Grammatical Error Correction for Indian Language

A Project report submitted by

Puja Gupta

in partial fulfillment of the requirements for the award of the degree of

M.Tech

॥ त्वं ज्ञानमयो विज्ञानमयोऽसि ॥

Indian Institute of Technology Jodhpur

Department of Mathematics

November 2022

Declaration

I hereby declare that the work presented in this Project Report titled Grammatical Error Correction for Indian Languages submitted to the Indian Institute of Technology Jodhpur in fulfilment of the requirements for the award of the degree of M.Tech, is a bonafide record of the research work carried out under the supervision of Dr. Gaurav Harit and Dr. Kirankumar Hiremath. The contents of this Project Report in full or in parts, have not been submitted to, and will not be submitted by me to, any other Institute or University in India or abroad for the award of any degree or diploma.

Puja Gupta

Puja Gupta
(M21MA004)

Certificate

This is to certify that the Project Report titled **Grammatical Error Correction for Indian Language**, submitted by ***Puja Gupta (M21MA004)*** to the Indian Institute of Technology Jodhpur for the award of the degree of M.Sc.-M.Tech, is a bonafide record of the research work done by her under my supervision. To the best of my knowledge, the contents of this report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Gaurav Harit

Dr. Gaurav Harit
Department of Computer Science
Indian Institute of Technology, Jodhpur

Dr. Kirankumar R. Hiremath
Department of Mathematics
Indian Institute of Technology, Jodhpur

Abstract

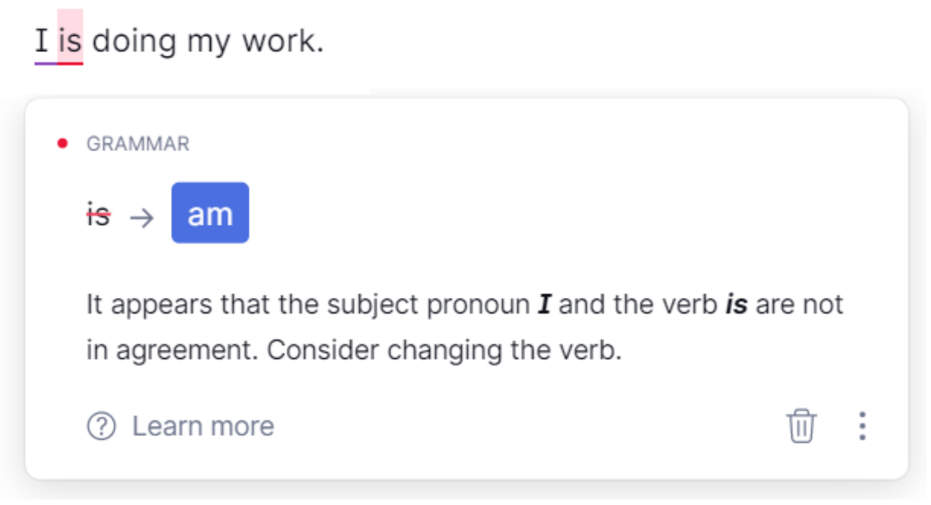
As the name suggests, Grammatical Error Correction (GEC) is the task of error detection and correction in the text. Though the problem seems easy, due to the diverse vocabulary and complex language rules, it is pretty tough to achieve. Writing is one of the most common means of communicating and sharing ideas and information; thus, a model which is capable of grammatically correct sentences in a fast and efficient way is much needed. Many models and techniques are available for grammatical error correction in English, such as Grammarly, Google docs, and numerous other text editors. However, very few works are available in the Indian languages.

In recent years, the use of transformers and other attention-based models for various multilingual tasks has motivated us to develop a state-of-the-art model for performing grammatical error correction tasks for Hindi. After achieving decent results in the GEC tasks for English using the Sequence-to-Sequence model with an attention mechanism, we have proposed the same method for Hindi. The datasets developed by Etoori et al.[1] and Ankur et al.[2] for Hindi, helped us train our model. The different word embeddings used for training the model are Word2Vec, Glove, and FasText. A comparative study and analysis using different architectures have been done regarding model performance, size, and inference time. The model trained using the attention mechanism outperformed the one without attention, as it allows to put focus on the more significant part of the input over others. Also, the best results are obtained from the model trained using the FastText word embedding.

Contents

1 Introduction

As the name suggests, Grammatical Error Correction (GEC) is the task of error detection and correction in the text. As one of the most important research problems in NLP, GEC can be formulated as a sequence-to-sequence task, where a model is trained using grammatically incorrect sentences as input and return a grammatically correct sentence. Though the problem seems easy, but due to the diverse vocabulary and complex language rules, it is quite tough to achieve. Writing is one of the most common means (other than speaking) for sharing ideas and information or communicating with each other; thus, a model capable of grammatically correcting languages is much needed by writers to speed up their work with minimal error. Also, such a model is of great help to individuals who are not fluent in any language. With the advancement in AI and Deep Learning, along with abundant textual data being available, a lot of research work is available for GEC tasks in English. Some commonly used applications implementing GEC for English are Grammarly, Google docs, and numerous other text editors. As it is the basic need to get grammatically correct text in automatic typing, language translation, speech/image-to-text generation, or any other language problem, a similar model (fast and memory efficient) should be available for other Indian languages as well.


I is doing my work.

GRAMMAR

~~is~~ → am

It appears that the subject pronoun ***I*** and the verb ***is*** are not in agreement. Consider changing the verb.

Figure 1.1: Example of Grammatical Error Correction for English

1.1 Motivation

Many models and techniques are available for grammatical error correction in the English language, but not much work has been done for the Indian languages. The few existing models are mostly rule-based or use Statistical Machine Translation Techniques. With the expanding internet traffic with millions of Indian users online, the market for incorporating other Indian languages for content has just exploded. With this, there is a need for some tools for implementing GEC tasks for Indian languages as well which can help the writers.

In recent years, the use of transformers and other attention-based models for performing various multilingual tasks has motivated us to develop a model for performing grammatical error correction tasks for Hindi. Since GEC using neural networks is a very data extensive task, millions of parallel data should be available for training the model to ensure proper learning is done. The work by Etoori et al.[1] and Ankur et al.[2] focused on generating such datasets for Hindi, helped us to carry out our baseline experimentation and analysis in this direction and develop a model for achieving GEC for the Hindi language.

1.2 Objective

We focus on building a state-of-the-art model for performing the Grammatical Error Correction task for Hindi Language and generalizing the process so the same task can be implemented for other low-resource Indian languages using their own language specific dataset. As of now, unlike English, no tool is available for Grammatical Error Correction in Hindi. So building such a model, which is both efficient and fast, will ease human life and can be incorporated into a range of text-generating and editing tools.

The main focus of this part of the project was learning the different architectures available for performing GEC tasks and validating and regenerating the benchmark results obtained by Garg[6] in her work with necessary modifications to improve the model's performance. Since the functional code implemented by Garg[6] was not available, the entire experiments and analysis are done from scratch.

1.3 Thesis Organization

There are total 7 chapters in this thesis. Chapter 1 consists of the introduction, the motivation behind the project and what are the objectives we are trying to achieve. Chapter 2 consists of the literature review in the related field, a brief description about the datasets available and used for both English and Hindi GEC and the different architectures used for this task. Chapter 3 consists of information about different word embeddings available and the ones used. Chapter 4 consists of the proposed method for GEC for Hindi language. Chapter 5 includes the entire analysis and experimentation done, along with result achieved for performing the task of Grammatical Error Correction both for English and Hindi Language. Chapter 6 and 7 includes the conclusion and the list of future work that will be done by me in the final semester.

2
Literature Review

A detailed study on grammatical error correction for English has been done by Garg[6]. Unlike English, very less work has been done for Hindi and other Indian languages. Figure 2.1 summarizes all the related work done for GEC in Hindi.

