Thesis Mid Term for the Degree of Master of Computer Information System

Nepali Text Part Of Speech Tagging Using Different Deep Learning Algorithms

Tika Ram Khojwar
(2020-2-92-0018)

Nepal College of Information Technology
Faculty of Management Studies
Pokhara University, Nepal

September, 2023

ACKNOWLEDGEMENTS

First of all, I would like to express my deep gratitude to Dr. Pradip Paudyal, my supervisor, for his unwavering help and insightful comments during my research work. His constructive feedback was the reason to improve my ideas and increase the overall quality of the work.

I'd also want to thank Mr. Saroj Shakya, the master's Program coordinator, for his amazing coordination and support. He provided me with tremendous possibilities to further my studies and academic aspirations.

As an NCIT student, I owe a debt of gratitude to the entire faculty for their accomplishments constant availability, and advice. Their provision of the right tools and advice contributed significantly to the successful outcome of my work dissertation.

My heartfelt thanks go to my parents for their constant advice, encouragement, and support in difficult times. Their belief in me has never wavered and I owe my success to their constant motivation.

Finally, I would like to thank my friend Mr. Sujan Paudel. His camaraderie, support, and assistance played a crucial role in getting me through this thesis journey.

In summary, the road to completing this thesis would not be long possible without the generous help and support of my mentors, educators, parents, and colleagues. I express my deep gratitude to you to all of them.

ABSTRACT

POS tagging is an essential and foundational task in numerous Natural Language Processing (NLP) applications, such as machine translation, text-to-speech conversion, question answering, speech recognition, word sense disambiguation, information retrieval, text summarization, Named Entity Recognition, and sentiment analysis, among others. POS tagging entails assigning the correct tag to each token in the corpus, considering its context and the language's syntax. An optimal part-of-speech tagger plays a crucial role in computational linguistics. Its importance cannot be emphasized enough because inaccuracies in tagging can greatly affect the performance of complex natural language processing systems. In this work, deep learning algorithms GRU, LSTM, BiLSTM and mBERT have been implemented for fine grained Nepali POS-tagging. The models have been trained - using genism's Word2Vec word embedding technique. Experiment results show that among the tested models, BiLSTM outperform the rest with F1 score of 99.76 % for fine grained Nepali text POS tagging on the Nepali Monolingual Written Corpus.

Keywords: POS Tagging, Nepali Text, Natural Language Processing, GRU, LSTM, BiLSTM, mBERT.

Table of content

Title	Page
Acknowledgements	i
Abstract	ii
Table of contents	iii
List of tables	vi
List of figures	vii
Abbreviations/Acronyms	ix
CHAPTER 1	
INTRODUCTION	
1.1. Background	1
1.2. Statement of problem	1
1.3. Research questions	2
1.4. Research objectives	2
1.5. Significance/Rationale of the study	2
CHAPTER 2	
LITERATURE REVIEW	
2.1. Literature Review	3
CHAPTER 3	
METHODOLOGY	
3.1. Data Collection	9
3.2. Pre-processing	10
3.2.1. Extract Sentenses and POS Tags from xml files	10
3.2.2. Word Embedding with Word2Vec	11
3.2.3. Create vocabulary and POS Tags Mapping	12

3.2.4. Convert Words and Tags to Indices 12
3.2.5. Pad Sequences 12
3.3. Model 13
3.3.1. GRU 13
3.3.2. LSTM 14
3.3.3. BiLSTM 15
3.3.4 mBERT 16
3.4. Validation Criteria 16
3.4.1. Confusion Matrix 17
3.4.1.1. Accuracy 17
3.4.1.2 Precision 18
3.4.1.3 Recall 18
3.4.1.4 F1 Score 18
3.4.2. K-fold Cross Validation 19
3.5. Coding-wise Methodology 20
3.6. Tools and Environment 21

CHAPTER 4

RESULTS AND DISCUSSION

4.1. Experiment using GRU 24
4.2. Experiment using LSTM 27
4.3. Experiment using BiLSTM 31
4.4. Comparing Results Between GRU, LSTM and BiLSTM 35

CHAPTER 5

CONCLUSION AND TASK TO BE COMPLETED

5.1. Conclusion 38
5.2. Task To Be Completed 38

APPENDIX A: GANTT CHARTS 39
REFERENCES 40

List of table

Title	Page
Table 4.1 Comparision between three Deep Learning Models based on Loss, Accuracy, Val_loss, Val_accuracy	35
Table 4.2 Comparison between three Deep Learning Models based on Precision, Recall, F1 score and Accuracy	36
Table 4.3: Comparison between 3 Deep Learning Models based on Testing Dataset	36

List of figures

	Title	Page
Fig. 3.1	Proposed Methodology for POS Tagging	8
Fig. 3.2	Sample of Training Dataset	9
Fig. 3.3	Tags distribution in the corpus	10
Fig. 3.4	Vector representation of 'असोज'	11
Fig. 3.5	Words similar to 'असोज'	11
Fig. 3.6	GRU model	14
Fig. 3.7	LSTM cell with gating controls	15
Fig. 3.8	Architecture of the BiLSTM network	16
Fig. 3.9	Confusion Matrix	17
Fig. 3.10	Coding-wise Methodology	20
Fig. 4.1	Summary of GRU Model	24
Fig. 4.2	History of training GRU model	25
Fig. 4.3	Accuracy and Loss over time of GRU model	25
Fig. 4.4	Evaluate GRU model using test dataset	26
Fig. 4.5	Confusion matrix for GRU model	26
Fig. 4.6	Precision, Recall, F1 score and Accuracy	27
Fig. 4.7	Output of user input sentences using GRU	27

Fig. 4.8 Summary of LSTM Model 28
Fig. 4.9 History of training LSTM model 28
Fig. 4.10 Accuracy and Loss over time of LSTM model 29
Fig. 4.11 Evaluate LSTM model using test dataset 29
Fig. 4.12 Confusion matrix for LSTM model 30
Fig. 4.13 Precision, Recall, F1 score and Accuracy 30
Fig. 4.14 Output of user input sentences using LSTM model 31
Fig. 4.15 Summary of BiLSTM Model 32
Fig. 4.16 History of training BiLSTM model 32
Fig. 4.17 Accuracy and Loss over time of BiLSTM model 33
Fig. 4.18 Evaluate BiLSTM model using test dataset 33
Fig. 4.19 Confusion matrix for BiLSTM model 34
Fig. 4.20 Precision, Recall, F1 score and Accuracy 34
Fig. 4.21 Output of user input sentences using BiLSTM model 35

List of abbreviation/acronyms

NLP	Natural Language Processing
POS	Part Of Speech
HMM	Hidden Markov Model
SVM	Support Vector Machine
ANN	Artificial Neural Network
GRU	Gated Recurrent Unit
LSTM	Long Short-Term Memory
BiLSTM	Bi-directional Long Short-Term Memory
mBERT	Multilingual Bidirectional Encoder Representations from Transformers

CHAPTER 1

INTRODUCTION

1.1 Background

Natural Language Processing is a field of artificial intelligence that focuses on the interaction between computers and human language. NLP involves the development of algorithms and techniques to enable computers to understand, interpret, and generate human language in a way that is meaningful and useful.

In NLP, Part-of-Speech tagging is a fundamental task. It involves assigning POS tags (Noun, pronoun, verb, adjective, adverb, preposition etc.) to each word in a sentence of a natural language. The input for the algorithm consists of a sequence of words in a natural language sentence and a predefined set of POS tags. The output is the most suitable POS tag assigned to each word in the sentence. POS tagging provides valuable information about a word and its neighboring words, which proves beneficial for various advanced NLP tasks like speech and natural language processing applications, semantic analysis, machine translation, text-to-speech conversion, question answering, speech recognition, word sense disambiguation and information retrieval, text summarization, Named entity recognition and more [2] [7] [9].

Nepali is a morphologically rich language. One of the characteristics features of the Nepali language is its rich inflectional system, especially the verbal inflection system. A verb in Nepali can easily display more than 20 inflectional forms while encoding different morphological features, including aspect, mood, tense, gender, number, honorifics, and person [1].

1.2 Statement of the problem

Nepali is a morphologically rich language. Nepali POS tagging is not simple as giving a sequence of tags for a sequence of tokens. The ambiguity is higher for languages with complex and rich morphology, like Nepali. For this reason, most work in Nepali POS tagging is limited to coarse-grained tagsets that do not disambiguate morphological features such as gender, number, honorifics, and person encoded within a word. Limited work has been done in terms of fine-grained POS Tagging for Nepali. Fine-grained distinctions of grammatical categories of words are useful in improving the accuracy and

quality of NLP applications for Nepali. Our primary focus in this research is fine-grained Nepali POS tagging. We assess different neural network architectures, including GRU, LSTM, BiLSTM and mBERT for the task. We also implement genism's Word2Vec word embedding for fine-grained Nepali POS tagging.

1.3 Research questions

Some research questions are:

- How the most popular DL algorithms GRU, LSTM, BiLSTM and mBERT can be used for Nepali POS-tagging problem?
- What would be the performance of GRU, LSTM, BiLSTM and mBERT models for Nepali POS-tagging problem on the dataset?

1.4 Research objectives

The best model depends on various factors, including the availability of training data, Language, computational resources, and the specific requirements of the application. So, experiment with different models and compare their performance on the specific task or dataset to determine the most suitable model.

The main objective of this paper is:

- To implement and optimize the parameters of Deep Learning Algorithms to achieve the better results on the Nepali Monolingual Written Corpus.
- To analyze the performance of GRU, LSTM, BiLSTM, and mBERT for nepali text POS tagging.

1.5 Significance/Rationale of the study

The main significance of this research is to assign the correct tag to the word of text. Only correct assignment of the tag gives correct sense of the words. Which proves beneficial for various advanced NLP tasks like speech and natural language processing applications, semantic analysis, machine translation, text-to-speech conversion, question answering, speech recognition, word sense disambiguation and information retrieval, text summarization, Named entity recognition.

CHAPTER 2

LITERATURE REVIEW

2.1 Literature Review

There are only few researches have been done in the field of POS tagging for Nepali language. Some of them used statistical model (HMM) for identifying the tags while some used supervised machine learning model and some used supervised deep learning model to train the model.

I. Shrestha, S. S. Dhakal [1] applied three deep learning models: BiLSTM, BiGRU, and BiLSTM-CRF for fine-grain POS tagging for the Nepali language. It used Nepali National Corpus (NNC). It has 17 million manually and semi-manually words tagged with 112 POS-tags. Results showed that deep learning models could capture fine-grained morphological features like gender, person, number, and honorifics that are encoded within words in highly inflectional languages like Nepali with a large enough dataset. This study trained all the models using two embedding: pre-trained multi-lingual BERT and randomly initialized Bare embedding. It found that training a randomly initialized Bare embedding is better than the ones trained using large pre-trained multi-lingual BERT embedding for downstream tasks in Nepali like POS tagging. Among the tested models, the BiLSTM-CRF with the Bare embedding performed the best and achieved a new state-of-the-art F1 score of 98.51% for fine-grained Nepali POS tagging. This research contributes to the advancement of NLP techniques tailored specifically for the Nepali language.

Sarbin Sayami et al. [4] addressed the implementation and comparison of various deep learning-based POS taggers for Nepali text. The examined approaches include LSTM, GRU, BiLSTM and mBERT. These models were trained and evaluated using Nepali English parallel corpus annotated with 43 POS tag and contains nearly 88000 words which were collected from Madan Puraskar Pustakalaya. The design of that Nepali POS Tagset was inspired by the PENN Treebank POS Tagset. The dataset was divided into three sections i.e. training, development and testing. The accuracy obtained for simple RNN, LSTM, GRU and Bidirectional LSTM were 96.84%, 96.48%, 96.86% and 97.27%

respectively. Therefore, Bi-directional LSTM outperformed all other three variants of RNN.

Greeshma Prabha et al. [9] proposed a deep learning based POS tagger for Nepali text which was built using Recurrent Neural Network (RNN), Long Short Term Memory Networks (LSTM), Gated Recurrent Unit (GRU) and their bidirectional variants. It used POS Tagged Nepali Corpus generated by translating 4325 English sentences from the PENN Treebank corpus tagged with 43 POS tags. The results demonstrated that the proposed model outperforms existing state-of-the-art POS taggers with an accuracy rate exceeding 99%. This research contributed to the field by showcasing the effectiveness of deep learning techniques in improving POS tagging for Nepali text.

