Enhanced Tamil Morphological Analysis through Deep Learning and Embedding-Based Similarity Techniques

V. Nareashkaan

2025

Enhanced Tamil Morphological Analysis through Deep Learning and Embedding-Based Similarity Techniques

Vijayaragavan Nareashkaan
Index No: 20001193

Supervisor: Dr. HND Thilini
B.Sc. (Col), Ph.D. (Col)

May 2025

Submitted in partial fulfillment of the requirements of the B.Sc. (Honours) in Computer Science Final Year Project

Declaration

I certify that this dissertation does not incorporate, without acknowledgement, any material previously submitted for a degree or diploma in any university and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. I also hereby give consent for my dissertation, if accepted, be made available for photocopying and for interlibrary loans, and for the title and abstract to be made available to outside organizations.

Candidate Name: V. Nareashkaan

Signature of the Candidate Date: 25/04/2025

This is to certify that this dissertation is based on the work of Mr. V. Nareashkaan under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Principle Supervisor's Name: Dr. HND Thilini

Signature of the Principle Supervisor Date: 25/04/2025

Co-Supervisor's Name: Dr. BHR Pushpananda

Signature of the Co-Supervisor Date: 25/04/2025

Abstract

Morphological analysis is vital in NLP, especially for morphologically rich languages like Tamil, which pose challenges due to complex inflectional and derivational forms. Traditional rule-based methods struggle with scalability and unseen words, while deep learning remains underexplored for Tamil.

This research proposes a hybrid approach combining deep learning for lemma prediction and an embedding-based similarity method for grammatical feature prediction. Various architectures—including Recurrent Neural Network (RNN), Long Short term memory (LSTM) and Gradient recurrent unit (GRU)—are evaluated for lemma prediction, while FastText embeddings enable effective feature transfer for unseen words, addressing the out-of-vocabulary problem.

The model is trained on curated word-lemma pairs and grammatical annotations, demonstrating high accuracy and generalization. This work offers a scalable, low-resource-friendly solution for Tamil morphological analysis and contributes to advancing Tamil NLP.

Acknowledgment

I would like to express my sincere gratitude to my research supervisor Dr. HND Thilini (senior lecturer at the University of Colombo School) for her valuable guidance, support and encouragement throughout this research work. I would also like to extend gratitude to my research Co-Supervisor Dr B.H.R. Pushpananda (senior lecturer at the University of Colombo School) and my research advisor. Mr C. Liyanage (Linguist at LTRL) for their valuable suggestions and guidance during this research work.

I'm happy to thank Dr R.Weerasinghe for the valuable suggestions during this research work. Finally, I would like to thank my beloved parents for being my strength always. It gives me great pleasure to acknowledge the assistance and contributions of everyone who has helped me complete my research successfully. This project would not have been possible without the guidance and support which was provided by them.

Contents

List of Figures

List of Tables

List of Acronyms

AI Artificial Intelligence

NLP Natural Language Processing

MAG Morphological Analyzer Generator

RNN Recurrent Neural Network

LSTM Long Short-Term Memory

GRE Gated Recurrent Units

CRF Conditional Random Fields

MSD Morphosyntactic descriptors

SVM Support Vector Machine

OOV Out-of-Vocabulary

Chapter 1

Introduction

1.1 Background to the Research

A morphological analyzer is a tool which breaks up a single word into its basic components identifying its root, prefixes, suffixes and infixes. This is a very essential process in NLP since it is used in understanding and generating language.Morphological analysis is crucial for various applications such as text mining, search engine, personal assistants, social media, and e-commerce platforms, to help users quickly find the necessary information. This concept is mostly used in query-based systems, morphological analysis enhances information retrieval by reducing the index size and grouping terms related to the same root word.

