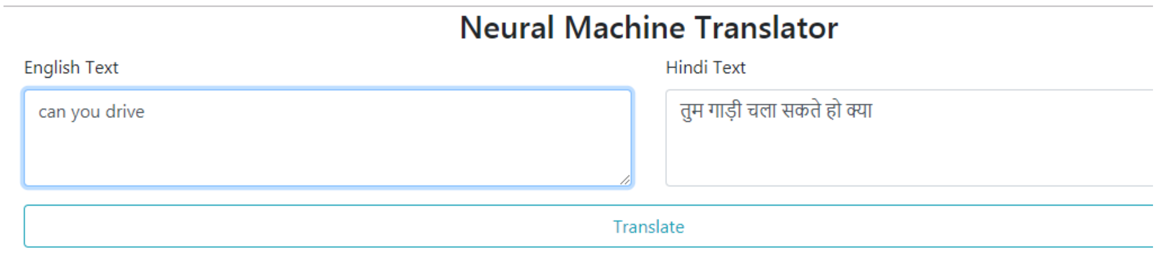
English Text

im fine

Hindi Text

मैं ठीक हूँ

Figure 5.1: Output-1


Neural Machine Translator

English Text

can you drive

Hindi Text

तुम गाड़ी चला सकते हो क्या

Figure 5.2: Output

Chapter 6

Conclusion and Future Scope

6.1 Conclusion

Translation has become essential tool in the modern and ever more globalized world that we live in. Machine translation is a tool that can help businesses and individuals in a variety of ways. Machine translation developed using deep learning based LSTM encoder-decoder techniques is one of the path breaking achievement in the field of translation. The use of LSTMs help in improving the accuracy of the translation. The model is effectively developed using Python language and run on a normal desktop with GPU. This model aims at translating the English text to Hindi Text and vice versa with context using the RNN's by receiving the text input from the user and displaying the output text. We have used more than two layers and it has improved the translation quality. The Bilingual Evaluation Understudy Score (BLEU), is a metric for evaluating a generated sentence to a reference sentence. After analyzing all the above research papers we have learnt that deep learning is an effective method for giving accurate results for translating text. The time to process the data is too high and requires a lot of time to train the data.

6.2 Future Scope

The machine translator can further be used to trained for translation for Hindi to English text and improved using attention mechanism. The drawback of LSTM can be overcome by attention mechanism, attention is simply a vector, normally the outputs of dense layer using softmax function. Before Attention mechanism, translation relies on reading a complete sentence and compress all information into a fixed-length vector, as you can imagine, a sentence with hundreds of words represented by several words will surely lead to information loss or inadequate translation, etc. We can further improve our model by using character level recognition which can help us to achieve open vocabulary along with improving the speed as well as memory utilization.

1. Stronger hardware support to improve network strength
2. Stronger memory support (RAM or GPU) enabling deeper architectures as models would be able to hold more parameters, even for larger datasets
3. Try to procure better quality datasets.

References

[1] Ruchit Agrawal and Dipti Misra Sharma *Experiments On Different Recurrent Neural Networks For English-Hindi Machine Translation* 2017 DOI : 10.5121/csit.2017.71006

[2] Janhavi R. Chaudhary, Ankit C. Patel *Machine Translation Using Deep Learning : A Survey* 2018 International Journal of Scientific Research in Science, Engineering and Technology [IJSRSET]

[3] Francisco Guzman, Shafiq Joty, Lluıs Marquez, Preslav Nakov *Machine translation evaluation with neural networks* 2017 Qatar Computing Research Institute

[4] Minh-Thang Luong and Christopher D. Manning *Achieving Open Vocabulary Neural Machine Translation With Hybrid Word - Character Models* 2016 Computer Science Department, Stanford University, Stanford, CA 94305 Translation

[5] Minh-Thang Luong, Michael Kayser, Christopher D. Manning *Deep Neural Language Models for Machine Translation* 2014 Computer Science Department, Stanford University, Stanford, CA, 94305

[6] Aditi Kalyani and Priti S. Sajja. *A Review of Machine Translation Systems in India and different Translation Evaluation Methodologies* 2015 International Journal of Computer Applications

[7] MT Luong Sutskever O. Vinyals W. Zaremba *Addressing the Rare Word Problem in Neural Machine Translation* 2017 Computer Science Department, Stanford University, Stanford, CA, 94305

[8] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ruslan Salakhutdinov *Dropout: A Simple Way to Prevent Neural Networks from Overfitting* 2014 Journal of Machine Learning Research

[9] Kishore Papineni, Salim Roukos, Todd Ward, Wei-Jing Zhu *BLEU: a Method for Automatic Evaluation of Machine Translation* 2002 Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL), Philadelphia

[10] Ondřej Bojar, Vojtěch Diatka, Pavel Rychlý, Pavel Straňák, Vít Suchomel, Aleš Tamchyna, Daniel Zeman *HindEnCorp – Hindi-English and Hindi-only Corpus for Machine Translation* Charles University in Prague, Faculty of Mathematics and Physics,Institute of Formal and Applied Linguistics

[11] Kingma, D. P., & Ba, J. L. (2015). Adam: a Method for Stochastic Optimization. *International Conference on Learning Representations*, 1–13.

Acknowledgements

We would like to express our special thanks of gratitude to our guide Prof. Uday Nayak, our Head of Department Prof. Prasad Padalkar, as well as our principal Dr. Prasanna Nambiar who gave us the golden opportunity to do this useful project on the topic ”Machine Translator using Deep Learning”, which also helped us in doing a lot of Research and we came to know about so many new things, we are really thankful to them. A lot of effort has been put into the entire project with wholehearted dedication of the project team. Our project guide has helped us immensely in the entire project management aspect, which is a complete new domain to us.

We would also like to thank the staff on information technology department for their guidance and support in every possible way. We are also thankful to our project coordinator Prof Sunantha Krishnan, for her constant support and encouragement.

SHUBHAM AGRAWAL	05	(————————)
ADITYA BHAGWAT	10	(————————)
ROHIT NAIR	45	(————————)
DAELYN OLIVEIRA	47	(————————)

Date:22/04/2019