Ashish Pradhan et al. [2] presented a comprehensive study and comparing two techniques, HMM and GRNN, for POS Tagging in Nepali text. The POS taggers aimed to address the issue of ambiguity in Nepali text. Evaluation of the taggers was performed using corpora from TDIL (Technology Development for Indian Languages) which contained a total of 424716 tagged words with 39 tags, tags followed the guidelines of ILCI (Indian Languages Corpora Initiative), BSI (Bureau of Indian Standard), with implementation carried out using Python and Java programming languages, along with the NLTK Toolkit library. The achieved accuracy rates were as follows: 100% for known words (without ambiguity), 58.29% for ambiguous words (HMM), 60.45% for ambiguous words (GRNN), and 85.36% for non-ambiguous unknown words (GRNN). Although the GRNN tagger achieved the accuracy as high as the HMM Tagger, it failed or became unstable when the training dataset was greater than 7000 words, while the HMM Tagger was trained with more than 400000 words with corresponding tags. A total of 4000 words were used for testing on both HMM and GRNN taggers.

Archit Yajnik [6] focused on POS tagging for Nepali text using the GRNN. Because GRNN was less expensive as compared to standard algorithms viz. Back propagation, Radial basis function, support vector machine etc. And also neural network was usually much faster to train than the traditional multilayer perceptron network. The corpus had total 7873 Nepali words with 41 tags. Out of which 5373 samples were used for training and the remaining

2500 samples for testing. The results showed that 96.13% of words were correctly tagged on the training set, while 74.38% were accurately tagged on the testing set using GRNN. To compare the performance, the traditional Viterbi algorithm based on HMM was also evaluated. The Viterbi algorithm achieved classification accuracies of 97.2% and 40% on the training and testing datasets, respectively. The study concluded that the GRNN-based POS tagger demonstrated more consistency compared to the traditional Viterbi decoding technique. The GRNN approach yield a higher accuracy on the testing dataset, suggesting its potential for improved POS tagging in Nepali text compared to the Viterbi algorithm.

Archit Yajnik [8] introduced POS tagging for Nepali text using three Artificial Neural Network (ANN) techniques. A novel algorithm was proposed, extracting features from the marginal probability of the Hidden Markov Model. These features were used as input vectors for Radial Basis Function (RBF) network, General Regression Neural Networks (GRNN), and Feed forward neural network. The training database contained 42100 words whereas the testing set consisted of 6000 words with 41 tags. The GRNN-based POS tagging technique outperformed the others, achieving 100% accuracy for training and 98.32% accuracy for testing. This research contributed to Nepali POS tagging by presenting a novel algorithm and highlighting the effectiveness of the GRNN approach.

Archit Yajnik [3] focused on POS tagging for Nepali text using the HMM and Viterbi algorithm. The study revealed that the Viterbi algorithm outperforms HMM in terms of computational efficiency and accuracy. Database was generated from NELRALEC Tagset with 41 tags. A report on Nepali Computational Grammar was made available by Prajwal Rupakheti et al. Database contained 45000 Nepali words with corresponding Tag, out of which 15005 samples were randomly collected for testing purpose. The Viterbi algorithm achieved an accuracy rate of 95.43%. The article also provided a detailed discussion of error analysis, specifically examining the instances where mismatched occurred during the POS tagging process.

Abhijit Paul et al. [5] discussed HMM based POS tagging for the Nepali language. The study evaluated the HMM tagger using corpora from Technology Development for Indian Languages (TDIL) which contained around 1,50,839 tagged words and tagset consist of 42

tags including generic attributes and language specific attribute values. It had been followed the guidelines of ILCI (Indian Languages Corpora Initiative), BSI (Bureau of Indian Standard). The implementation was done using Python and the NLTK library. The HMM-based tagger achieved an accuracy of over 96% for known words but the system was not performing well for the text with unknown words yet. Overall, the paper provided insights into the effectiveness of HMM for Nepali POS tagging and highlights areas for future improvement.

Tej Bahadur Shahi et al. [7] focused on SVM based POS tagger for Nepali language which used the dictionary as a primary resources. This dictionary was collected from the Final Nepali Corpus which contained only 11147 unique words. The POS tagging approaches like rule-based and HMM cannot handle many features that would generally be required for modeling a morphologically rich language like Nepali. SVM was efficient, portable, scalable and trainable. So, this paper proposes a SVM based tagger. The SVM tagger constructed feature vectors for each word in the input and classifies them into one of two classes using a One Vs Rest approach. The SVM algorithm achieved an accuracy rate of 96.48% for known words, 90.06% for unknown words and 93.27% in overall. That means SVM tagger demonstrated strong performance for known words. In comparison to rule-based and Hidden Markov Model (HMM) approaches, the SVM-based tagger exhibits a slightly higher overall accuracy.

Despite the considerable progress made in Nepali POS tagging, the existing literature reveals several persistent research gaps that warrant attention. First and foremost, the majority of studies tend to employ coarse-grained tagsets, neglecting the need for fine-grained distinctions that can disambiguate morphological features such as gender, number, honorifics, and person encoded within Nepali words. This limitation hampers the development of more accurate and nuanced NLP applications for the language. Furthermore, while some studies have delved into the realm of fine-grained POS tagging, there is a notable lack of consensus on the most effective neural network architectures for this task. The comparative analysis of models, such as GRU, LSTM, BiLSTM, and mBERT, across different studies highlights the need for a standardized evaluation framework. Additionally, the choice of word embeddings, including pre-trained models

like BERT and custom embeddings like Word2Vec, introduces variability in performance outcomes, necessitating a comprehensive examination of their impact on fine-grained POS tagging.

CHAPTER 3

METHODOLOGY

In this chapter, we present the proposed method for determining the POS tags of the provided text. The following figure provides an overview of the tagging process.

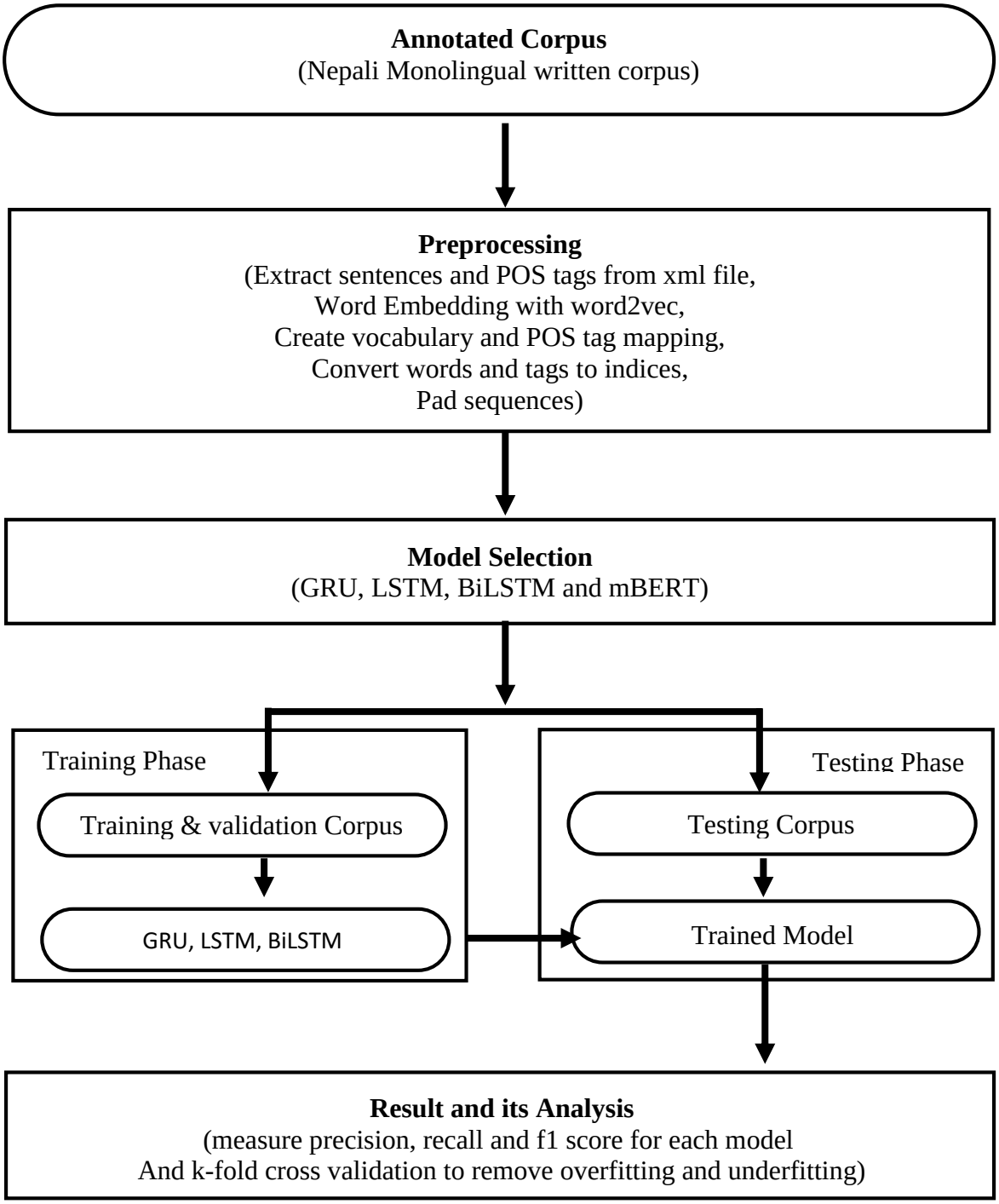


Fig 3.1: Proposed methodology for POS tagging

3.1 Data Collection

This research study has implemented morphologically annotated Nepali National Corpus (NNC). NNC is collection of xml files, has over 14 million words with 112 Nelralec tagset. Each xml files are in same format, sentences wrap within “s” element with “n” attribute and words are wrap within “w” element with “ctag” attribute. Ctag represents POS tag of crossponding word. The corpus is divided into two main parts: the core corpus (core sample) and the general corpus. The core sample (CS) is a compilation of Nepali written texts from 15 diverse genres, with each text containing 2000 words. These texts were published between 1990 and 1992. On the other hand, the general corpus (GC) comprises written texts gathered from various sources, including the internet, newspapers, books, publishers, and authors. Following is a sample of the xml file:

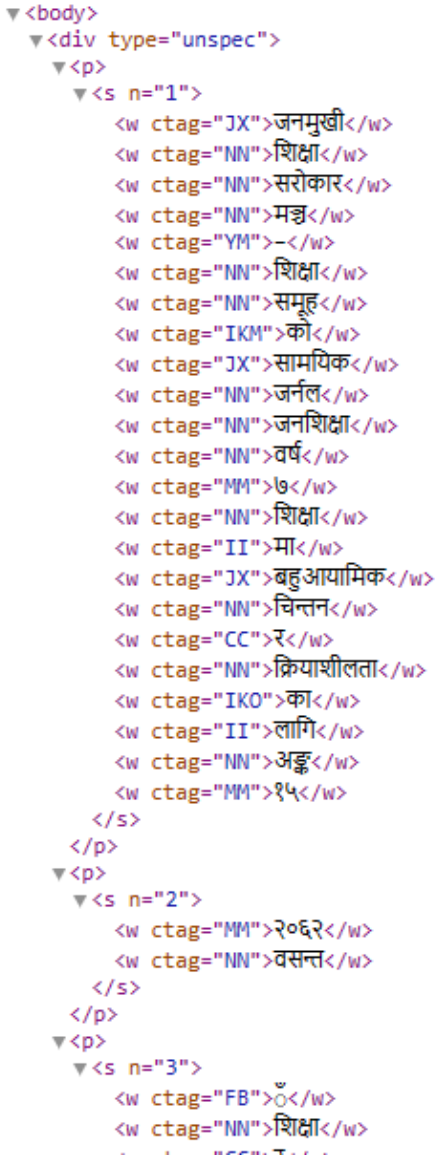


Fig. 3.2: Sample of training dataset

The tag distribution in the tag-set is depicted by the bar graph below:

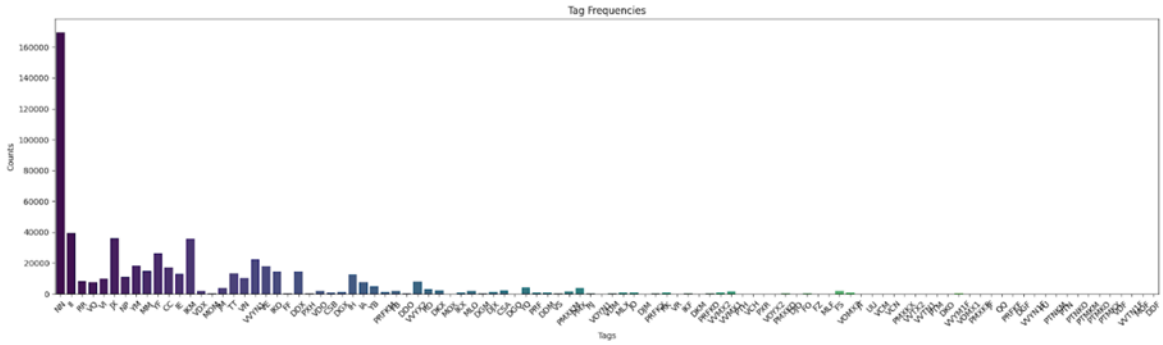


Fig. 3.3: Tags distribution in the corpus

The tag distribution chart shows and uneven distribution of the tags in the corpus. The first tag 'NN' has the highest number of occurrence totaling to 169767 whereas the second highest number of occurrence is for 'II' totaling 39387. There are tags with only a few hundred or tens of occurrences with few tags having 1 occurrences. This shows a very non-uniform distribution of the tags in the dataset.