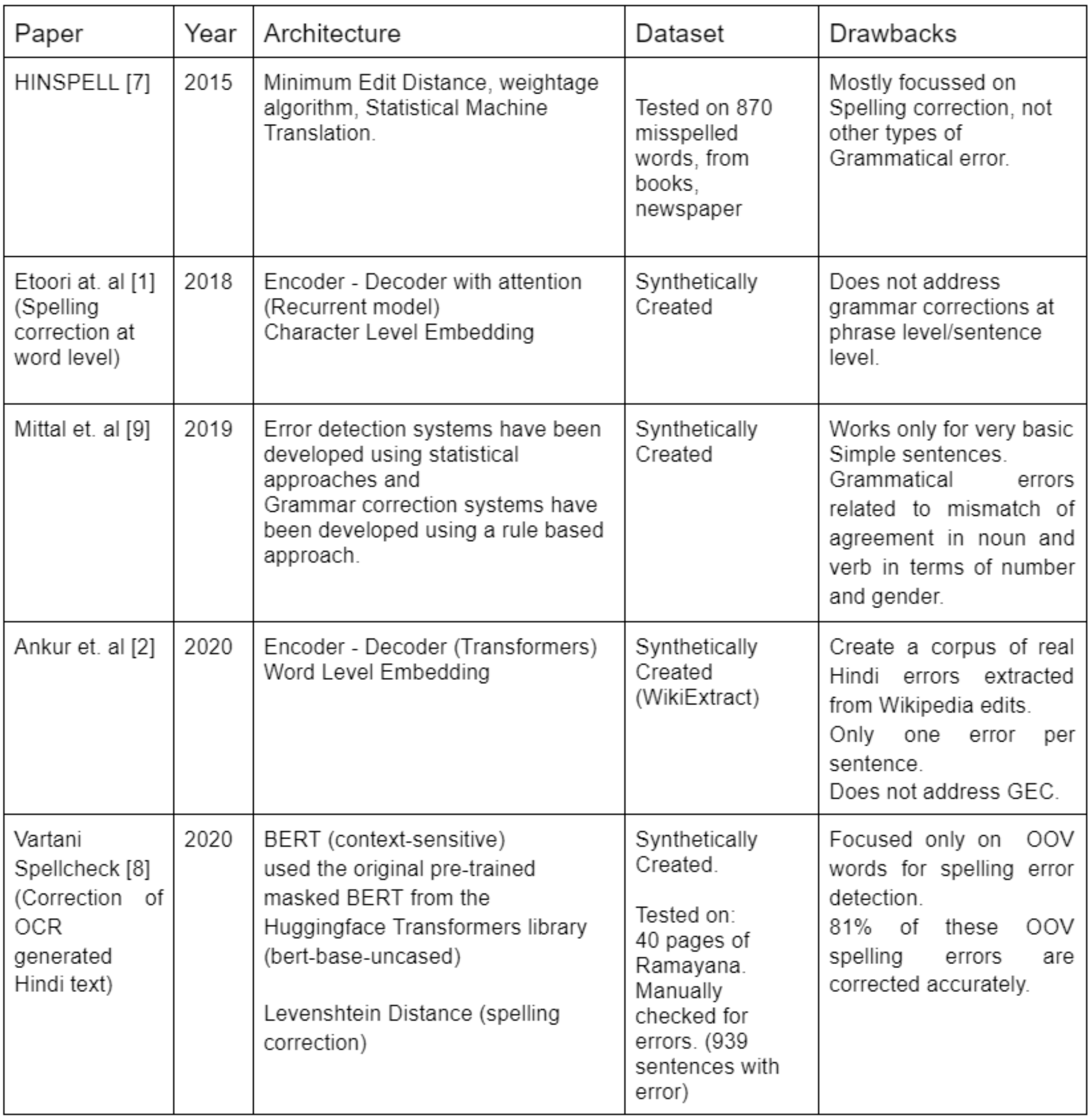
2.1 Dataset Used

Grammatical Error Correction in NLP is a data-extensive task requiring millions of data to train the state-of-the-art models and ensure proper learning. The dataset required for this task consists of parallel supervised data, which are correct and incorrect sentence pairs. Though many benchmark datasets, like LANG8, NUCLE, WI-LOCNESS, CoNLL-2014, etc., are available for English, the nonavailability of good-quality datasets for many Indian languages is a significant problem.

2.1.1 English Datasets

- **LANG8 [3]** dataset is generated using a Japanese website where users post incorrect sentences from the language they are learning, and native speakers of those languages correct them. Those sentences where both incorrect and correct sentences are available are used.
- **NUCLE [3]** corpus consists of 1,400 essays written by the National University of Singapore's students on different topics. These essays are annotated with error tags and corrections by the English instructor. This dataset has also been used in the CoNLL-2014 and BEA-2019 shared tasks.
- **WI-LOCNESS [3]** consists of two datasets: 1. LOCNESS, a corpus of essays written by native English students, which are annotated with errors. 2. Cambridge English Write and Improve (W&I), an online platform where users post incorrect sentences in the form of writings, essays, letters, stories, etc., are manually annotated and corrected.
- **CoNLL-2014 [4]** dataset is created by a system that detects the grammatical errors present in the input texts (short English essays written by non-native English speakers) and returns the corrected essays.
- **Colossal Clean Crawled Corpus (C4) [10]** is one of the largest language datasets available, with more than 156 billion tokens collected from over 365 million domains across the internet. It was used to train the T5 transformer model, which is one of the largest trained transformer models available.

For English, we have taken the LANG8 dataset to train the model.



Paper	Year	Architecture	Dataset	Drawbacks
HINSPELL [7]	2015	Minimum Edit Distance, weightage algorithm, Statistical Machine Translation.	Tested on 870 misspelled words, from books, newspaper	Mostly focussed on Spelling correction, not other types of Grammatical error.
Etoori at. al [1] (Spelling correction at word level)	2018	Encoder - Decoder with attention (Recurrent model) Character Level Embedding	Synthetically Created	Does not address grammar corrections at phrase level/sentence level.
Mittal et. al [9]	2019	Error detection systems have been developed using statistical approaches and Grammar correction systems have been developed using a rule based approach.	Synthetically Created	Works only for very basic Simple sentences. Grammatical errors related to mismatch of agreement in noun and verb in terms of number and gender.
Ankur et. al [2]	2020	Encoder - Decoder (Transformers) Word Level Embedding	Synthetically Created (WikiExtract)	Create a corpus of real Hindi errors extracted from Wikipedia edits. Only one error per sentence. Does not address GEC.
Vartani Spellcheck [8] (Correction of OCR generated Hindi text)	2020	BERT (context-sensitive) used the original pre-trained masked BERT from the Huggingface Transformers library (bert-base-uncased) Levenshtein Distance (spelling correction)	Synthetically Created. Tested on: 40 pages of Ramayana. Manually checked for errors. (939 sentences with error)	Focused only on OOV words for spelling error detection. 81% of these OOV spelling errors are corrected accurately.

Figure 2.1: Related work done for GEC in Hindi

Datasets	Training Sentences	Validation Sentences	Test Sentences
LANG8	1.9M	-	-
NUCLE	57K	-	-
WI-LOCNESS	34.3K	4.4K	4.5K
CoNLL-2014	57K	-	1.3K
Colossal Clean Crawled Corpus (C4)	200M	-	-

Table 2.1: Statistics of datasets available for the English Language

2.1.2 Hindi Datasets

Etoori et al.[1] and Ankur et al.[2] in their work, have focused on generating parallel supervised (correct and incorrect sentence pairs) datasets for Hindi.

- **Etoori's Dataset [1]** has created and released a synthetically generated parallel dataset containing incorrect and correct sentence pairs.
- **HiWikEd [2]** created a parallel corpus (HiWikEdits) of synthetic errors by inserting errors into grammatically correct sentences using a rule based process (focusing specifically on inflectional errors). The incorrect sentence generated by them has only one error per sentence. Also, a test corpus of real Hindi errors was extracted by them from Wikipedia edits for evaluating the performance of models on real-time errors.

Datasets	Training Sentences	Validation Sentences	Test Sentences
Etoori's Dataset	140K	30K	30K
HiWikEd	2.6M	-	13K

Table 2.2: Statistics of datasets available for Hindi

The dataset generated by Etoori et al.[1] has been used in our project to carry out the baseline experimentation and train the model.

2.2 Architectures

Initially, for the machine translation task, the work of text correction started and was implemented using basic Recurrent Neural Network (RNN), Gated recurrent units (GRU), and Long short-term memory (LSTM) models. With the advancement of deep learning, researchers started shifting from using a base model to more advanced sequence-to-sequence and transformer models, which showed more promising results. The models are trained using both word and character-level embeddings, which have their own advantages as per the application.

2.2.1 Sequence2Sequence Encoder Decoder Model

Sutskever et al.[11] proposed a sequence-to-sequence architecture using multilayered LSTM (Long Short-Term Memory) units to map the input sequence of words fed to the Encoder in the form of a context embedding vector. This context embedding vector obtained from the final layer of the Encoder is then fed to another Decoder (LSTM-based) unit to decode the target sequence. The input and output sequence lengths can or cannot be the same. This architecture was first used to perform the English-to-French machine translation task. The sequence-to-sequence model showed better performance than the previously used Statistical Machine Translators. The LSTM units used in the Encoder and Decoder also helped in the sensible phrase and sentence representation. Though in the work proposed by Sutskever et al.[11], LSTM was used, but any other architecture such as RNN, GRU, and bi-direction LSTM can also be used in the Encoder and Decoder. The same architecture can also be used for the Grammatical Error Correction task. Figure 2.2. shows a Seq2Seq encoder decoder model without attention.

2.2.2 Sequence2Sequence Encoder Decoder Model with Attention

Sequence-to-Sequence model proposed by Sutskever et al.[11] has the disadvantage of not performing better when we have long sentences. The Seq2Seq model compresses the information

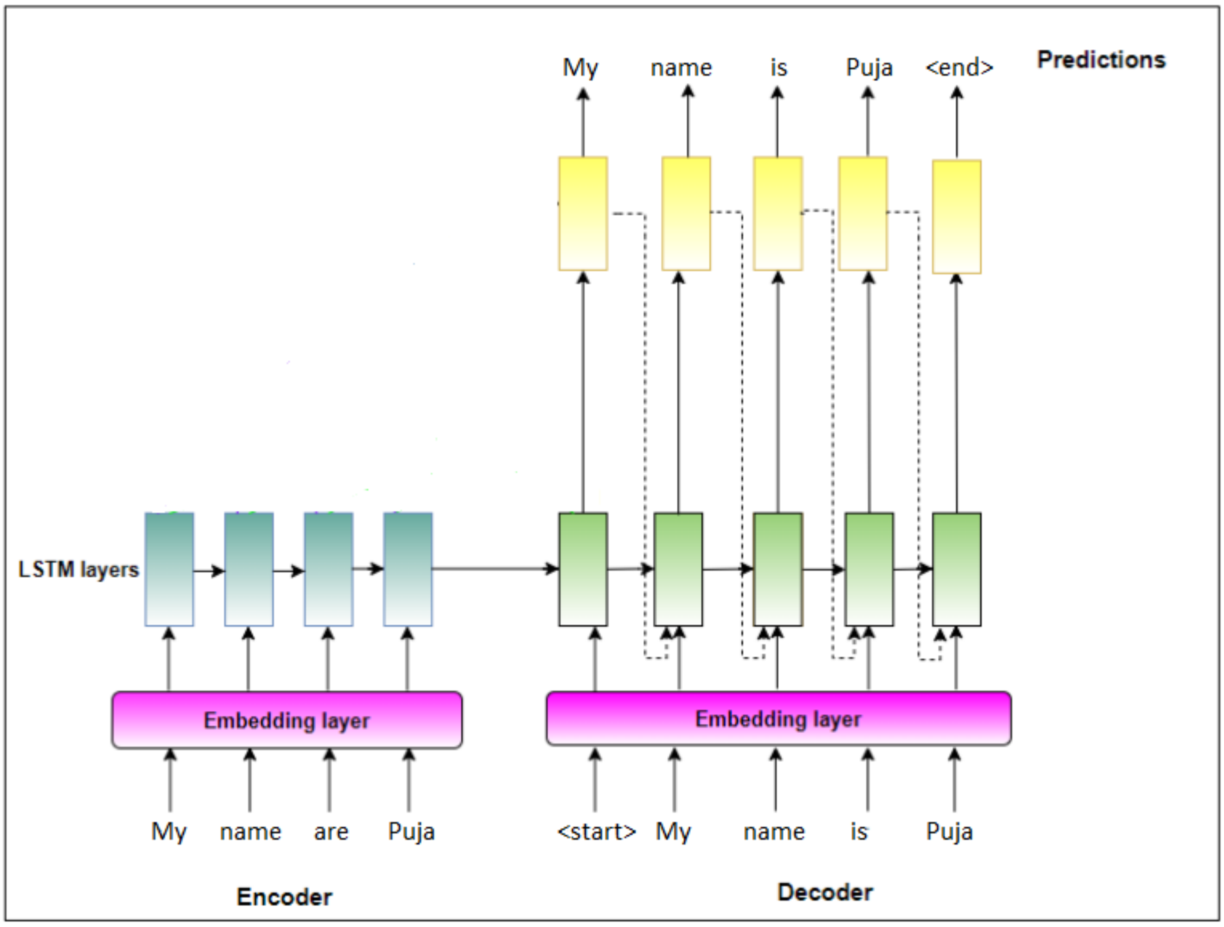


Figure 2.2: [6] Seq2Seq Encoder Decoder model without Attention

about the input sentence into a fixed-length context vector embedding. However, if the input sentence is longer, there will be a greater information loss leading to an information bottleneck. It has been shown that the accuracy decreases as the sentence length increases with respect to the encoder and decoder. Vaswani et al.[12] proposed the concept of an attention mechanism to overcome this disadvantage.