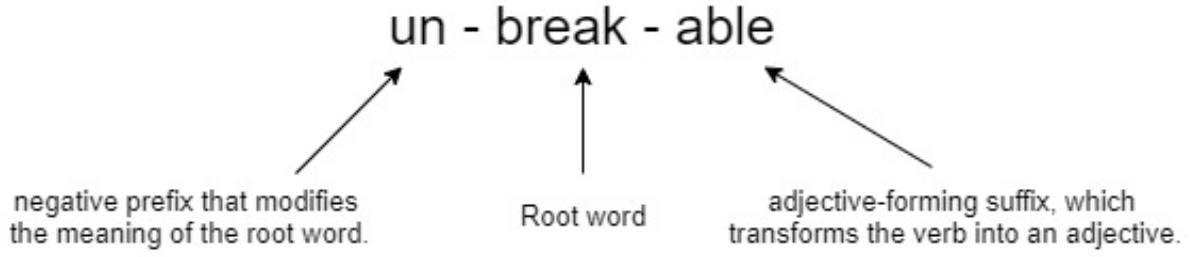

Figure 1.1: Example for morphological analysis in the English language

In this research, the focus will be on morphological tagging and lemmatization, a key area within morphological analysis that involves associating a given word form with its base form (lemma) and detailed morphosyntactic features. As defined by Nicolai and Kondrak (2017), morphological analysis often refers to the tasks of lemmatization and tagging, collectively describing a word's syntactic properties such as part-of-speech, tense,

number, or case. This study aims to develop an approach that will tag each word with these morphosyntactic descriptors, also referred to as MSD tags, while normalizing each word to its lemma or base form.

Morphological tagging is closely aligned with part-of-speech tagging but adds layers of morphosyntactic detail Labeau et al. (2015), Conforti et al. (2018), making it crucial for applications in NLP where precise linguistic information is essential. The research will therefore emphasize these tasks to enhance the understanding and computational representation of Tamil morphology, focusing specifically on accurately predicting lemmatized forms and detailed morphological tags for a given word.

Input word	Lemma	Grammatical features
இருக்கிறாள்	இரு	Gender=Fem|Number=Sing|Person=3|Tense=Pres

Figure 1.2: Example where multiple suffixes attach to the root verb

Lemmatization, stemming, and morphological tagging are some of the essential methods for understanding the behavior of words in a particular language. Morphological analysis requires examining how words are constructed from morphemes, a process that varies widely between languages. In this research, the focus is on Tamil morphological analysis, specifically identifying morphemes and their associated grammatical features. For this purpose, a combined approach is proposed. It utilizes a list of Tamil words annotated with grammatical features, and by leveraging word embedding techniques, the model aims to assign grammatical features to unseen words based on their embedding similarity to words in the annotated list. This allows the transfer of grammatical information from similar words, addressing OOV issues for novel terms. While this method uses word embedding similarity, deep learning techniques may also be explored to enhance accuracy and model performance. By employing a flexible, data-driven approach, this research aims to develop a robust and scalable solution for Tamil morphological analysis.

1.2 Problem Statement

Despite numerous advancements in developing morphological analyzers for Tamil—such as rule-based methods, finite-state transducers, and machine learning techniques like

SVM—there remains a notable lack of research using deep learning approaches. Traditional methods often require extensive, manually crafted rules, which struggle to adapt to Tamil's complex agglutinative morphology, including its extensive affixation and diverse verb conjugation patterns. These methods inherently face challenges in handling Tamil's morphological intricacies and in achieving scalability for real-world applications.

In contrast, other Dravidian languages like Malayalam, as well as morphologically complex languages such as Sanskrit, Nepali, Sinhala, and Finnish, have benefited from deep learning models that automatically learn and apply complex morphological rules, significantly outperforming traditional methods. The success of deep learning for Malayalam highlights the potential for similar improvements in Tamil morphological analysis. However, no substantial work has yet explored this approach for Tamil, representing a critical research gap.

This study seeks to address this gap by not only applying deep learning techniques but also incorporating a similarity-based method that leverages word embeddings. By transferring grammatical features from similar words, this approach could mitigate the limitations of traditional data collection and rule-based methods, reducing dependency on extensive annotated datasets. Such a hybrid approach could provide a more flexible, data-efficient solution, advancing Tamil morphological analysis and contributing to the field of Tamil NLP by addressing current scalability and adaptability challenges.

In this research, two types of approaches will be explored for Tamil morphological analysis: a deep learning approach and a similarity-based approach using word embeddings. The deep learning approach leverages advanced models to learn complex morphological patterns automatically, a method that has shown strong results in languages like Malayalam and Sanskrit. The similarity-based approach, on the other hand, utilizes word embeddings to find grammatical features for unseen words by identifying the closest match in a list of annotated words and transferring its features.

By combining these two approaches, this research aims to address the limitations of relying solely on extensive annotated data. The deep learning model will enable complex morphological analysis, while the similarity-based method will improve the model's ability to generalize to OOV words. Together, these approaches provide a flexible and potentially more efficient solution for Tamil morphological analysis, with applications in practical NLP tasks.