3.2 Pre-processing

Only good dataset can give good output. To make good dataset, we need to transform the text into something meaningful that the algorithm can use. Preprocessing includes the extract sentences and POS tags from xml file, word embedding with word2vec, create vocabulary and POS tag mapping, convert words and tags to indices and pad sequences.

3.2.1 Extract Sentences and POS Tags from XML files

Nepali monolingual written corpus files are in XML format. We cannot use XML files as input to train a deep learning model. Therefore, we first extract sentences and their corresponding parts-of-speech tags in list form. For example:

all_sentences = [['सत्तापक्ष', 'सित', 'नजीक', 'रहेर', 'काम', 'गर्न', 'विपक्षीदल', 'प्रतिवद्ध'], ['काठमाडौं', ',', 'असोज', '१६', '।'], …]

all_pos_tags = [['NN', 'II', 'RR', 'VQ', 'NN', 'VI', 'NN', 'JX'], ['NP', 'YM', 'NN', 'MM', 'YF'], ...]

3.2.2 Word Embedding with Word2Vec

"Word2Vec is a popular word embedding technique in the fields of NLP and ML. It is used to represent words as dense vectors in a continuous vector space, where words with similar meanings are located closer to each other in this space. Word2Vec models are trained on large text corpora and are capable of capturing semantic relationships between words. To achieve this, we first created a Word2Vec model using Gensim in Python. We then passed all_sentences as an argument to the created model, resulting in the embedding_model. Let's examine the vector representation of a 'असोज' word and find similar words using the embedding_model."

```
Vector representation of 'असोज':
[-0.61241823  0.38024774  0.04081193  0.13905744  0.14232595 -0.45095217
 -0.10359821  0.60562104 -0.03477165 -0.28667906 -0.25833422 -0.4752194
 -0.5728589   0.5555608  -0.08489339 -0.28250316 -0.12623817 -0.11951029
  0.40326816 -0.2251008  -0.21760793  0.16786504 -0.00678924  0.01699983
  0.02061158  0.00430579 -0.15978703  0.23189414 -0.45993102 -0.00154939
  0.14740264 -0.48094407  0.04010671 -0.2447216  -0.21270886  0.1505602
  0.48446426 -0.14220607 -0.44455615 -0.04188625 -0.1867587  -0.2132533
 -0.34608984  0.04800472  0.5762164   0.0762004  -0.2485139  -0.07760854
  0.19436257  0.3417582   0.33642417 -0.16990125 -0.06390376 -0.02860144
 -0.15383117 -0.26653397  0.32357383 -0.342046   -0.41413072 -0.07138456
  0.00849417  0.26246512 -0.04986182  0.21560971 -0.11352767  0.23363551
  0.3411019   0.43029702 -0.0839266   0.485876   -0.30533066  0.12078191
  0.4424832  -0.2048535   0.4222532   0.05982683  0.02288452  0.6799946
 -0.08957221 -0.03917739 -0.31971005 -0.4168772  -0.31055528 -0.06093651
 -0.36312857  0.07146426  0.1187695   0.2879384  -0.08672892 -0.02928283
  0.2365958   0.1584901   0.18106869  0.02508811  0.5744465   0.428506
  0.3208022  -0.03398637 -0.22060467  0.2853352 ]
```

Fig. 3.4: Vector representation of 'असोज'

```
Words similar to 'असोज':
फागुन 0.9912300109863281
वैशाख 0.9903742671012878
कात्तिक 0.9891312122344971
```

Fig. 3.5: Words similar to 'असोज'

3.2.3 Create Vocabulary and POS Tag Mapping

Vocabulary and POS tag mappings convert words and tags into numerical indices for deep learning models. For vocabulary and POS tag mappings, we firstly create vocab set and pos_tags_set set for unique words and unique POS tags respectively. Then create word_to_idx dictionary to maps each unique word in the vocabulary to a unique index. And same for the tag_to_idx. Then create idx_to_tag dictionary. It is reverse of the tag_to_idx dictionary. We also use 'UNK' token to handle out-of-vocabulary words or tags that are not present in the initial training data. Following are sample of the word_to_idx and tag_to_idx dictionaries.

word_to_idx = {'समर्थित': 1, 'व्यक्तिविशेष': 2, 'खुलेछन्': 3, 'चर्कँदो': 4, …}

tag_to_idx = {'IE': 1, 'VVTN1': 2, 'DGF': 3, 'PTH': 4, 'YQ': 5, …}

3.2.4 Convert Words and Tags to Indices

It is the process where we converted words and POS tags to their corresponding numerical indices using the mappings created earlier (word_to_idx and tag_to_idx). Each inner list represents a sentence, and it contains numerical indices corresponding to the words or POS tags in that sentence. Following are the sample of sentences_indices and pos_tags_indices list of lists.

sentences_indices = [[24283, 33809, 29171, 17993, 31405, 11612, 27897, 29351],

[39860, 14035, 16097, 37307, 10206], …]

pos_tags_indices = [[20, 89, 25, 24, 20, 43, 20, 88], [44, 39, 20, 55, 46], …]

3.2.5 Pad Sequences

When training neural networks algorithms that expect fixed-length or uniform-length input sequences. The purpose of padding is to ensure all sequences have the same length. Padding is often necessary when working with sequences of varying lengths, which is common in NLP tasks.

3.3 Model

There are several models have been widely used and achieved good performance in POS tagging tasks. Different models have their own different features and specific task. There is no single "best" model for POS tagging, as the effectiveness of a model can vary depending on factors such as the dataset, language, and specific requirements of the task. According to previous research Deep Learning algorithms based models gives better accuracy in testing dataset and can deal with ambiguous, unknown words as compare to rule based, statistical and machine learning algorithms for POS tagging.

3.3.1 GRU

A GRU is a type of RNN architecture that have shown effectiveness in various NLP tasks, including POS tagging. GRU has two gates, an update gate and a reset gate, which control information flow in and out of the hidden state [4]. The update gate helps retain relevant information from the past while the reset gate helps update the hidden state with new information. GRUs can result in fewer parameters, making them computationally more efficient and easier to train, especially in scenarios with limited data. GRUs share parameters across different positions in a sequence which allows the model to generalize patterns learned at one position to other positions, which is beneficial for tasks like POS tagging where similar linguistic patterns may occur at different positions in a sentence. Pre-trained word embedding can be used as input to a GRU model for POS tagging, leveraging transfer learning. The GRU can then adapt to the specific POS tagging task, benefiting from the semantic relationships captured by pre-trained embedding. GRUs have demonstrated competitive performance in various NLP tasks, and they are considered a state-of-the-art choice for certain applications, especially in cases where computational efficiency is crucial.

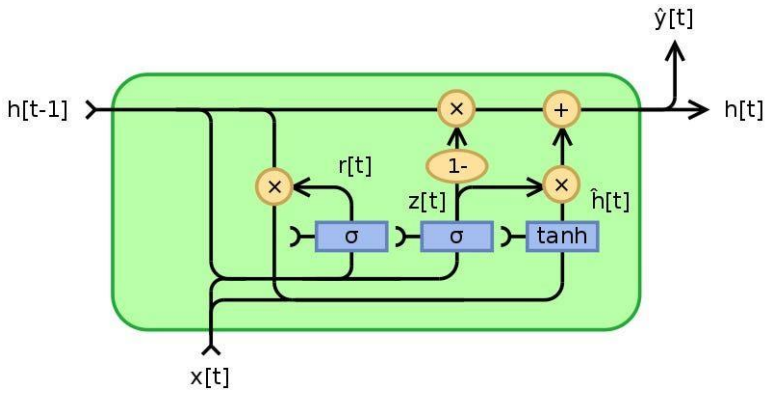


Fig. 3.6: GRU model [12]

3.3.2 LSTM

LSTM is a type of RNN architecture that have been widely used in NLP tasks, including POS tagging. LSTMs are designed for sequential data processing, which aligns well with the nature of POS tagging. In POS tagging, the order of words in a sentence is crucial for understanding the syntactic structure. LSTMs also have a memory cell that can retain information over long sequences, allowing the model to capture dependencies between words that are separated by significant distances within a sentence [9]. This is crucial for accurately assigning POS tags. Pre-trained word embedding can be used as input to an LSTM model for POS tagging, leveraging transfer learning. The LSTM can then adapt to the specific POS tagging task, benefiting from the semantic relationships captured by pre-trained embedding. LSTM can also handle variable-length sequences, making them suitable for POS tagging tasks where sentences have different lengths. LSTM have demonstrated state-of-the-art performance in various NLP tasks, including POS tagging. They have been widely adopted and have shown effectiveness in capturing complex linguistic patterns.

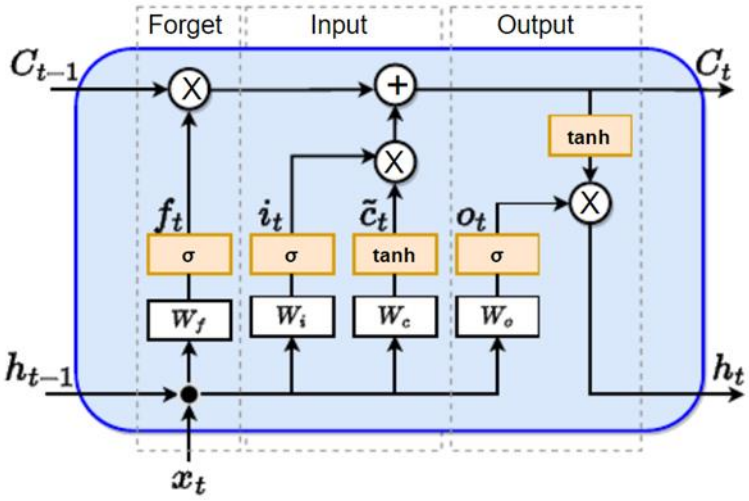


Fig. 3.7: LSTM cell with gating controls [13]

3.3.3 BiLSTM

BiLSTM is well-suited for POS tagging tasks due to their ability to capture contextual information from both the past and the future. BiLSTM can capture long-term dependencies in sequential data [4]. The bidirectional nature of BiLSTM helps in understanding the syntactic and semantic context in a more holistic way [9]. This is particularly important in POS tagging, where the correct assignment of a POS tag often relies on the broader linguistic context. POS tagging can be challenging when words have multiple possible POS tags depending on the context. BiLSTM, by considering both past and future context, are better equipped to handle such ambiguity and make more informed predictions. BiLSTM can handle variable-length sequences, making them suitable for POS tagging tasks where sentences have different lengths. BiLSTMs have demonstrated state-of-the-art performance in various NLP tasks, including POS tagging [1]. They are widely used in research and industry due to their effectiveness in capturing complex linguistic patterns.