The use of the attention mechanism lets the decoder look back to the source (gives access to the encoder's hidden states) and learn how much attention/focus needs to be given to each encoder's hidden state for this decoder timestamp. Attention weights are calculated for each decoder's timestamp, concatenated with the context vector and input embedding, and fed to the decoder for decoding the output sentence. It was seen that incorporating the attention mechanism significantly improved the performance of the encoder-decoder model. Tao et al. [13] and Schmaltz et al.[14] have used the attention-based sequence-to-sequence architecture for implementing grammatical error correction for the Chinese language. Figure 2.3. shows a Seq2Seq encoder-decoder model with attention.

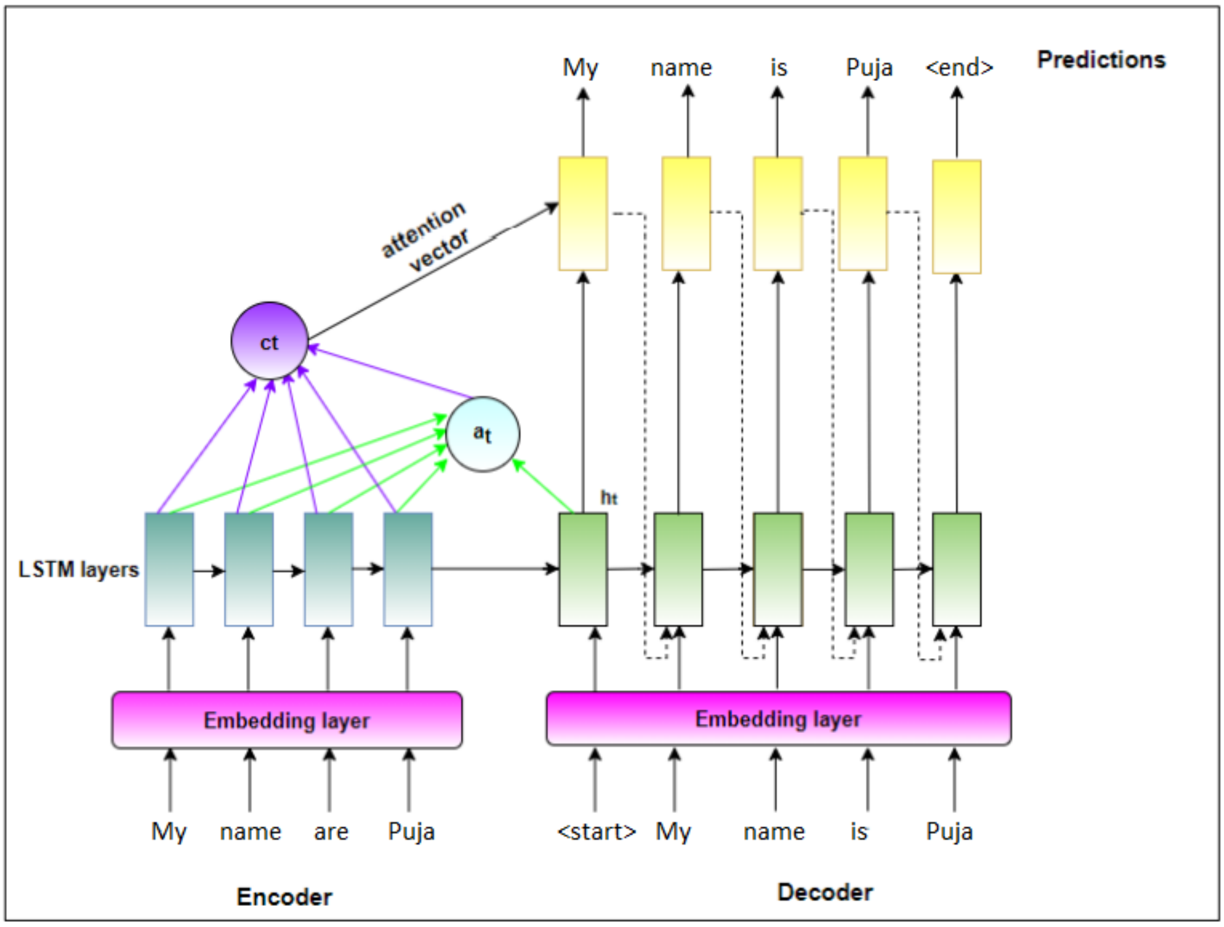


Figure 2.3: [6] Seq2Seq Encoder Decoder model with Attention mechanism for GEC in English

3
Word Embeddings

In NLP, the input we have (dataset) is in the form of textual data, which cannot be directly fed to the model, as the machine does not understand the textual data. So we convert the textual data into some word vector first (numerical form) and feed that vector representation of the word to the model for processing. Word embeddings are used to convert words into vector representation to make them understandable by the machines by capturing the different morphological, contextual, semantic, or syntactic relationships between the words. Models learn these word vectors through different unsupervised learning techniques, which enables them to store contextual information of words in a low-dimensional vector. We have various pre-trained state-of-the-art word embedding models available for English and Hindi. These can convert any textual sentence into the corresponding vector representation, which can be fed to the model to learn. Different word embeddings used in this project have been described below.

3.1 Word2Vec

Word2Vec provides the N-dimensional vector representation of the document vocabulary. If we plot the word embeddings in a 2D space, similar words are plotted together, and words unrelated to each other are plotted at a further distance. The spatial position of two words is calculated based on the Cosine Similarity between them.

The two different methods to obtain Word2Vec word embeddings (both processes involve training Neural Networks with very large datasets such as big long sequence of texts from Wikipedia) are Skip Gram and Common Bag of Words.

3.1.1 Continuous Bag-of-words

In Continuous Bag-of-words (CBOW) we take the context and predicts the central word based on the highest probability. Figure 3.1. shows a block diagram of CBOW. For simplicity, let's consider a language that hasonly five words (My, Final, M.Tech, Project, On, GEC). We initially encode each word as one-hot vector (one bit as one and the rest all 0). To generate word vectors from this vocabulary, we iterate through a corpus. For simplicity, let's say our corpus has just one sentence: ”My final M.Tech project on GEC.” Now we need to select a window size for iterating through this sentence (m=3). Since the window size is 3, we need to predict the center word using the two surrounding words. We need a simple neural network for this. In this case, the neural network will have the context word ”My” and ”Final” as input, fed to the neural network to predict ”Final.” The size of each vector also needs to be chosen (say 3), then the size of the hidden layer for our neural network will also be 3. Figure 3.2. shows the basic architecture and working of this neural network. The neural network tries to predict the central word given the context words. We pass it to the softmax function to get the probability and compare it with the actual word. The error is used to update the weights and then we slide the window to get all the weight matrix, which helps us to get is the set of vectors.