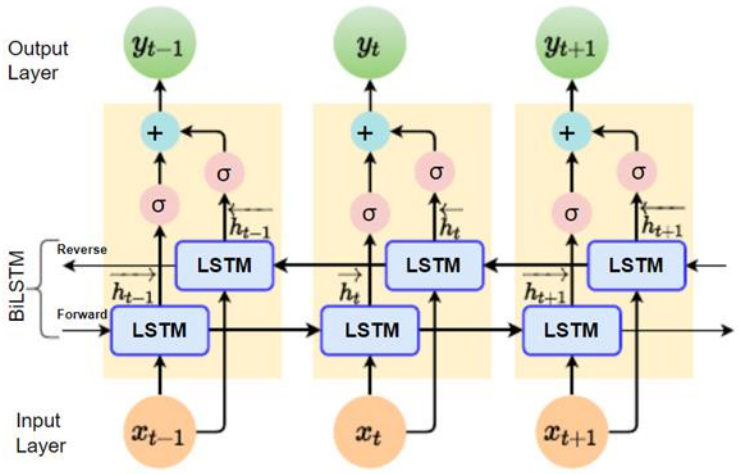


Fig. 3.8: Architecture of the BiLSTM network [13]

3.3.4 mBERT

Multilingual BERT is a BERT-based model, is pre-trained on large-scale multilingual corpora, allowing to learn rich contextual embedding for words. These embedding capture the syntactic and semantic relationships between words in context, which is beneficial for POS tagging. The bidirectional nature of Multilingual-BERET enables the model to understand the context in a more comprehensive manner, aiding in accurate POS tagging. Multilingual-BERT useful in POS tagging tasks where the same word may have different POS tags depending on its role in a sentence. Because it provides contextualized representations for words. Multilingual-BERT can handle POS tagging tasks in various languages without the need for language-specific models, offering a unified solution for languages with limited labeled data. Multilingual-BERT have demonstrated state-of-the-art performance in various NLP tasks, including POS tagging. Their effectiveness in capturing complex linguistic patterns has made them widely adopted in research and industry.

3.4 Validation Criteria

Once a model is developed, it is very important to check the performance of the model. To measure the performance of a predictor, there are commonly used performance metrics

such as confusion matrix. In classification problems, the primary source of performance measurements is confusion matrix.

3.4.1 Confusion Matrix

Confusion Matrix is a performance evaluation metric which provides a summary of the predictions made by a classification model, highlighting the correct and incorrect classifications across different classes. It is typically represented as a table with rows and columns corresponding to the predicted and actual classes, respectively. It helps in assessing the model's accuracy and identifying common types of errors.

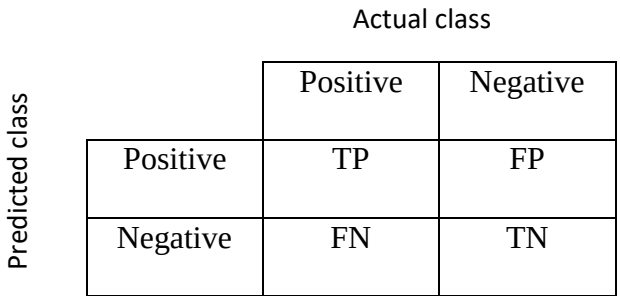


Fig. 3.9: Confusion Matrix

3.4.1.1 Accuracy

Accuracy is a commonly used metric in the evaluation of POS tagging. It measures the overall correctness of the model's predictions, regardless of the specific POS tags involved. It is easy to understand and communicate. It represents the ratio of correct predictions to the total number of instances and is often used for its clarity and simplicity. Accuracy provides a quick and initial assessment of the model's performance. It is often used in the early stages of model development and experimentation to gauge how well the model is learning from the training data. It is calculated as

$$\text{Accuracy} = (TP + TN) / (TP + TN + FP + FN) \quad (1)$$

3.4.1.2 Precision

Precision is an important metric in the evaluation of POS tagging. Precision measures the accuracy of the positive predictions made by the model, specifically, the proportion of instances predicted as a particular POS tag that are actually correct. High precision indicates that the model is effective at minimizing misclassification of instances as a particular POS tag. This is important for maintaining the quality and reliability of the POS tagging results. It is calculated as

$$\text{Precision} = TP / (TP + FP) \quad (2)$$

3.4.1.3 Recall

Recall is an important metric in the evaluation of POS tagging using deep learning algorithms. Recall, also known as sensitivity or true positive rate, measures the ability of a model to correctly identify all instances of a particular POS tag. Recall helps in assessing how well a model generalizes to unseen data by capturing the ability to correctly identify instances of POS tags not only in the training data but also in new and diverse texts. High recall ensures that the model does not overlook instances of important POS tags, especially those that might be less frequent but linguistically significant. It is calculated as

$$\text{Recall (Sensitivity or True Positive Rate)} = TP / (TP + FN) \quad (3)$$

3.4.1.4 F1 Score

The F1 score is a commonly used metric in NLP tasks, including POS tagging. The F1 score is particularly useful in situations where there is an imbalance between the classes or POS tags. For example, some POS tags may occur much more frequently than others. In such cases, accuracy alone may not be a reliable metric because a model could achieve high accuracy by simply predicting the majority class most of the time. The F1 score considers both false positives and false negatives, making it suitable for imbalanced datasets. It provides a balanced measure of a model's precision and recall. This is especially important

in POS tagging, where different POS tags may have different linguistic characteristics and challenges. It is calculated as

F1 Score = 2 * (Precision * Recall) / (Precision + Recall) (4)

3.4.2 K-fold Cross Validation

K-fold cross-validation is a common technique used in various Deep learning tasks, including POS tagging, to evaluate the performance and generalization ability of a model. K-fold cross-validation allows to make better use of the available data by partitioning it into K subsets, training and validating the model K times, each time using a different subset for validation. Nepali National Corpus is imbalanced (i.e., some POS tags have significantly fewer instances than others), K-fold cross-validation ensures that each fold has a representative distribution of instances from all classes, reducing the risk of biased evaluation. It also reduces the risk of overfitting or underfitting. In supervised deep learning model, there are two type of dataset one is known dataset (training dataset) and other is unknown dataset (validation and testing dataset). In this research, we split our dataset into 3 parts training, validation and testing set.

3.5 Coding-wise Methodology

The following figure shows the detailed coding-wise implementation of this research.

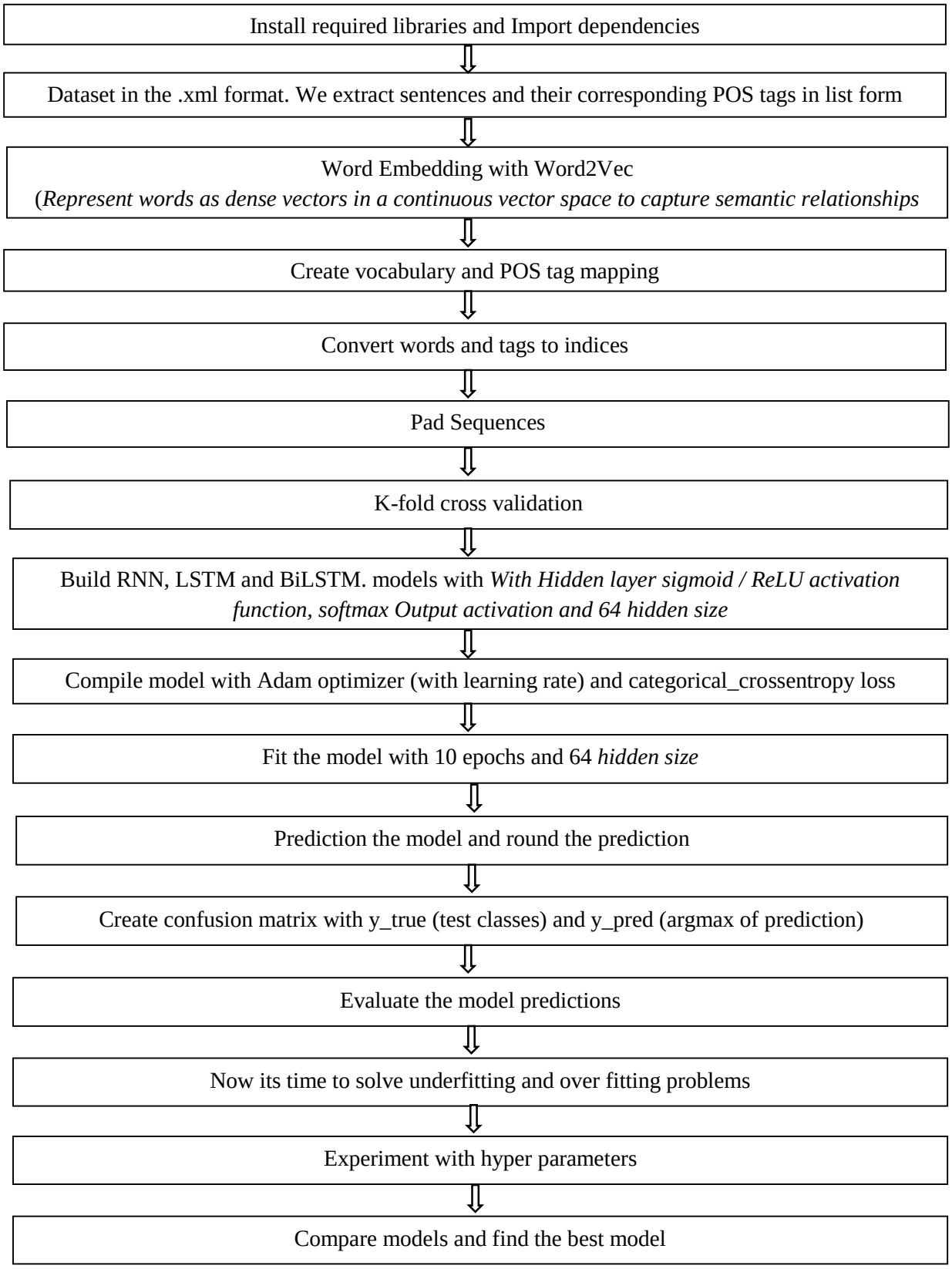


Fig. 3.10: Coding-wise Methodology

3.6 Tools and Environment

The proposed programming language for the entire project is Python. The following tools and libraries needs to be used:

a. Python: Python is a high-level, versatile programming language known for its simplicity and readability. It features a rich standard library, supports multiple programming paradigms, and is widely used in web development, data analysis, machine learning, and scientific computing. Python's extensive community and ecosystem make it a popular choice for a broad range of applications. The thesis is implemented on python 3.10.9 version.

b. Anaconda: Anaconda is an open-source platform and distribution for data science and machine learning that simplifies the management of Python and R libraries, environments, and dependencies. It provides a user-friendly interface for creating isolated environments and installing packages, making it a popular choice for data scientists and developers to streamline project setups and maintain version control of libraries. Anaconda also includes a package manager called conda, which facilitates package installation and environment management.

c. Tensorflow: TensorFlow is an open-source machine learning framework developed by Google that facilitates building, training, and deploying deep learning models. It offers a comprehensive ecosystem for various machine learning tasks, including neural networks, natural language processing, and computer vision. TensorFlow is known for its flexibility, scalability, and support for both research experimentation and production-level deployment.

d. Keras: Keras is a high-level deep learning framework for Python that simplifies the creation and training of neural networks. It offers an intuitive API for building complex models, supports multiple backends (e.g., TensorFlow), and is widely used for various machine learning tasks, including image recognition, natural language processing, and reinforcement learning. Its modularity, ease of use, and extensive community support make it a popular choice among researchers and practitioners.

e. NumPy: It is a fundamental Python library for numerical computations, providing support for multidimensional arrays and matrices along with a vast collection of mathematical functions. It is widely used in scientific computing, data analysis, and machine learning due to its efficiency and versatility. NumPy serves as a foundation for many other Python libraries and tools in the data science ecosystem.

f. Pandas: Pandas is a Python library for data manipulation and analysis, offering data structures like DataFrames and Series. It simplifies tasks like data cleaning, transformation, and exploration. Pandas is widely used in data science for handling and analyzing structured data efficiently.

g. Matplotlib: Matplotlib is a versatile Python library for creating static, animated, or interactive visualizations and plots. It provides a wide range of customizable options for creating charts, graphs, and other data visualizations. Matplotlib is commonly used in scientific computing, data analysis, and data presentation.

h. Seaborn: Seaborn is a Python data visualization library built on top of Matplotlib, designed to create informative and attractive statistical graphics. It simplifies the process of creating aesthetically pleasing visualizations for data analysis and exploration. Seaborn offers a high-level interface for creating various types of plots, including scatter plots, bar charts, and heatmaps, making it a valuable tool for data visualization in Python.

i. Scikit-learn (sklearn): It is a widely-used machine learning library in Python, known for its user-friendly API and comprehensive set of tools for classification, regression, clustering, dimensionality reduction, and more. In this thesis, sklearn used for splitting the dataset into training, validation and test sets by using train_test_split function. And plotting confusion matrix by using confusion_matrix function.

j. NLTK (Natural Language Toolkit): It is a Python library for natural language processing and text analysis. It provides a wide range of tools, resources, and corpora for tasks such as tokenization, stemming, part-of-speech tagging, sentiment analysis,

and more. NLTK is widely used in research and education to work with textual data and develop NLP applications.

CHAPTER 4

RESULTS AND DISCUSSION

This chapter deals with analysis of results of 3 models GRU, LSTM and BiLSTM.

4.1 Experiment using GRU

GRU model with over 580k words, 10 epochs, 64 batch size, 0.3 dropout, Adam optimizer, 0.001 learning rate, sparse_categorical_crossentropy loss function gave 99.71 % F1-score and 99.69 % accuracy.

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_7 (Embedding)     (None, 303, 100)          4043400

 gru_2 (GRU)                 (None, 303, 64)           31872

 dropout_6 (Dropout)         (None, 303, 64)           0

 dense_16 (Dense)            (None, 303, 64)           4160

 dense_17 (Dense)            (None, 303, 64)           4160

 time_distributed_6 (TimeDis  (None, 303, 103)         6695
 tributed)

=================================================================
Total params: 4,090,287
Trainable params: 4,090,287
Non-trainable params: 0
_________________________________________________________________
```

Fig. 4.1: Summary of GRU Model

Figure 4.1 shows that GRU model has one embedding layer, one dropout layer, two dense layer and one time distributed layer. There is no non-trainable parameter.

```
Epoch 1/10
317/317 [==============================] - 141s 430ms/step - loss: 0.5275 - accuracy: 0.9377 - val_loss: 0.1558 - val_accuracy:
0.9616
Epoch 2/10
317/317 [==============================] - 142s 448ms/step - loss: 0.0828 - accuracy: 0.9783 - val_loss: 0.0320 - val_accuracy:
0.9926
Epoch 3/10
317/317 [==============================] - 140s 441ms/step - loss: 0.0211 - accuracy: 0.9948 - val_loss: 0.0196 - val_accuracy:
0.9956
Epoch 4/10
317/317 [==============================] - 140s 440ms/step - loss: 0.0104 - accuracy: 0.9974 - val_loss: 0.0174 - val_accuracy:
0.9963
Epoch 5/10
317/317 [==============================] - 140s 442ms/step - loss: 0.0073 - accuracy: 0.9982 - val_loss: 0.0176 - val_accuracy:
0.9963
Epoch 6/10
317/317 [==============================] - 140s 443ms/step - loss: 0.0060 - accuracy: 0.9985 - val_loss: 0.0165 - val_accuracy:
0.9965
Epoch 7/10
317/317 [==============================] - 139s 439ms/step - loss: 0.0051 - accuracy: 0.9987 - val_loss: 0.0164 - val_accuracy:
0.9966
Epoch 8/10
317/317 [==============================] - 139s 438ms/step - loss: 0.0046 - accuracy: 0.9988 - val_loss: 0.0159 - val_accuracy:
0.9968
Epoch 9/10
317/317 [==============================] - 151s 476ms/step - loss: 0.0042 - accuracy: 0.9989 - val_loss: 0.0164 - val_accuracy:
0.9968
Epoch 10/10
317/317 [==============================] - 161s 508ms/step - loss: 0.0039 - accuracy: 0.9990 - val_loss: 0.0152 - val_accuracy:
0.9971
```

Fig. 4.2: History of training GRU model

Figure 4.2 shows that training loss and validation loss are decreasing over time. Training accuracy and validation accuracy are increasing over time. It took almost 143 seconds to train each epoch which is faster. At the first and second epoch, training accuracy and loss are more. Because GRU has fewer parameters and starts with initial parameter, which lead to faster convergence and allow to fit the training data more quickly in the early epochs.

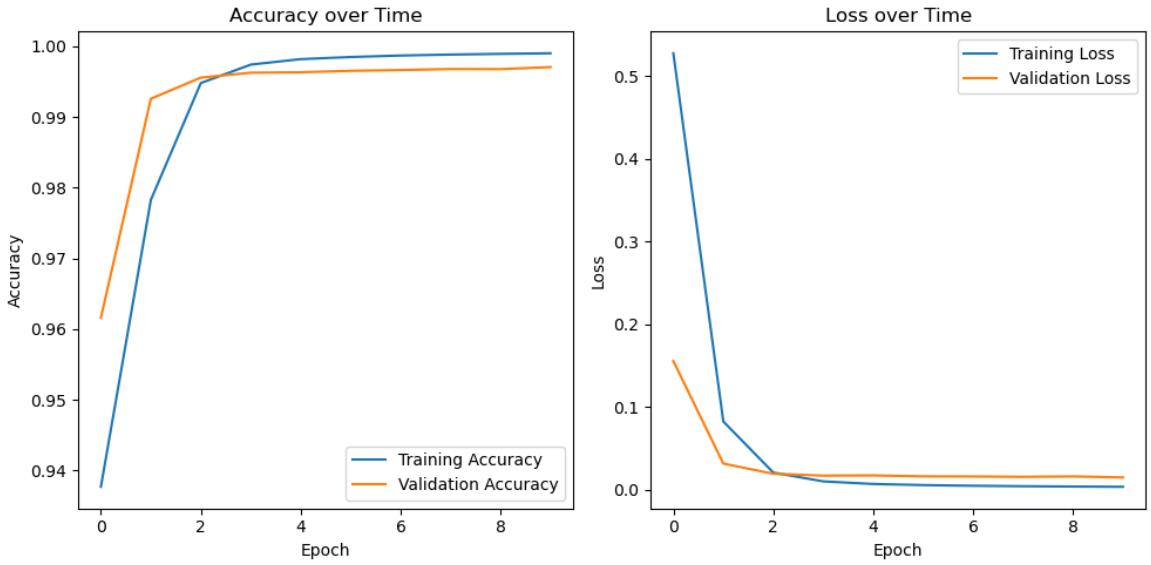


Fig. 4.3: Accuracy and Loss over time of GRU model

Figure 4.3 shows that the training and validation accuracy are increasing over epochs. Training accuracy is higher than the validation accuracy. After the 3rd and 4th epochs validation accuracy and training accuracy increases slowly respectively. Training and

validation loss are decreasing over epochs. That means it minimizes the error in the model. Training loss is better than the validation loss.

```
# Evaluate the model
loss, accuracy = gru_model.evaluate(sentences_test, pos_tags_test, verbose=1)
print("Loss: {0},\nAccuracy: {1}".format(loss, accuracy))
```

```
136/136 [==============================] - 13s 97ms/step - loss: 0.0154 - accuracy: 0.9969
Loss: 0.015375316143035889,
Accuracy: 0.9969483017921448
```

Fig. 4.4: Evaluate GRU model using test dataset

Figure 4.4 shows that the model resulted, the testing loss: 0.0154, testing accuracy: 99.69%.

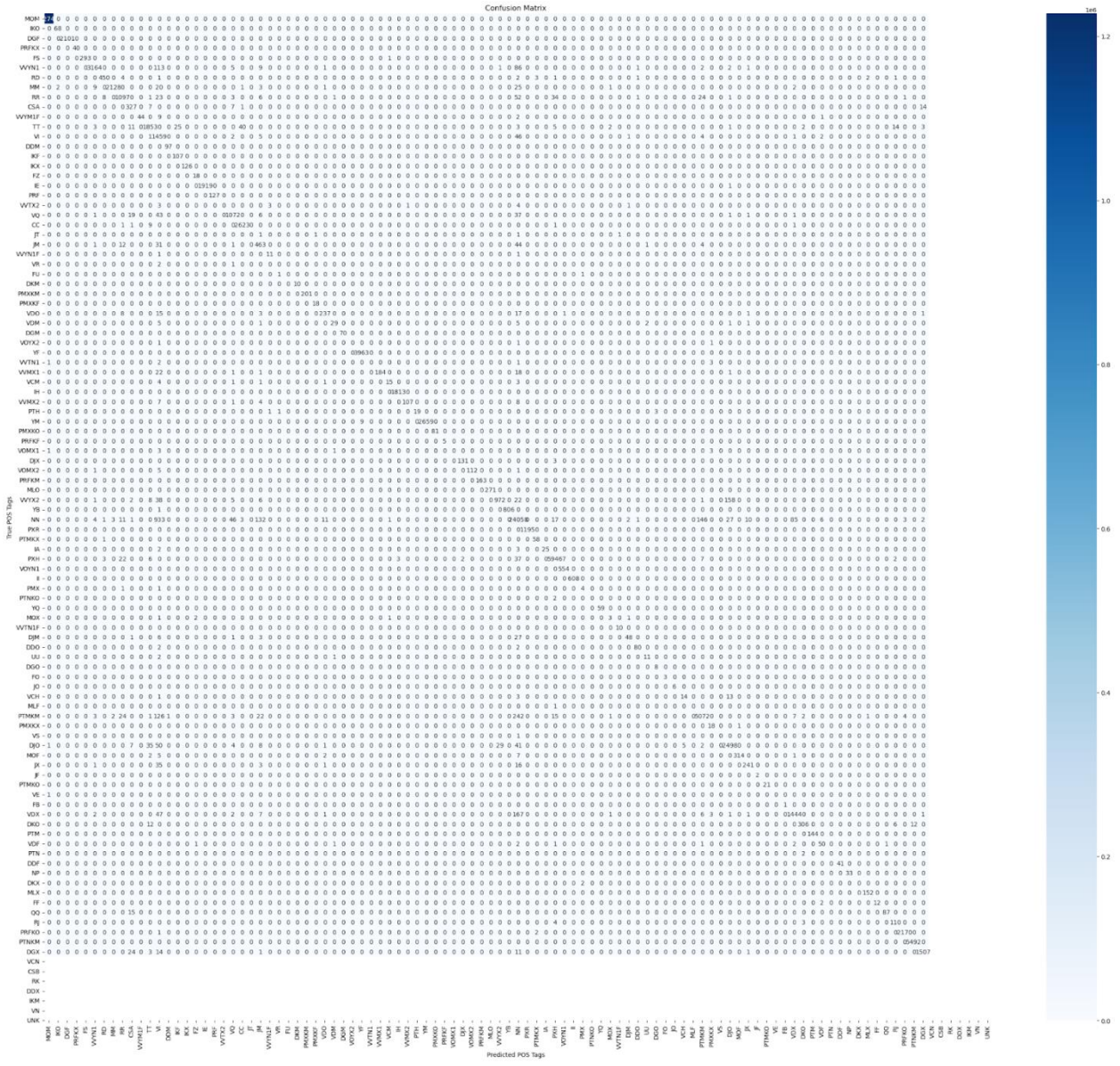


Fig. 4.5: Confusion matrix for GRU model

Figure 4.5 shows that confusion matrix for GRU model.

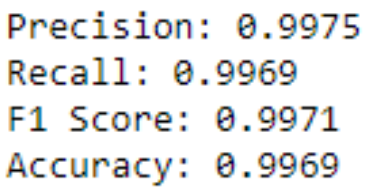


Fig. 4.6: Precision, Recall, F1 score and Accuracy

Figure 4.6 shows that Precision, Recall, F1 Score and Accuracy of GRU model, all are above 99%.

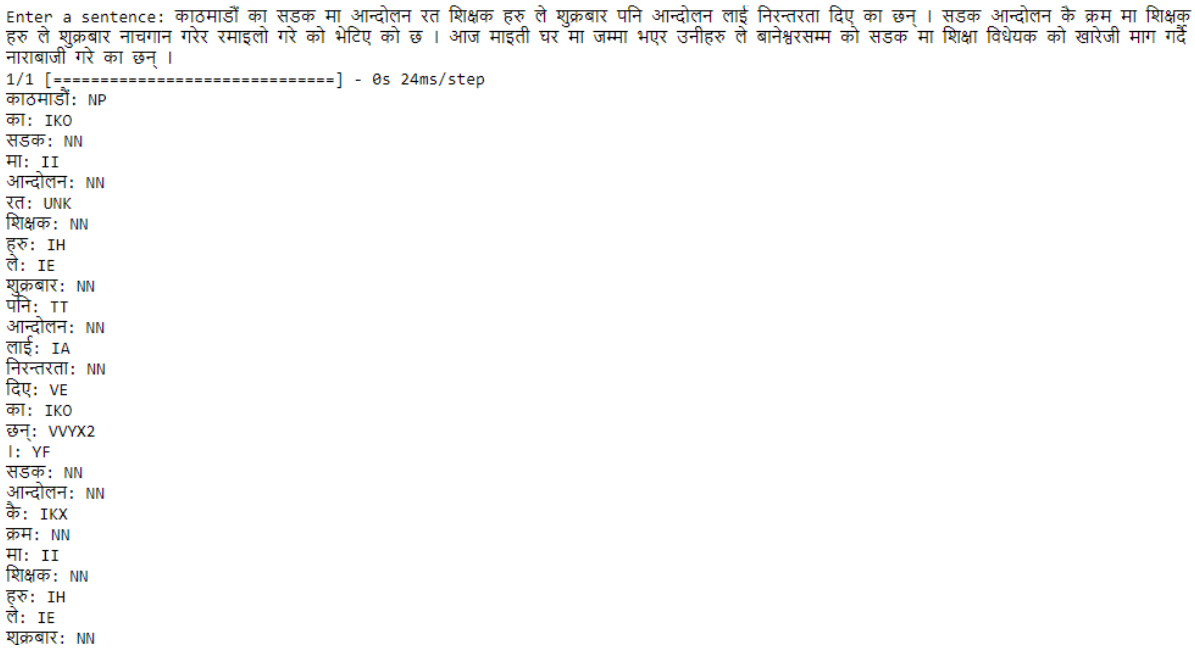


Fig. 4.7: Output of user input sentences using GRU

Figure 4.7 shows that while testing the GRU model with user input, the model has tagged most of the word with correct tag. It also deal with unknown words with UNK tag.

4.2 Experiment using LSTM

LSTM model with over 580k words, 10 epochs, 64 batch size, 0.3 dropout, Adam optimizer, 0.001 learning rate, sparse_categorical_crossentropy loss function gave 99.65 % F1-score and 99.64 % accuracy.

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 303, 100)          4043400

 lstm_4 (LSTM)               (None, 303, 64)           42240

 dropout_3 (Dropout)         (None, 303, 64)           0

 dense_7 (Dense)             (None, 303, 64)           4160

 dense_8 (Dense)             (None, 303, 64)           4160

 time_distributed_3 (TimeDis  (None, 303, 103)         6695
 tributed)

=================================================================
Total params: 4,100,655
Trainable params: 4,100,655
Non-trainable params: 0
_________________________________________________________________
```