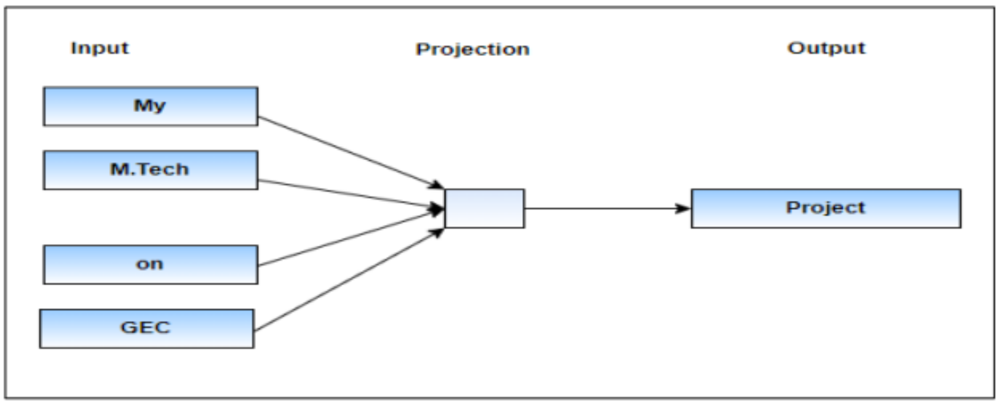


Figure 3.1: Common Bag Of Words

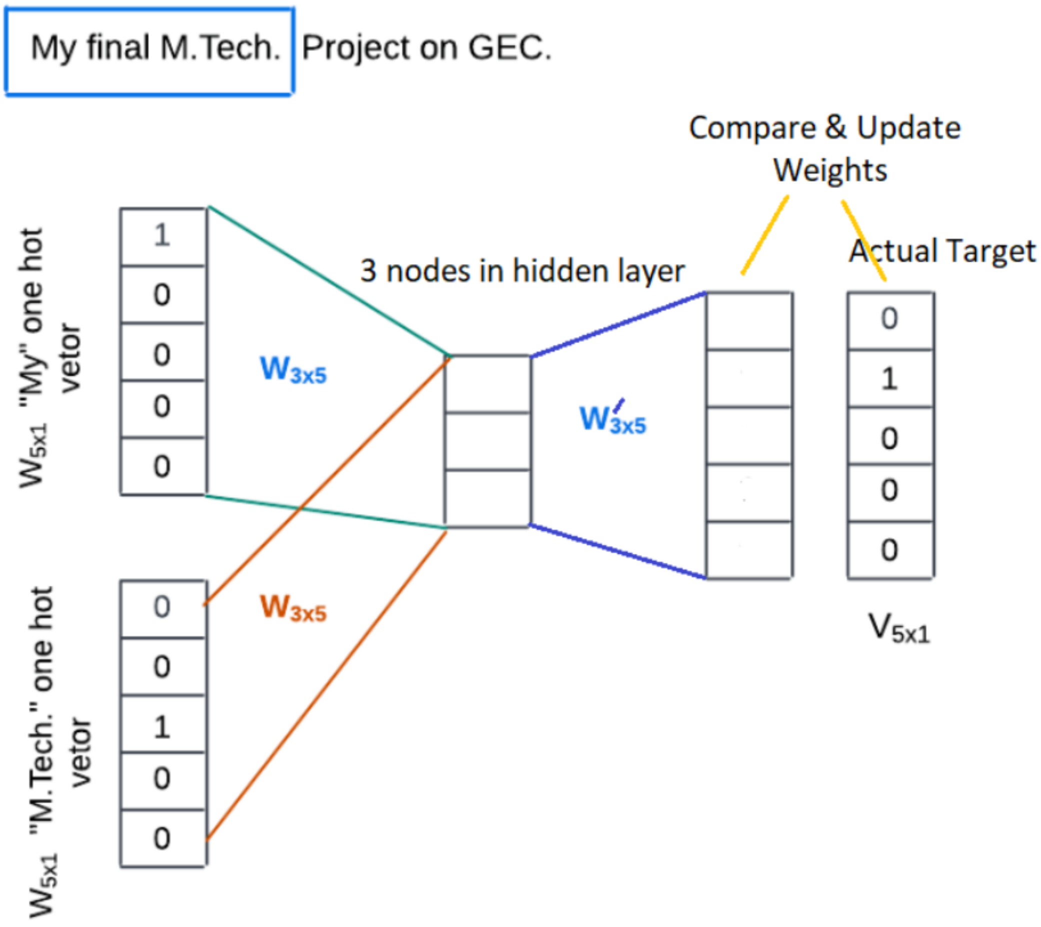


Figure 3.2: Working of CBOW Algorithm

3.1.2 Skip Gram

In contrast to CBOW, in SkipGram (SG) we predict the the context (single or combination of words) based on the central word. So basically, for each central word, t = 1,...m, we predict the surrounding words in the ”m” window size by maximizing the objective/loss function, which is defined as the probability of any context word given the surrounding word. Figure 3.3. shows a block diagram of Skip Gram and Figure 3.4. shows the working of the SG architecture for the example discussed in section 3.1.1.

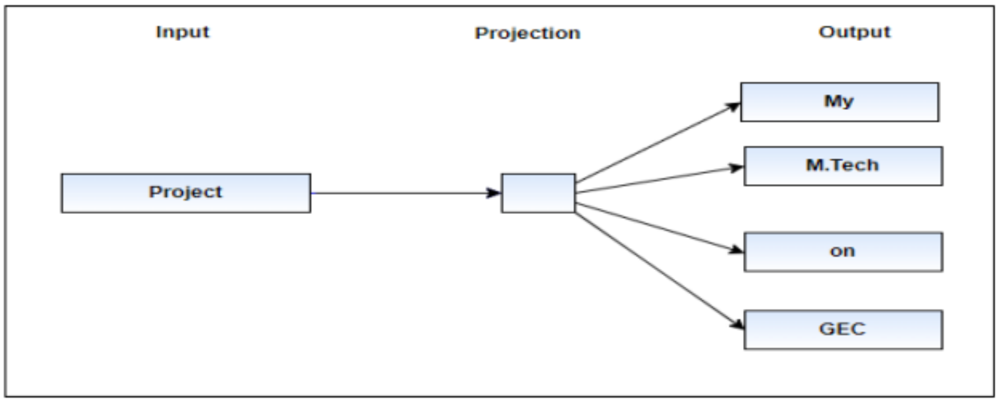
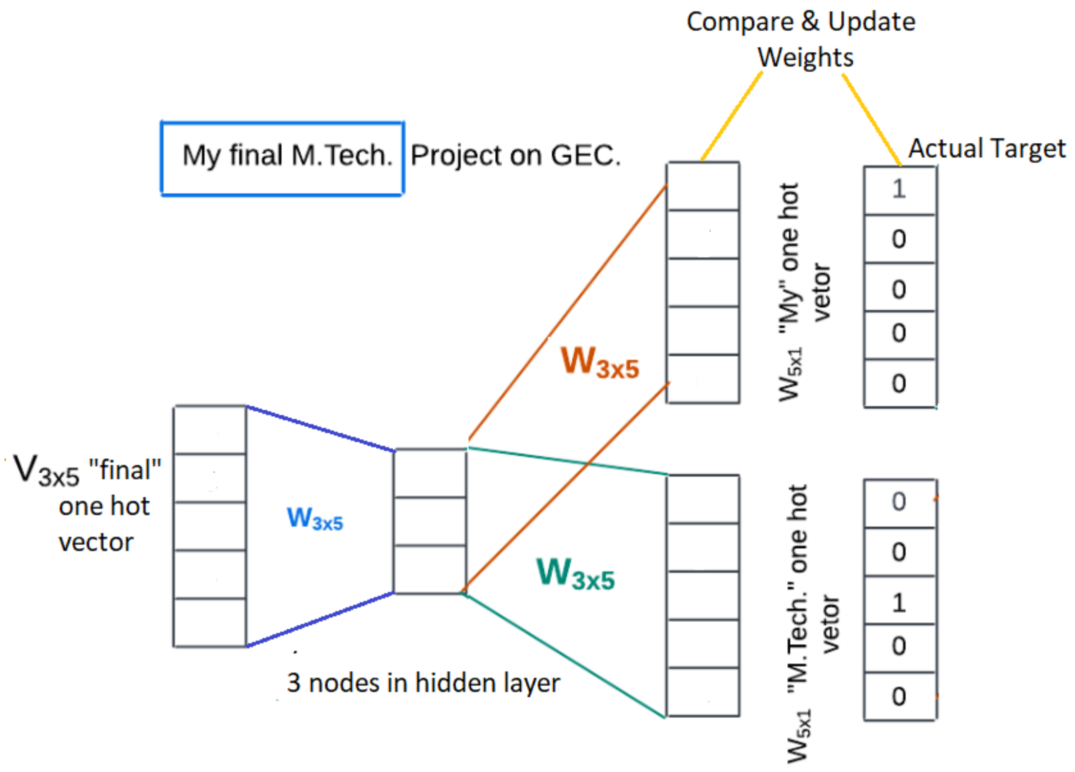


Figure 3.3: Skip Gram



Compare & Update Weights
Actual Target
My final M.Tech. Project on GEC.
V_{3x5} "final" one hot vector
W_{3x5}
W_{3x5}
W_{3x5}
3 nodes in hidden layer
W_{5x1} "My" one hot vetor
W_{5x1} "M.Tech." one hot vetor
1
0
0
0
0
0
0
1
0
0

Figure 3.4: Working of Skip Gram

Few drawbacks of Word2Vec are:

- It does not handle disambiguation i.e. it does not capture the context (sense) in which a word is being used.
- It cannot handle OOV (Out of Vocabulary) words i.e. it won't be able to generate a vector representation of the words which are not seen during the training.
- It does not capture sub-word information of a word.

3.2 Glove

Glove, a refinement technique of Word2Vec, is another method of finding word embeddings. Unlike Word2Vec, global statistics are also considered to obtain the vector representation of words instead of just relying on the local context information of the words. Glove provides better performance than Word2Vec as it also captures the sub linear relationships of the word vectors. In Glove, relationships between word pairs are also taken into account rather than capturing just word and word relationships. Also, less weight is given to the most frequent word pairs to prevent meaningless stop words like a, an, the, etc., from dominating the training progress. One drawback of Glove is that it takes a lot of memory space as the model is being trained (using unsupervised learning) on the co-occurrence word matrix.

3.3 Fasttext

Fasttext is another technique for generating word embeddings and was created by the Facebook Research Team. Unlike Word2Vec, the n-gram technique is used to generate the word representation in Fasttext. This n can vary from 1 to length on word, and these n-gram words are used along with the original word vocabulary for training the model. We basically break the word into multiple chunks and add two tokens at the start and end of the word chunks. The new chunks formed and the whole word are combined and fed to the neural network to predict the next word. For simplicity, let's say we have one and given one word fed to the neural network want to predict the next word. sentence: ”My final project on GEC.” So if n=3, the word ”final” will be broken down into [fin,ina,nal] and along with complete f”final” will be fed to the neural network to predict the next word ”project”. Figure 3.5 shows the working of above example using Fasttext

Advantage of using Fasttext over Word2Vec and Glove:

- Unlike Word2Vec and Glove, it can give word vectors for OOV (Out of Vocabulary) and rare words because a single word is further broken into character n-grams.
- While learning word vector representation, it also considers the internal structure of a word and can be used with smaller datasets.

3.4 Contextual Word Embeddings

Word Embeddings such as Word2Vec, Glove, and Fasttext are also called Non-Contextual Word Embeddings. They usually suffer the drawback of not capturing the sense or context of a

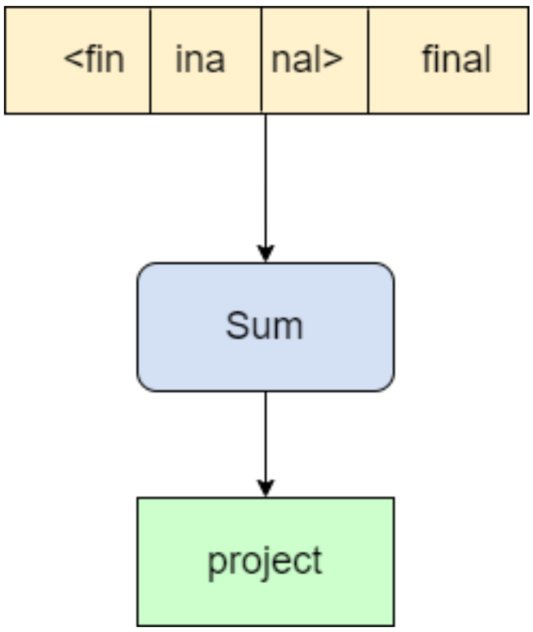


Figure 3.5: Working of FastText

word used in a sentence, as the same word can have different meanings in different sentences. However, each word will have a single representation in these word embeddings. Contextual Word Embeddings such as ELMo, BERT, MuRIL, etc., are dynamic embeddings and can be used to overcome these drawbacks. These models consider the context in which a word has been used so that the same word can have different vector representations in different sentences.

4
Proposed method for GEC

4.1 Sequence2Sequence Model with Attention

Many models and techniques are available for grammatical error correction in the English language, but not much work has been done for the Indian languages. The few existing models are mostly rule-based or use Statistical Machine Translation Techniques which is the motivation behind this project to propose state-of-the-art grammatical error correction tool for Hindi. A sequence-to-sequence word level model, as discussed in section 2.2.2, shown in figure 4.1 has been implemented with 1 Long short-term memory (LSTM) layer each in the Encoder and Decoder with an attention mechanism. Word-level input using different word embeddings, as discussed in the below subsection, has been used. The dataset proposed by Etoori. et al.[1] is used to train the above model, containing 140K parallel sentences and 10K test sentences. Exploratory Data Analysis was performed to get a sense of the dataset. Duplicate data entries and unnecessary spaces were removed from the dataset, leaving 1,39,798 training sentences and 10K test sentences. Indic NLP library for Tokenisation was used to create the word vocabulary for the encoder and decoder. The encoder word vocabulary consists of 73829 words, and the decoder word vocabulary consists of 72971 words. The word embedding of incorrect sentences in the dataset is given to the encoder, and the word embedding of correct sentences along with the context word embedding generated from the final Encoder layer and learnt attention weights are given to the decoder to train the model. All models are trained for 50 epochs with Early stopping criteria on validation loss with patience 5. All the codes are written in TensorFlow using Keras libraries. Cuda version 11.1 and cuDNN version 8.1.1. are used. The experiment and analysis code repository is available on the 172.25.0.209 server under the user id gupta92.

4.1.1 Word2Vec

A pre-trained Word2Vec CBOW (Common Bag of Words) model for the Hindi language of size 4 GB by Kumar et al. [5] was downloaded and used in the project for generating the word embeddings. The dimension of embeddings used is 300. The word embeddings generated using this model are fed to both the encoder and decoder of the Seq2Seq model for training. The model is being trained for 50 epochs with an early stopping criterion monitoring the validation loss with patience five, i.e., the number of epochs monitored with no improvement, after which the model training is stopped. A batch size of 64 and 32 attention units is used to train the model.

4.1.2 Glove

A pre-trained Glove model on the Hindi language by Kumar et al. [5] of size 4.6 GB is used for generating the word embeddings. The word embedding generated for each word is of dimension

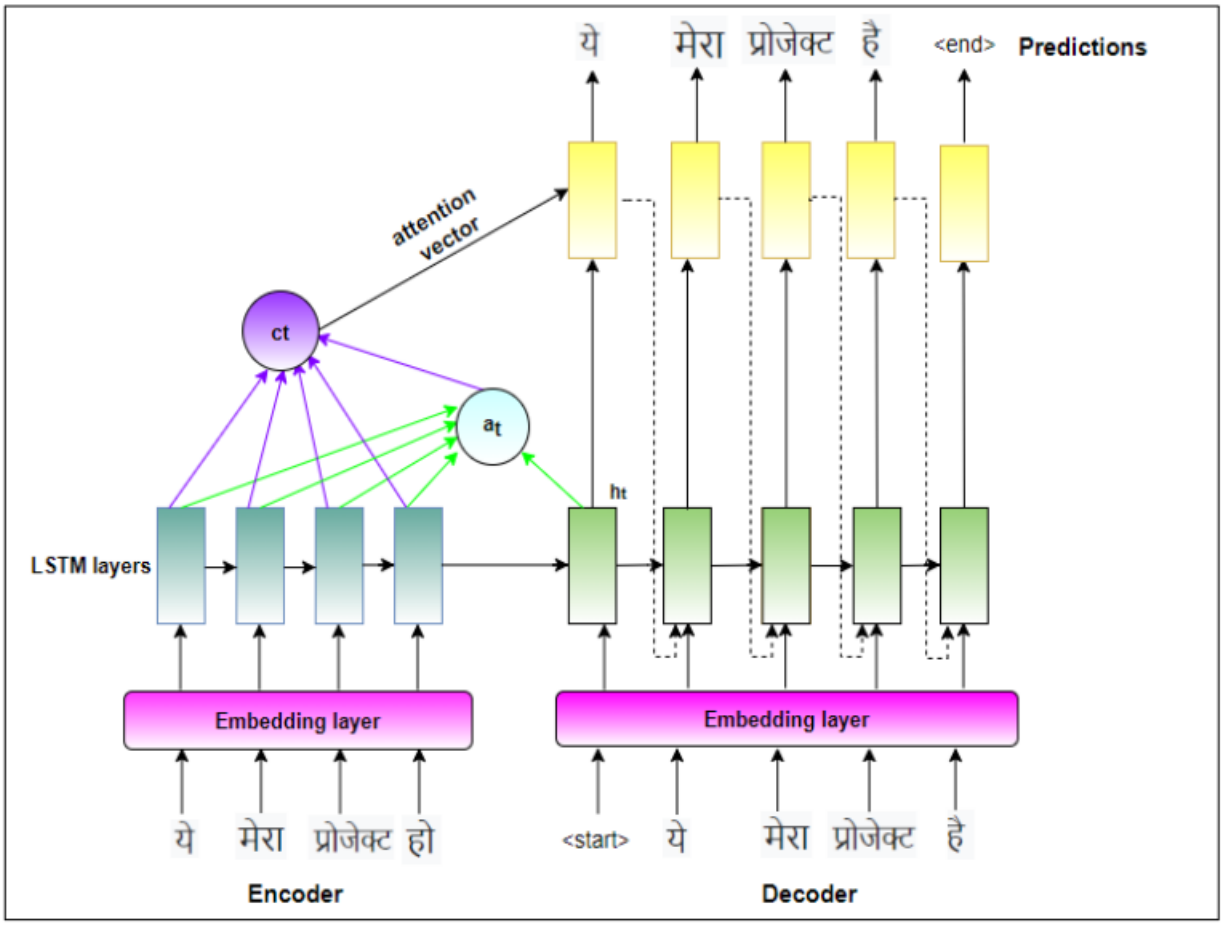


Figure 4.1: [6] Seq2Seq Encoder Decoder model with Attention for GEC in Hindi

300. The model is trained using these embeddings for 50 epochs with an early stopping criterion. A batch size of 32 and 32 attention units is used to train the model.

4.1.3 Fasttext

A pre-trained Fasttext model on the Hindi language of size 2.6 GB is downloaded using fastText library and used in our project for generating the word embeddings. The word embedding with dimension 300 is generated for the words of encoder-decoder vocabulary and is used to train the model for 50 epochs with an early stopping criterion (monitoring the validation loss) to avoid overfitting the model. A batch size of 32 and 32 attention units is used to train the model.

4.2 Loss Function

Sparse Categorical Cross Entropy Loss is used for calculating the loss while training the model. In the below equation y_i refers to true label and y_i' refers to predicted label. Unlike Categorical cross-entropy, where the true labels are one-hot encoded, in Sparse Categorical cross-entropy, the true labels are integer encoded.

$$J(w) = -\frac{1}{N}\sum_{i=1}^{N}[y_i log(y_i^{'}) + (1-y_i)log(1-y_i^{'})]$$

4.3 Evaluation

The model's performance is evaluated in terms of accuracy metrics, i.e., training accuracy, validation accuracy, training loss, and validation loss. The trained Seq2Seq model (using different word embeddings) is then used to calculate the BLEU score on the 10k test data.

4.3.1 Sentence Prediction

Greedy decoding is used for correct sentence prediction in the project. Greedy decoding goes with whichever word has the highest probability at hand (does not consider what comes before that word) and might end with a sentence with a lower probability as a whole. BEAM search can be used to overcome this issue. In contrast to Greedy Search, using Beam search, the best N sequences so far are considered, and the probabilities of all the combinations of the preceding words along with the current word are used to predict the sentence.

4.3.2 Accuracy

It tells how well the model is performing in terms of correctly predicting the sentence. The accuracy can be calculated using the formulae:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

where TP = True Positive, TN = True Negative, FP = False Positive and FN = False Negative. Though accuracy might not always give the correct picture of how well the model is performing for NLP tasks. Other Evaluation techniques like BLEU score and GLEU Score gives a better understanding of the model's performance.

4.3.3 BLEU Score

BLEU (Bilingual Evaluation Understudy Score) Score is used to evaluate machine-translated text and ranges between the value 0 to 1. It uses the similarity between the translated and original sentences to calculate the score. It is a precision focused metric that evaluates the n-gram overlap of reference and predicted sentences. The matching of n-grams is position-independent.

5
Experiments and Analysis

The experiment and analysis code repository is available on the 172.25.0.209 server under the user id gupta92.

5.1 Seq2Seq Word level Models for English

In the project's initial phase, experiments were carried out using Seq2Seq Encoder-Decoder model as discussed in section 2.2.1 and 2.2.2, for English to study the model's performance, as this technique is already available. The dataset used is LANG8 with 80624 training sentences, 20156 validation sentences, and the model's performance was tested on 2000 test sentences.

Model	Embedding	Training Loss	Validation Loss	BLEU Score
Encoder Decoder	Trainable	0.5163	1.1652	0.1294
Encoder Decoder (Attention)	Trainable	0.2610	0.6519	0.4426

Table 5.1: Test results obtained for different models for English

These results motivated us to implement the same method for GEC in Hindi as well.

5.2 Seq2Seq Word level Models for Hindi

We have trained the Seq2Seq Encoder Decoder model, as discussed in chapter 4, both with and without attention mechanism. The dataset generated by Etoori et al.[1], as discussed in section 2.1.2, has been used in the project to carry out the baseline experimentation and train the model. Also the different word embeddings, as discussed in the section 4.1.1, 4.1.2 and 4.1.3 are used while training the model such as Word2Vec, Glove, and Fasttext. It has been observed that the model trained using attention mechanism outperformed the model trained without attention (Results shown in Table 5.3). Also, the best performance was achieved using Fasttext word embedding over others (Results shown in Table 5.4).