Fig. 4.8: Summary of LSTM Model

Figure 4.8 shows that LSTM model has one embedding layer, one dropout layer, two dense layer and one time distributed layer. There are all trainable parameters.

```
Epoch 1/10
317/317 [==============================] - 148s 456ms/step - loss: 0.4629 - accuracy: 0.9411 - val_loss: 0.1550 - val_accuracy:
0.9609
Epoch 2/10
317/317 [==============================] - 146s 460ms/step - loss: 0.1194 - accuracy: 0.9665 - val_loss: 0.0689 - val_accuracy:
0.9843
Epoch 3/10
317/317 [==============================] - 148s 468ms/step - loss: 0.0564 - accuracy: 0.9848 - val_loss: 0.0355 - val_accuracy:
0.9923
Epoch 4/10
317/317 [==============================] - 150s 473ms/step - loss: 0.0316 - accuracy: 0.9915 - val_loss: 0.0259 - val_accuracy:
0.9944
Epoch 5/10
317/317 [==============================] - 149s 469ms/step - loss: 0.0207 - accuracy: 0.9947 - val_loss: 0.0228 - val_accuracy:
0.9954
Epoch 6/10
317/317 [==============================] - 149s 470ms/step - loss: 0.0154 - accuracy: 0.9962 - val_loss: 0.0210 - val_accuracy:
0.9968
Epoch 7/10
317/317 [==============================] - 147s 465ms/step - loss: 0.0124 - accuracy: 0.9970 - val_loss: 0.0213 - val_accuracy:
0.9963
Epoch 8/10
317/317 [==============================] - 148s 466ms/step - loss: 0.0104 - accuracy: 0.9975 - val_loss: 0.0208 - val_accuracy:
0.9968
Epoch 9/10
317/317 [==============================] - 148s 468ms/step - loss: 0.0090 - accuracy: 0.9979 - val_loss: 0.0206 - val_accuracy:
0.9969
Epoch 10/10
317/317 [==============================] - 147s 464ms/step - loss: 0.0080 - accuracy: 0.9981 - val_loss: 0.0215 - val_accuracy:
0.9966
```

Fig. 4.9: History of training LSTM model

Figure 4.9 shows that training loss and validation loss are decreasing over time. Training Accuracy and validation accuracy are increasing over time. The maximum testing accuracy is 99.81%. It took almost 149 seconds to train each epoch. Which is closer to GRU model.

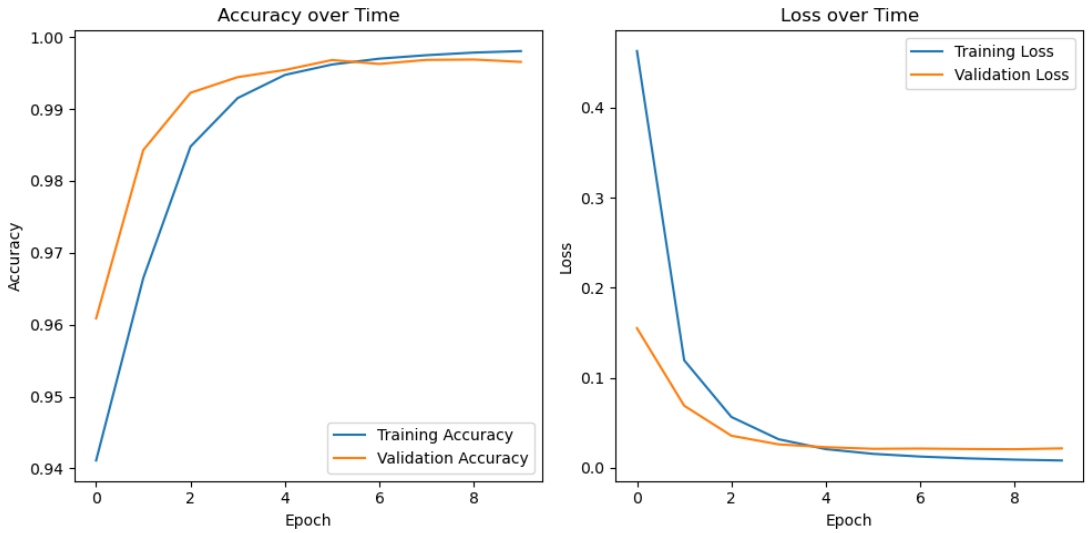


Fig. 4.10: Accuracy and Loss over time of LSTM model

Figure 4.10 shows that the training and validation accuracy are increasing over epochs. Training accuracy is higher than the validation accuracy. After the 3rd and 5th epochs validation accuracy and training accuracy increases slowly respectively. Training and validation loss are decreasing over epochs. That means it minimizes the error in the model. Training loss is better than the validation loss.

```
# Evaluate the model
loss, accuracy = model.evaluate(sentences_test, pos_tags_test, verbose=1)
print("Loss: {0},\nAccuracy: {1}".format(loss, accuracy))
```

```
136/136 [==============================] - 26s 193ms/step - loss: 0.0220 - accuracy: 0.9964
Loss: 0.022001389414072037,
Accuracy: 0.9964016675949097
```

Fig. 4.11: Evaluate LSTM model using test dataset

Figure 4.11 shows that the model resulted, the testing loss: 0.0220, testing accuracy: 99.64%.

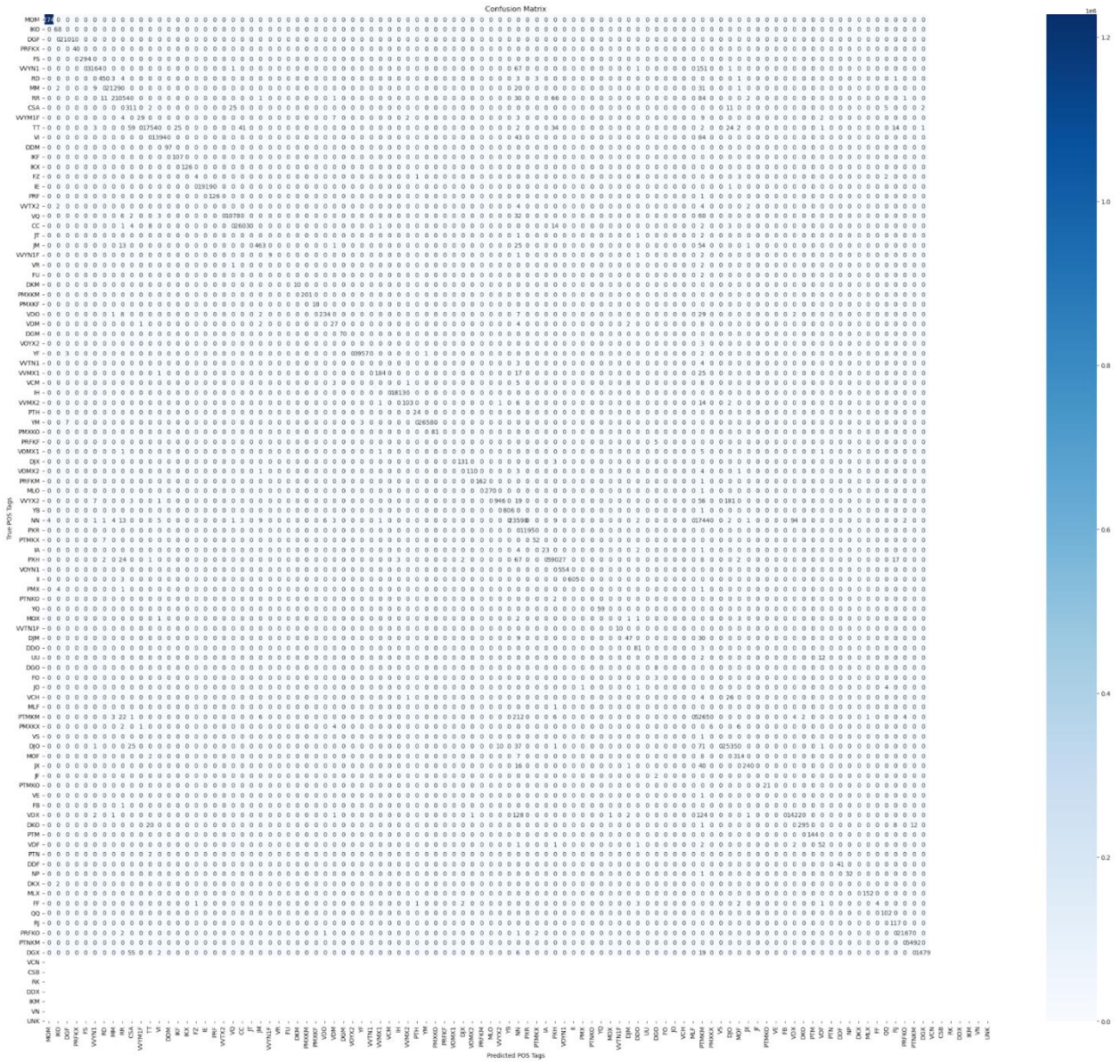


Fig. 4.12: Confusion matrix for LSTM model

Figure 4.12 shows that Confusion matrix for LSTM model.

```
Precision: 0.9969
Recall: 0.9964
F1 Score: 0.9965
Accuracy: 0.9964
```

Fig. 4.13: Precision, Recall, F1 score and Accuracy

Figure 4.13 shows that Precision, Recall, F1 Score and Accuracy of LSTM model, all are above 99%.