5.2.1 Exploratory Data Analysis

Exploratory Data Analysis was performed on Etoori's dataset to get a sense of the data. Duplicate data entries and unnecessary spaces are removed from the dataset. Analysis of word distribution of incorrect and correct sentences indicates that the maximum number of characters present in the input is 1823 with 410 words. Also, more than 99 percentile of the inputs in the dataset has 51 words with 248 characters. The same kind of distribution has been observed for both incorrect and correct sentences. Also, the visualization of most occurring words in the incorrect and correct sentences has been observed using the word cloud.

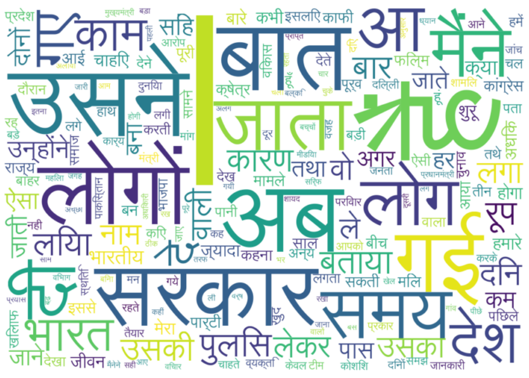
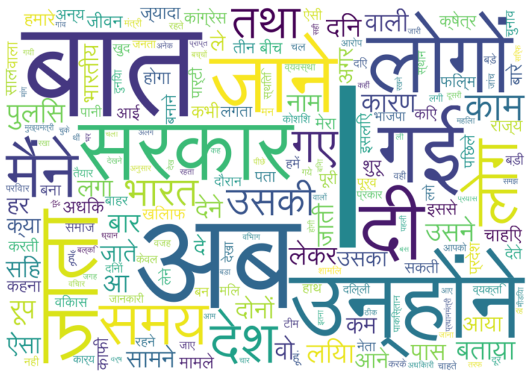
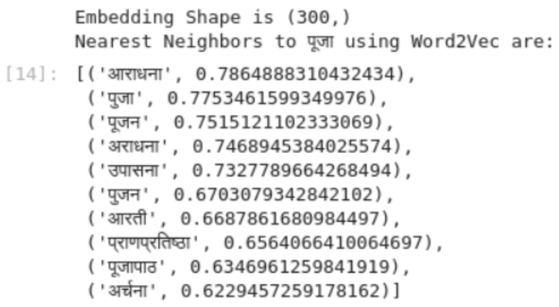
[a]

[b]


Figure 5.1: [a] Word cloud for incorrect sentences and [b] Word cloud for corresponding correct sentences in the Etoori's dataset.

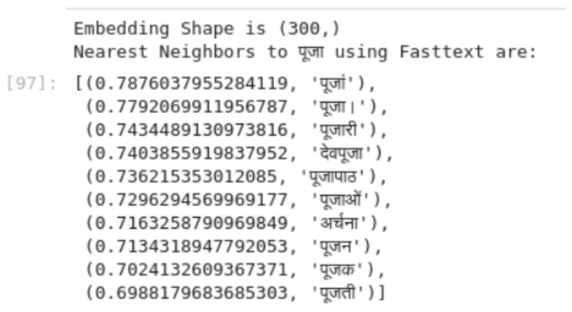
5.2.2 Different Word Embedding Analysis

1. Common Word Predictions

The nearest common word prediction using Word2Vec and Fasttext has been shown in the Figure 5.2. As we can see, the nearest words predicted using Word2Vec are almost synonyms of each other, whereas the nearest words predicted using Fasttext are more diverse in terms of the words which can be related to the given word.

[a]


```
         Embedding Shape is (300,)
         Nearest Neighbors to पूजा using Word2Vec are:
[14]:    [('आराधना', 0.7864888310432434),
          ('पुजा', 0.7753461599349976),
          ('पूजन', 0.7515121102333069),
          ('अराधना', 0.7468945384025574),
          ('उपासना', 0.7327789664268494),
          ('पुजन', 0.6703079342842102),
          ('आरती', 0.6687861680984497),
          ('प्राणप्रतिष्ठा', 0.6564066410064697),
          ('पूजापाठ', 0.6346961259841919),
          ('अर्चना', 0.6229457259178162)]
```

[b]


```
         Embedding Shape is (300,)
         Nearest Neighbors to पूजा using Fasttext are:
[97]:    [(0.7876037955284119, 'पूजां'),
          (0.7792069911956787, 'पूजा।'),
          (0.7434489130973816, 'पूजारी'),
          (0.7403855919837952, 'देवपूजा'),
          (0.736215353012085, 'पूजापाठ'),
          (0.7296294569969177, 'पूजाओं'),
          (0.7163258790969849, 'अर्चना'),
          (0.7134318947792053, 'पूजन'),
          (0.7024132609367371, 'पूजक'),
          (0.6988179683685303, 'पूजती')]
```

Figure 5.2: Nearest neighbor of the same Hindi word using [a] Word2Vec and [b] Fasttext

2. Visualization of Word Embeddings using tSNE

We can plot the word embeddings in a 2D space, which is helpful for checking how words are related to each other. Similar words are plotted together, and words unrelated to each other are plotted at a further distance. TSNE (t-Distributed Stochastic Neighbor Embedding) is the method to reduce the dimension of the word embeddings to 2 or 3, so they can be visualized in a 2D space. tSNE plot of different word embeddings with 50 words from the dataset vocabulary has been shown in the figure 6.3 and 6.4

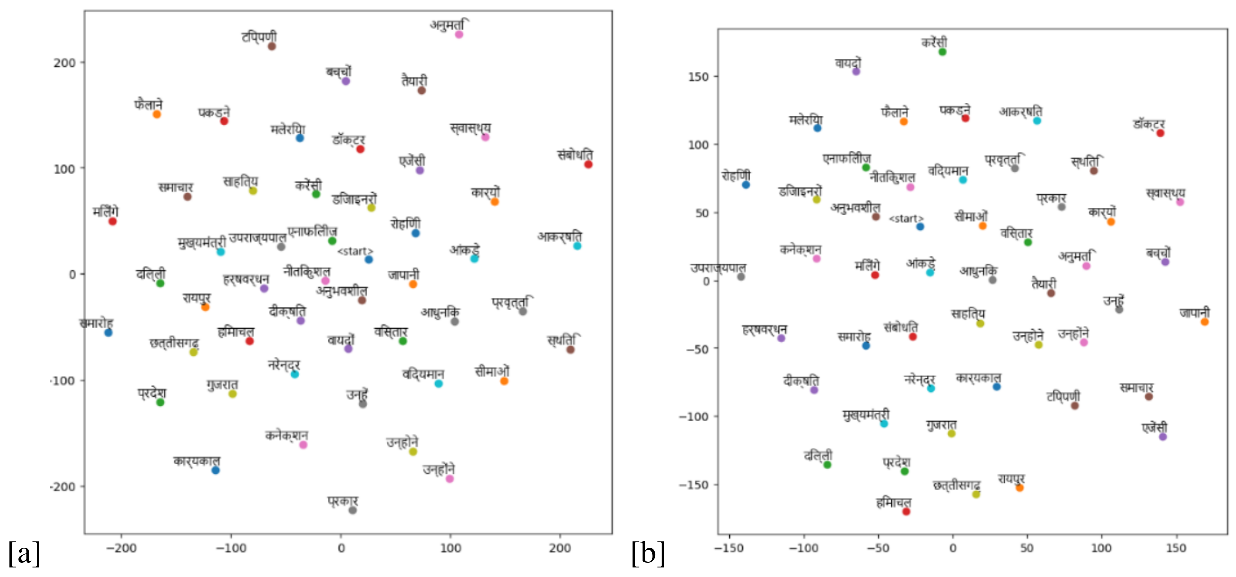


Figure 5.3: shows tSNE plot for [a]Word2Vec [b]Glove

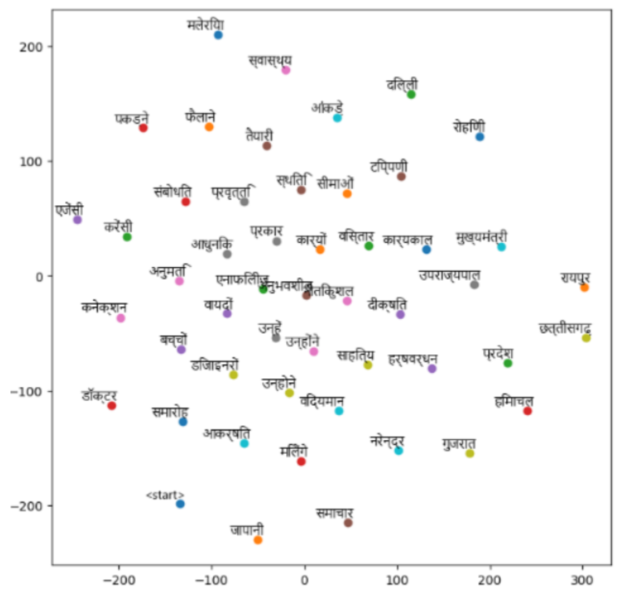


Figure 5.4: tSNE plot for Fasttext

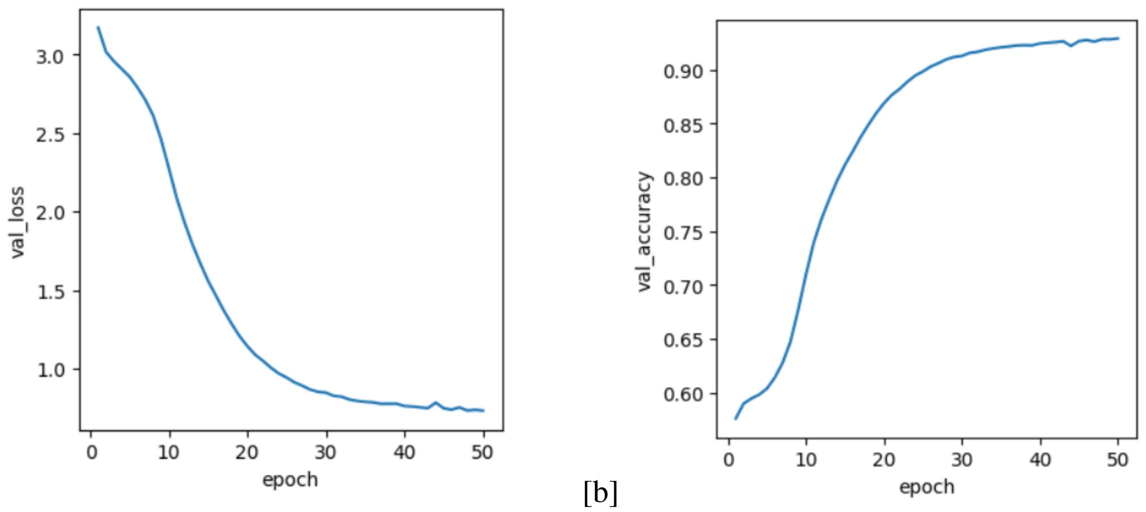
5.2.3 Model Training

The train-valid-test split for each of the models are listed in the Table 5.2.