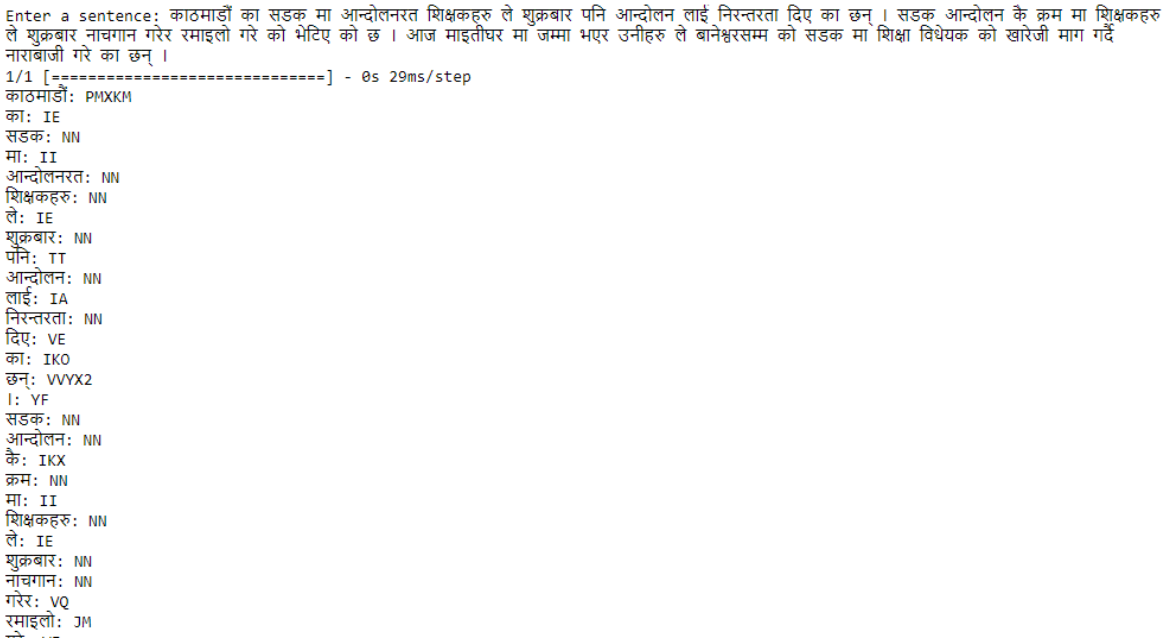


Fig. 4.14: Output of user input sentences using LSTM model

Figure 4.14 shows that while testing the LSTM model with user input, the model has tagged most of the word with correct tag. It also deal with unknown words with UNK tag.

4.3 Experiment using BiLSTM

BiLSTM model with over 580k words, 10 epochs, 64 batch size, 0.3 dropout, Adam optimizer, 0.001 learning rate, sparse_categorical_crossentropy loss function gave 99.76 % F1-score and 99.77 % accuracy.

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_11 (Embedding)    (None, 303, 100)          4043400

 bidirectional_3 (Bidirectio  (None, 303, 128)         84480
 nal)

 dropout_10 (Dropout)        (None, 303, 128)          0

 dense_28 (Dense)            (None, 303, 64)           8256

 dense_29 (Dense)            (None, 303, 64)           4160

 time_distributed_10 (TimeDi  (None, 303, 103)         6695
 stributed)

=================================================================
Total params: 4,146,991
Trainable params: 4,146,991
Non-trainable params: 0
_________________________________________________________________
```

Fig. 4.15: Summary of BiLSTM Model

Figure 4.15 shows that BiLSTM model has one embedding layer, one dropout layer, two dense layer and one time distributed layer. There are all trainable parameters.

```
Epoch 1/10
317/317 [==============================] - 347s 1s/step - loss: 0.3920 - accuracy: 0.9463 - val_loss: 0.1403 - val_accuracy: 0.
9644
Epoch 2/10
317/317 [==============================] - 329s 1s/step - loss: 0.0988 - accuracy: 0.9735 - val_loss: 0.0487 - val_accuracy: 0.
9886
Epoch 3/10
317/317 [==============================] - 379s 1s/step - loss: 0.0363 - accuracy: 0.9911 - val_loss: 0.0240 - val_accuracy: 0.
9953
Epoch 4/10
317/317 [==============================] - 363s 1s/step - loss: 0.0169 - accuracy: 0.9961 - val_loss: 0.0181 - val_accuracy: 0.
9966
Epoch 5/10
317/317 [==============================] - 372s 1s/step - loss: 0.0101 - accuracy: 0.9978 - val_loss: 0.0167 - val_accuracy: 0.
9970
Epoch 6/10
317/317 [==============================] - 369s 1s/step - loss: 0.0077 - accuracy: 0.9983 - val_loss: 0.0169 - val_accuracy: 0.
9973
Epoch 7/10
317/317 [==============================] - 331s 1s/step - loss: 0.0063 - accuracy: 0.9986 - val_loss: 0.0168 - val_accuracy: 0.
9975
Epoch 8/10
317/317 [==============================] - 374s 1s/step - loss: 0.0056 - accuracy: 0.9988 - val_loss: 0.0171 - val_accuracy: 0.
9976
Epoch 9/10
317/317 [==============================] - 351s 1s/step - loss: 0.0050 - accuracy: 0.9989 - val_loss: 0.0175 - val_accuracy: 0.
9976
Epoch 10/10
317/317 [==============================] - 359s 1s/step - loss: 0.0046 - accuracy: 0.9991 - val_loss: 0.0172 - val_accuracy: 0.
9977
```

Fig. 4.16: History of training BiLSTM model

Figure 4.16 shows that training loss and validation loss are decreasing over time. Training Accuracy and validation accuracy are increasing over time. The maximum testing accuracy is 99.91%. It took almost 360 seconds to train each epoch. Which is more than two times greater than GRU and LSTM model.

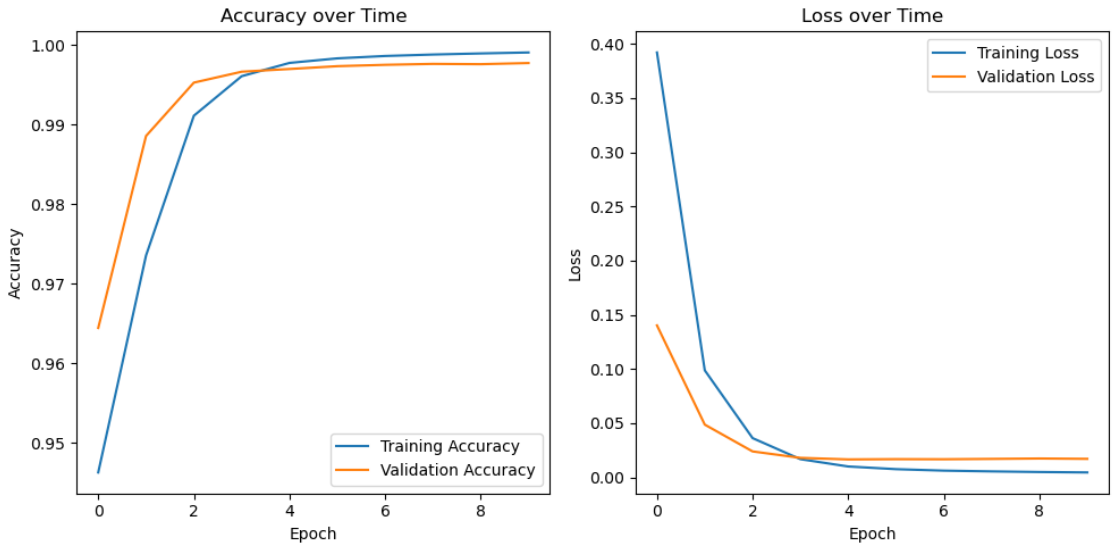


Fig. 4.17: Accuracy and Loss over time of BiLSTM model

Figure 4.17 shows that the training and validation accuracy are increasing over epochs. Training accuracy is higher than the validation accuracy. After the 3rd and 4th epochs validation accuracy and training accuracy increases slowly respectively. Training and validation loss are decreasing over epochs. That means it minimizes the error in the model. Training loss is better than the validation loss.

```
# Evaluate the model
loss, accuracy = bilstm_model.evaluate(sentences_test, pos_tags_test, verbose=1)
print("Loss: {0},\nAccuracy: {1}".format(loss, accuracy))
```

```
136/136 [==============================] - 38s 281ms/step - loss: 0.0176 - accuracy: 0.9977
Loss: 0.017569588497281075,
Accuracy: 0.9976545572280884
```

Fig. 4.18: Evaluate BiLSTM model using test dataset

Figure 4.18 shows that the model resulted, the testing loss: 0.0176, testing accuracy: 99.77%.

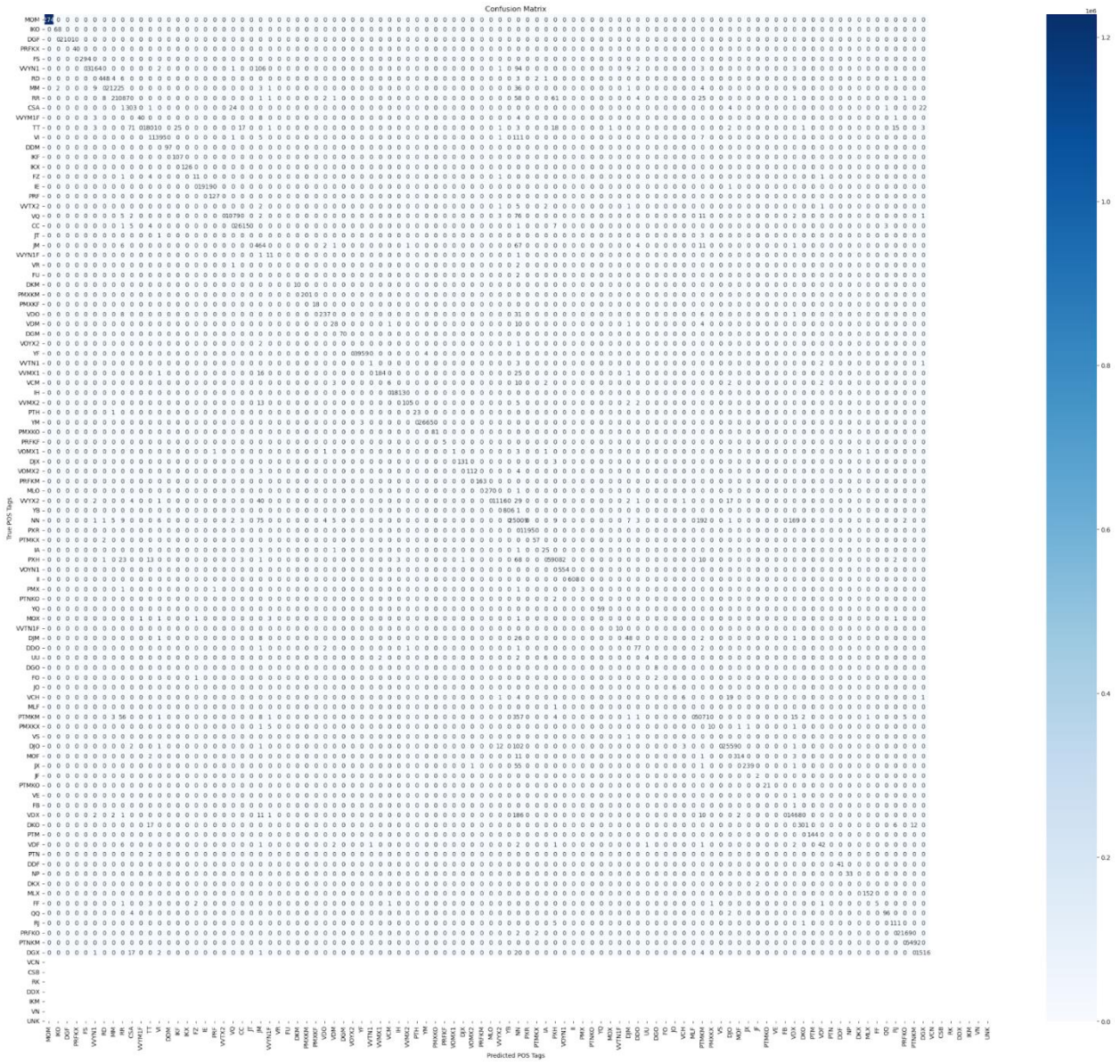


Fig. 4.19: Confusion matrix for BiLSTM model

Figure 4.19 shows that confusion matrix for BiLSTM model.

```
Precision: 0.9977
Recall: 0.9977
F1 Score: 0.9976
Accuracy: 0.9977
```

Fig. 4.20: Precision, Recall, F1 score and Accuracy

Figure 4.20 shows that Precision, Recall, F1 Score and Accuracy of GRU model, all are above 99%.

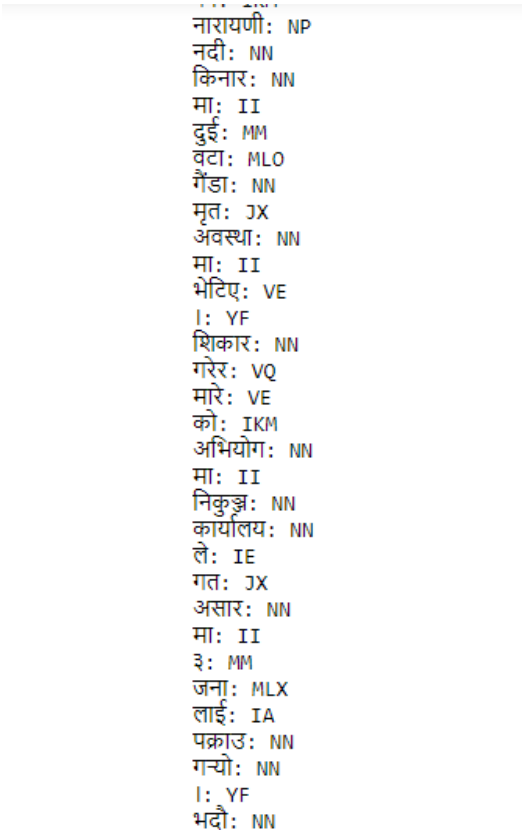


Fig. 4.21: Output of user input sentences using BiLSTM model

Figure 4.21 shows that while testing the BiLSTM model with user input, the model has tagged most of the word with correct tag. It also deal with unknown words with UNK tag.

4.4 Comparing Results between GRU, LSTM, BiLSTM

There are two tables which show the performance of the three models.

Models	**Loss**	**Accuracy**	**Val_loss**	**Val_accuracy**	**Evaluation Loss**	**Evaluation Accuracy**
LSTM	0.0080	0.9981	0.0215	0.9966	0.0220	0.9964
GRU	**0.0039**	0.9990	**0.0152**	0.9971	**0.0153**	0.9969

BiLSTM	0.0046	**0.9991**	0.0172	**0.9977**	0.0175	**0.9976**

Table 4.1: *Comparison between 3 Deep Learning Models based on Loss, Accuracy, Val_loss, Val_accuracy*

Models	**Precision**	**Recall**	**F1 Score**	**Accuracy**
LSTM	0.9969	0.9964	0.9965	0.9964
GRU	0.9975	0.9969	0.9971	0.9969
BiLSTM	**0.9977**	**0.9977**	**0.9976**	**0.9977**

Table 4.2: *Comparison between 3 Deep Learning Models based on Precision, Recall, F1 score and Accuracy*

Models	**Loss**	**Accurcy**
LSTM	0.0220	0.9964
GRU	**0.0153**	0.9969
BiLSTM	0.0175	**0.9976**

Table 4.3: *Comparison between 3 Deep Learning Models based on Testing Dataset*

According to above tables, BiLSTM has achieved the maximum performance scores as compare to other models. It is also new state-of-the-art F1 score of 99.76% for fine-grained Nepali POS tagging. GRU and LSTM also have good score. There is very small difference in the scores of models. Because this research used deep learning model with genism's Word2Vec word embedding. Every model have one embedding layer, one dropout layer, two dense layer and one time distributed layer. Every model trained with 10 epochs, 64 batch size, 0.3 dropout, 0.001

learning rate, Adam optimizer, and sparse categorical cross entropy loss function. Talking about existing research, Shrestha and Dhakal (2021) achieved a state-of-the-art F1 score of 98.51% using BiLSTM-CRF with Bare Embedding, emphasizing the effectiveness of their chosen model at that time. They also used 0.01 learning rate, 0.4 dropout, Adam optimizer and categorical cross-entropy as a loss function. There are only few things that help to make more f1 score. First is we use genism's Word2Vec word embedding instead of Bare Embedding, emphasizing. And next is our model have 2 dense layer, 0.001 learning rate and 0.3 dropout. Because choice of embedding and optimization parameters decide the best model.

CHAPTER 5

CONCLUSION AND TASK TO BE COMPLETED

4.1 Conclusion

In this work, we applied three deep learning models for fine-grain POS tagging for the Nepali language. We showed that deep learning models could capture fine-grained morphological features like gender, person, number, and honorifics that are encoded within words in highly inflectional languages like Nepali. POS Tagged Nepali National Corpus (NNC) with over 14 million words and NELRALEC tag sets with 112 fine-grained POS Tags was used for modelling the POS tagger. BiLSTM model achieved new state-of-the-art F1 score of 99.76% for fine-grained Nepali POS tagging with genism's Word2Vec word embedding. It also showed that deep learning models could capture fine-grained morphological features like gender, person, number, and honorifics that are encoded within words in highly inflectional languages like Nepali with a large enough dataset. This research also showed that if we extract xml files like we mentioned in above preprocessing topic, it reduces model training time.

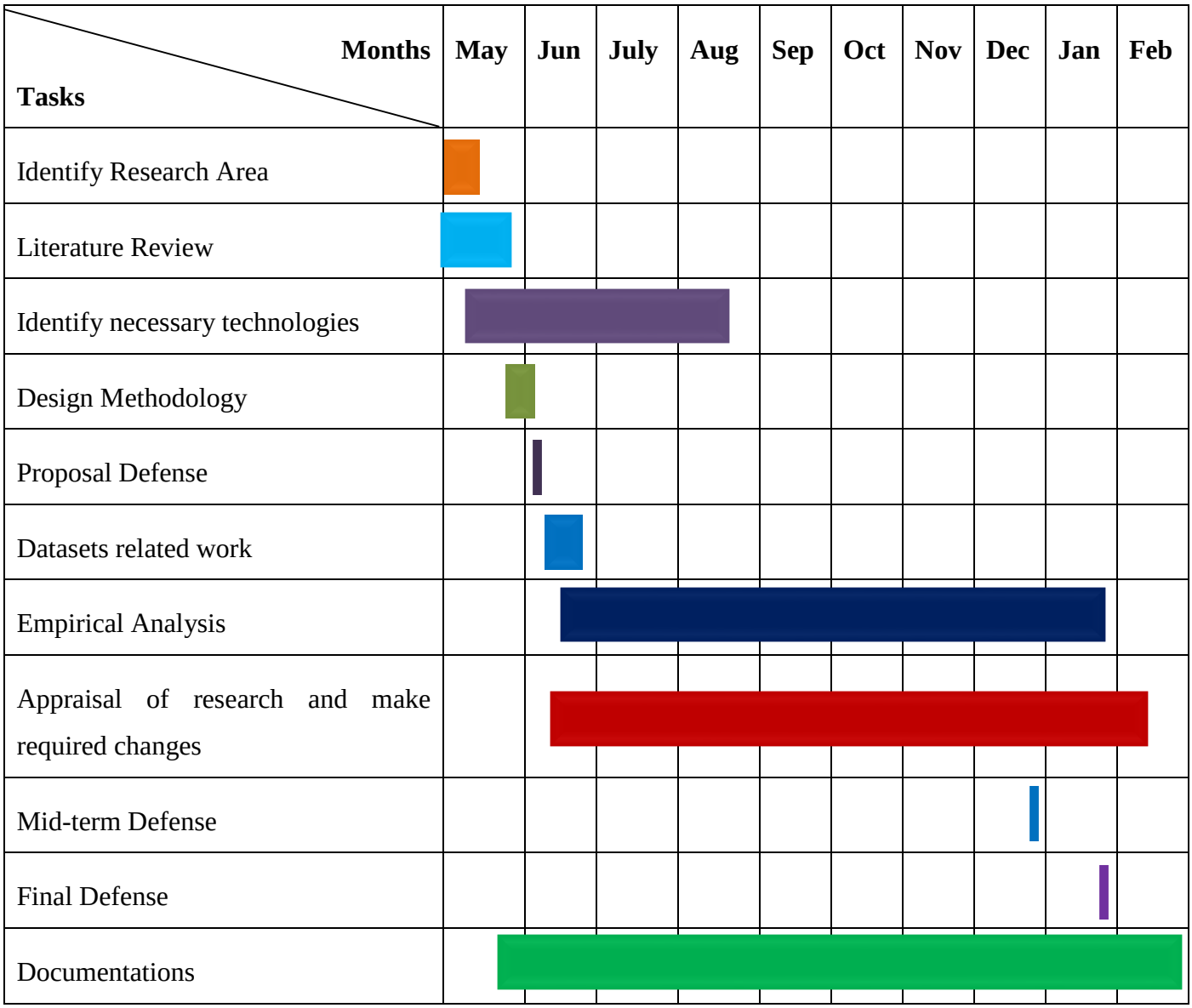
4.2 Task to be Completed

The following tasks need to be carried for completion of work:

a. Train GRU, LSTM and BiLSTM models on the Nepali Monolingual Written Corpus.
b. Fine tune pre-trained mBERT model by using the Nepali Monolingual Written Corpus. Because mBERT have been trained on large and diverse datasets, has already learned a lot of linguistic knowledge from vast text data, can save a lot of training time and resources compared to training a model from scratch. Additionally, mBERT often have convenient libraries and resources for fine-tuning, such as the Hugging Face Transformers library in Python. Transformer-based models like BERT gives state-of-the-art in many natural language processing tasks, including POS tagging. BERT capture contextual information much better than models like LSTM or BiLSTM which improve the accuracy of POS tagging because it takes into account the entire sentence's context when assigning POS tags to words.
c. Analyze the performance of the algorithms.

APPENDIX A: GANTT CHART

The project will be five months long and the works are divided accordingly. The planned schedule for the project are illustrated in Gantt Chart below:



REFERENCES

[1] I. Shrestha, S. S. Dhakal, "Fine-Grained Part-Of-Speech Tagging in Nepali Text," *Procedia Computer Science,* vol. 189, PP 300-311, 2021

[2] A. Pradhan, A. Yajnik, "Probabilistic and Neural Network Based POS Tagging of Ambiguous Nepali Text: A Comparative Study," *ISEEIE 2021: 2021 International Symposium on Electrical, Electronics and Information Engineering,* PP. 249–253, feb 2021

[3] A. Yajnik, "Part Of Speech Tagging Using Statistical Approach for Nepali Text," *International Scholarly and Scientific Research & Innovation,* vol. 11, no. 1, 2017

[4] S. Sayami, T. B. Shahi and S. Shakya, "Nepali POS Tagging Using Deep Learning Approaches," *NU. International Journal of Science*, Dec 2019.

[5] A. Paul, B. S. Purkayastha, S. Sakar, "Hidden Markov Model Based Part of Speech Tagging for Nepali Language," *International Symposium on Advanced Computing and Communication (lSACC),* Sep 2015.

[6] A. Yajnik, "General Regression Neural Network Based POS Tagging for Nepali Text," 4th *International Conference on Natural Language*, pp. 35–40, 2018.

[7] Tej Bahadur Shahi, Tank Nath Dhamala, Bikash Balami, "Support Vector Machines Based Part Of Speech Tagging for Nepali Text," *International Journal of Computer Applications* (0975 – 8887), Volume 70– No.24, May 2013.

[8] A. Yajnik, "ANN Based POS Tagging for Nepali Text," *International Journal on Natural Language Computing (IJNLC),* Vol.7, No.3, June 2018.

[9] Greeshma Prabha, P. V. Jyothsna, K. K. Shahina, B. Premjith, K. P. Soman, "A Deep Learning Approach for Part-Of-Speech Tagging in Nepali Language," *International Conference on Advances in Computing, Communications and Informatics (ICACCI),* Sep 2018.

[10] M. Jayaweera, N. G. J Dias, “Hidden Markov Model Based Part of Speech Tagger for Sinhala Language,” *International Journal on Natural Language Computing (IJNLC),* Vol. 3, No.3, June 2014.

[11] P. Sinha, N. M. Veyie, B. S. Purkayastha, “Enhancing The Performance of Part Of Speech Tagging of Nepali Language Through Hybrid Approach,” *International Journal of Emerging Technology and Advanced Engineering,* Vol. 5, Issue 5, May 2015.

[12] “Gated Recurrent Unit,” commons.wikimedia.org. https://commons.wikimedia.org/wiki/File:Gated_Recurrent_Unit,_type_1.svg (Accessed Sep 23, 2023)

[13] B. Bohara, Raymond I. Fernandez, V. Gollapudi, Xingpeng Li, “Short-Term Aggregated Residential Load Forecasting using BiLSTM and CNN-BiLSTM,” *International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*, 2022.