Figures 5.5, 5.6 and 5.7 shows the plot of validation loss Vs epoch and validation accuracy Vs epoch for the Encoder Decoder architecture with attention for Word2Vec, Glove and Fasttext respectively, showing a decrease in loss and increase in accuracy with epoch ensuring proper training. An early stopping criterion on validation loss has been used to avoid overfitting of the model.

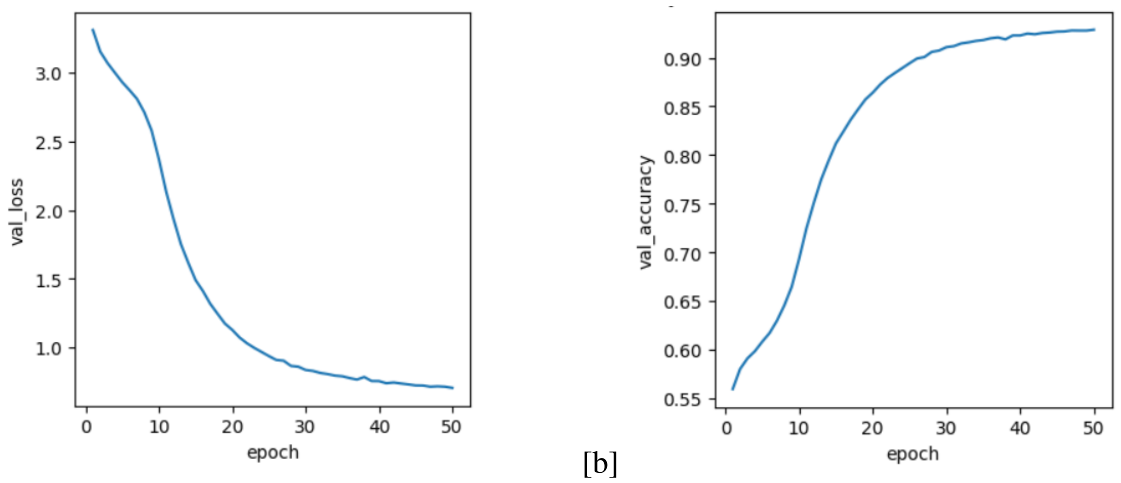
Model	Embedding	Training Sentences	Valid Sentences	Test Sentences
Encoder Decoder	Trainable	112k	28k	10k
Encoder Decoder(Attention)	Trainable	112k	28k	10k
Encoder Decoder(Attention)	Word2Vec	112k	28k	10k
Encoder Decoder(Attention)	Glove	112k	28k	10k
Encoder Decoder(Attention)	Fasttext	112k	28k	10k

Table 5.2: Number of sentences used for training, validating and testing different models for GEC in Hindi



[a] [b]

Figure 5.5: [a] Validation Loss Vs Epoch and [b] Validation Accuracy Vs Epoch for Encoder Decoder Architecture with Attention using Word2Vec Word Embedding

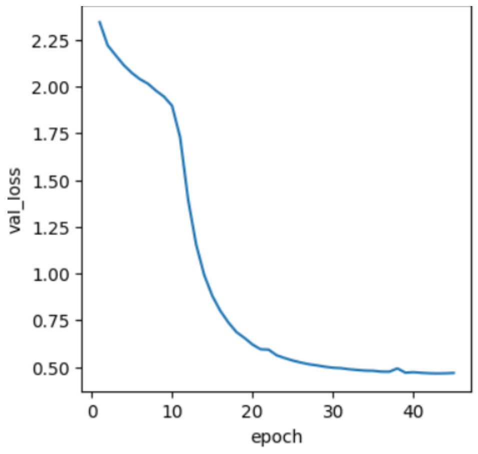


[a] [b]

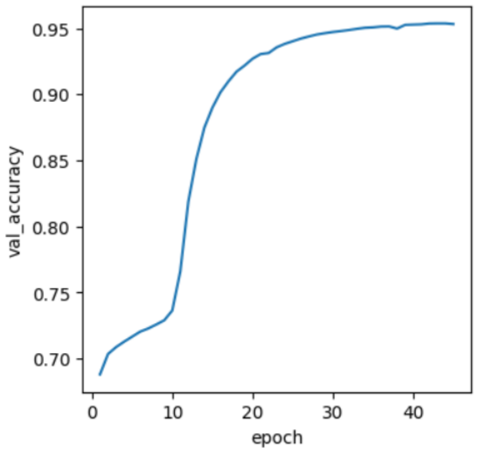
Figure 5.6: [a] Validation Loss Vs Epoch and [b] Validation Accuracy Vs Epoch for Encoder Decoder Architecture with Attention using Glove Word Embedding

5.2.4 Model Comparison

Table 5.3 indicates that the performance of Sequence2Sequence Encoder Decoder model with attention mechanism (BLEU Score 0.76631) outperformed the Seq2Seq Encoder Decoder model without Attention (BLEU Score 0.30497), as attention mechanism allows the model to put focus



[a]



[b]

Figure 5.7: [a] Validation Loss Vs Epoch and [b] Validation Accuracy Vs Epoch for Encoder Decoder Architecture with Attention using Fasttext Word Embedding

on the more significant part of the input and also it helps to learn the long sentences properly.

Model	Embedding	Validation Accuracy (%)	BLEU Score	Model Size (Bytes)	Model Parameters	Inference Time(ms)
Encoder Decoder	Trainable	69.55	0.30497	255761972	63934283	177
Encoder Decoder (Attention)	Trainable	99.02	0.76631	331019284	82746700	186

Table 5.3: Performance of Seq2Seq Encoder Decoder model with and without attention mechanism

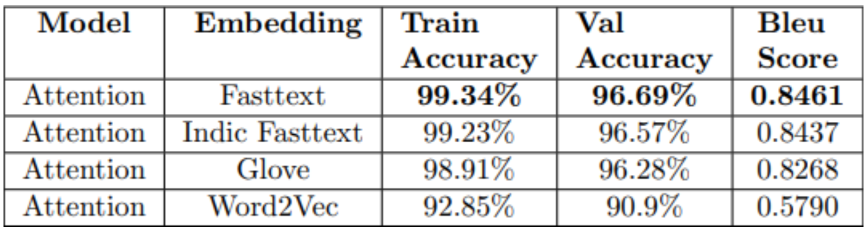
Table 5.4 indicates that the best performance has been achieved by the Encoder Decoder model with attention trained using Fasttext word embedding with a validation accuracy of 96.55% and BLEU Score of 0.8458 over others. Also a list of grammatical error correction for Hindi using best performing trained model (trained using Fasttext word Embedding) has been shown in Figure 5.9.

Model	Embedding	Training Accuracy (%)	Validation Accuracy (%)	BLEU Score (Greedy Search)
Encoder Decoder(Attention)	Trainable	99.95	99.02	0.76631
Encoder Decoder(Attention)	Word2Vec	98.08	95.44	0.7796
Encoder Decoder(Attention)	Glove	98.86	96.15	0.82002
Encoder Decoder(Attention)	Fasttext	99.30	96.55	0.8458

Table 5.4: Results obtained for different models trained using different embeddings

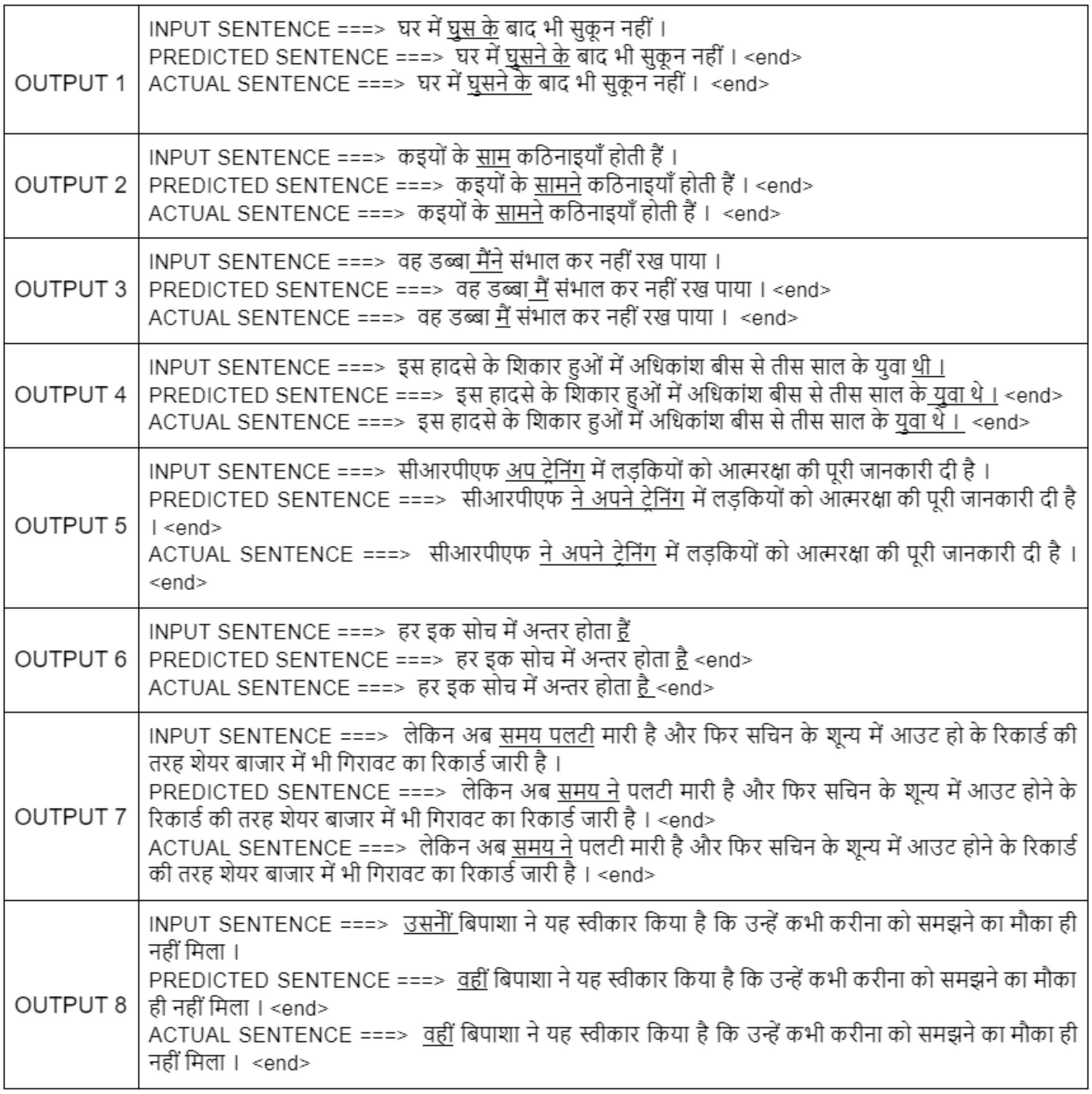
Model	Embedding	Model Size(Bytes)	Model Parameters	Inference Time(ms)
Encoder Decoder(Attention)	Trainable	331019284	82746700	200
Encoder Decoder(Attention)	Word2Vec	195513652	48870508	273
Encoder Decoder(Attention)	Glove	195513652	48870508	253
Encoder Decoder(Attention)	Fasttext	195513652	48870508	345

Table 5.5: Trained model size, Model Parameters and the Inference Time(ms) required for grammatical correcting one incorrect sentence.



Model	Embedding	Train Accuracy	Val Accuracy	Bleu Score
Attention	Fasttext	**99.34%**	**96.69%**	**0.8461**
Attention	Indic Fasttext	99.23%	96.57%	0.8437
Attention	Glove	98.91%	96.28%	0.8268
Attention	Word2Vec	92.85%	90.9%	0.5790

Figure 5.8: Results obtained by Garg[6] using Vyakaranly Word level Hindi Models



OUTPUT 1	INPUT SENTENCE ===> घर में घुस के बाद भी सुकून नहीं । PREDICTED SENTENCE ===> घर में घुसने के बाद भी सुकून नहीं । <end> ACTUAL SENTENCE ===> घर में घुसने के बाद भी सुकून नहीं । <end>
OUTPUT 2	INPUT SENTENCE ===> कइयों के साम कठिनाइयाँ होती हैं । PREDICTED SENTENCE ===> कइयों के सामने कठिनाइयाँ होती हैं । <end> ACTUAL SENTENCE ===> कइयों के सामने कठिनाइयाँ होती हैं । <end>
OUTPUT 3	INPUT SENTENCE ===> वह डब्बा मैंने संभाल कर नहीं रख पाया । PREDICTED SENTENCE ===> वह डब्बा मैं संभाल कर नहीं रख पाया । <end> ACTUAL SENTENCE ===> वह डब्बा मैं संभाल कर नहीं रख पाया । <end>
OUTPUT 4	INPUT SENTENCE ===> इस हादसे के शिकार हुओं में अधिकांश बीस से तीस साल के युवा थी । PREDICTED SENTENCE ===> इस हादसे के शिकार हुओं में अधिकांश बीस से तीस साल के युवा थे । <end> ACTUAL SENTENCE ===> इस हादसे के शिकार हुओं में अधिकांश बीस से तीस साल के युवा थे । <end>
OUTPUT 5	INPUT SENTENCE ===> सीआरपीएफ अप ट्रेनिंग में लड़कियों को आत्मरक्षा की पूरी जानकारी दी है । PREDICTED SENTENCE ===> सीआरपीएफ ने अपने ट्रेनिंग में लड़कियों को आत्मरक्षा की पूरी जानकारी दी है । <end> ACTUAL SENTENCE ===> सीआरपीएफ ने अपने ट्रेनिंग में लड़कियों को आत्मरक्षा की पूरी जानकारी दी है । <end>
OUTPUT 6	INPUT SENTENCE ===> हर इक सोच में अन्तर होता हैं PREDICTED SENTENCE ===> हर इक सोच में अन्तर होता है <end> ACTUAL SENTENCE ===> हर इक सोच में अन्तर होता है <end>
OUTPUT 7	INPUT SENTENCE ===> लेकिन अब समय पलटी मारी है और फिर सचिन के शून्य में आउट हो के रिकार्ड की तरह शेयर बाजार में भी गिरावट का रिकार्ड जारी है । PREDICTED SENTENCE ===> लेकिन अब समय ने पलटी मारी है और फिर सचिन के शून्य में आउट होने के रिकार्ड की तरह शेयर बाजार में भी गिरावट का रिकार्ड जारी है । <end> ACTUAL SENTENCE ===> लेकिन अब समय ने पलटी मारी है और फिर सचिन के शून्य में आउट होने के रिकार्ड की तरह शेयर बाजार में भी गिरावट का रिकार्ड जारी है । <end>
OUTPUT 8	INPUT SENTENCE ===> उसनीं बिपाशा ने यह स्वीकार किया है कि उन्हें कभी करीना को समझने का मौका ही नहीं मिला । PREDICTED SENTENCE ===> वहीं बिपाशा ने यह स्वीकार किया है कि उन्हें कभी करीना को समझने का मौका ही नहीं मिला । <end> ACTUAL SENTENCE ===> वहीं बिपाशा ने यह स्वीकार किया है कि उन्हें कभी करीना को समझने का मौका ही नहीं मिला । <end>

Figure 5.9: Grammatical Error Correction for Hindi using best performing trained model.

6
Conclusion

This project was initially started by analyzing and carrying out several experiments to study the existing models available for GEC in English. After achieving a significant result for English (Table 5.1), the same method was applied to the Hindi language. As discussed in chapter 4, the architectures used are the Sequence2Sequence model with and without attention using different word embeddings such as Word2Vec, Glove, and Fasttext. Pre-trained models were used to generate the embeddings for Hindi and train the Sequence2Sequence model (with and without attention mechanism) from scratch. As shown in Chapter 5, section 5.2.4, We are able to reproduce the benchmark results obtained by Garg[6] in her work for GEC in Hindi. As expected, the model trained using the attention mechanism outperformed the one without attention (Table 5.3) as it allows to put focus on the more significant part of the input over others. Also, the best results were obtained from the model trained using the Fasttext word embedding (Table 5.4) over the ones trained with Word2Vec and Glove.

7
Future Work

- Spelling Correction can also be a part of the grammatical error correction task. This part can also be included in the architecture proposed by Garg[6].
- The Etoori [1] and HiWikEd [2] dataset have only one error per sentence. A more error inclusive dataset (dataset with multiple forms of errors and more than one error per sentence) can be prepared to train the model for better learning.
- As of now, only non-contextual-based word embeddings have been used. Dynamic embeddings like BERT, ELMo, MuRIL, etc., can also be incorporated to get the contextual vector representation of a word, thus helping in better learning.
- For evaluation BLEU score using Greedy decoding has been used. There are other more effective ways of predicting the highest probable sentence than this method, such as the BEAM Search approach, GLEU, etc., which can also be incorporated.
- The process can be generalized so the same architecture can be made open source and used by other low-resource Indian languages for GEC tasks.

References

[1] P. Etoori, M. Chinnakotla, and R. Mamidi, *“Automatic spelling correction for resource-scarce languages using deep learning,” in Proceedings of ACL 2018, Student Research Workshop,* 2018*, p. 146–152.*

[2] A. Sonawane, S. Kumar, Vishwakarma, B. Srivastava, and A. K. Singh, *“Generating inflectional errors for grammatical error correction in hindi,” in Proceedings of the 1st Conference of the Asia Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing: Student Research Workshop,* 2020 *, p. 165–171.*

[3] C. Bryant, M. Felice, E. Andersen, and T. Briscoe, *“The bea-2019 shared task on grammatical error correction,” in In Proceedings of the 14th Workshop on Innovative Use of NLP for Building Educational Applications (BEA-2019), Association for Computational Linguistics,* 2019, *p. 52–75.*

[4] H. T. Ng, S. M. Wu, T. Briscoe, C. Hadiwinoto, R. H. Susanto, and C. Bryant, *“The conll-2014 shared task on grammatical error correction,” in In Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task (CoNLL-2014 Shared Task),* 2014.

[5] K. Saurav, K. Saunack, D. Kanojia, and P. Bhattacharyya, *“A passage to india”: Pre-trained word embeddings for indian languages,” in Proceedings of the 1st Joint SLTU and CCURL Workshop (SLTU-CCURL 2020),* 2020, *pp. 352–357.*

[6] V. Garg *”Vyakaranly”: Grammatical Error Correction for Indian Languages, Master’s Thesis, IIT Jodhpur,* 2022

[7] B. Kaur, H. Singh, *“Design and Implementation of HINSPELL -Hindi Spell Checker using Hybrid approach”,* 2015.

[8] A. Pal and A. Mustafi, *“Vartani Spellcheck – Automatic Context-Sensitive Spelling Correction of OCR-generated Hindi Text Using BERT and Levenshtein Distance”,* 2020.

[9] M. Mittal, S. Sharma and A. Sethi *” Detection and Correction of Grammatical Errors in Hindi Language Using Hybrid Approach” . International Journal of Computer Sciences and Engineering* 2019.

[10] J. Dodge, M. Sap, A. Marasovic, A. William, G. Gabriel, D. Groeneveld and M. Gardner, *”Documenting the English Colossal Clean Crawled Corpus”* 2021.

[11] I. Sutskever, O. Vinyals, and Q. V. Le, *“Sequence to Sequence Learning with Neural Networks”,* 2014.

[12] A. Vaswani, N.M. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser and I. Polosukhin,(2017). *“Attention is All you Need”,* 2017.

[13] G. Tao, X. Zhang, F. Wei, and M. Zhou, *"Automatic Grammatical Error Correction for Sequence-to-sequence Text Generation: An Empirical Study". In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics,* 2019.

[14] A. Schmaltz, Y. Kim, A. M. Rush, and S. M. Shieber, *"Sentence-Level Grammatical Error Identification as Sequence-to-Sequence Correction",* 2016.