1.3 Research Aim, Questions and Objectives

1.3.1 Research Aim

Aim of this research is to develop an effective and scalable approach for Tamil morphological analysis, focusing specifically on tasks such as morphological tagging and lemmatization. This study seeks to combine deep learning techniques with a similarity-based method using word embeddings to handle Tamil's complex morphology, including agglutinative structures and OOV words. The ultimate goal is to create a robust system that accurately predicts grammatical features and base forms, improving both computational efficiency and adaptability for real-world NLP applications in Tamil.

1.4 Research Questions

1. **What approaches are most effective for Tamil morphological analysis in low-resource settings?**

   Tamil's morphological richness and lack of annotated corpora make analysis challenging. This study proposes a hybrid method combining deep learning for lemma prediction and embedding-based similarity for grammatical feature prediction. It reduces manual annotation by leveraging gold-standard datasets, offering an efficient and scalable solution.

2. **How effectively can word embeddings predict morphological features for unseen Tamil words?**

   FastText embeddings, which use subword-level representations, enable accurate grammatical feature prediction for OOV words by capturing morphological patterns and similarities. This helps generalize beyond the training corpus and reduces dependency on large annotated datasets.

3. **What are the challenges in building a Tamil morphological corpus, and how can they be addressed?**

   Tamil's inflectional complexity and context-dependent word forms make annotation labor-intensive and inconsistent. This research mitigates the issue by combining manually curated lemma data with pre-annotated datasets, improving coverage

while limiting annotation effort.

1.4.0.1 Research Objectives

1. Develop and implement a deep learning model for Tamil morphological analysis that can accurately perform lemmatization, addressing the language's agglutinative structure and morphological complexity.

2. Design and test a similarity-based approach using word embeddings to predict grammatical features for unseen words, enabling effective handling of OOV words in Tamil.

3. Evaluate the effectiveness of combining deep learning with similarity-based methods for improved accuracy, generalization, and adaptability of Tamil morphological analysis.

4. Optimize data collection and annotation processes by minimizing reliance on extensive annotated datasets, while still achieving high performance through hybrid modeling approaches.

1.5 Significance of the Project

The significance of this research lies in enhancing natural language processing (NLP) capabilities for Tamil, a morphologically complex and under-resourced language. Existing Tamil morphological analyzers primarily rely on rule-based methods, which demand extensive manual effort and often lack scalability, especially for Tamil's agglutinative structure. This study introduces a hybrid approach that combines deep learning and similarity-based methods, providing a more adaptable and efficient solution. By developing a model that can accurately predict grammatical features and lemmas for Tamil words, even for OOV terms, this research addresses crucial limitations of traditional methods and contributes a valuable tool for a wide range of NLP applications, such as machine translation, sentiment analysis, and text-to-speech systems.

Furthermore, this research has broader implications beyond Tamil. Applying advanced deep learning techniques to a language with rich morphological complexity, this

study may offer insights into developing similar tools for other low-resource, morphologically complex languages, especially within the Dravidian language family.

1.6 Research Approach And Methodology

1. **Data Collection and Annotation:** The dataset for this research contains following category:

    - **Word-Lemma Pairing:** Collection of Tamil words (nouns, verbs, adjectives, adverbs, pronouns, verbal nouns) paired with their corresponding lemmas to support lemmatization tasks.

2. **Model Development and Similarity-Based Algorithm:**

    - **Deep Learning Models:** Character-level neural network models, including Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), and Gated Recurrent Unit (GRU), will be developed. These models will be trained on the annotated corpus to learn morphological patterns, capturing both syntactic and semantic structures inherent in Tamil.
    - **Similarity-Based Approach:** In addition to deep learning, a similarity-based approach using word embeddings will be implemented. This approach involves calculating word embedding similarity to find grammatical features for unseen words by identifying the closest annotated words and transferring their features.

3. **Experimentation and Evaluation:**

    - A series of experiments will be conducted across different deep learning architectures (RNN, LSTM, GRU) to assess model effectiveness.
    - Evaluation metrics will include per-tag accuracy for grammatical features and exact string match for lemmatization. These metrics will provide insights into the accuracy and reliability of the models across various morphological tagging and lemmatization tasks

1.7 Outline of the Dissertation

Chapter 1 – Introduction

This chapter presents information about the background to the research problem and the research questions which this study aims to find answers to. And also this chapter presents the project scope and the aim of the study

Chapter 2 – Literature review

This chapter presents information about similar works related to this study and a review of the existing literature. This chapter gives an idea about the way those studies related to this research. And also this chapter gives the knowledge about theories and concepts that have used in this study

Chapter 3 – Design

This chapter provides the design of the research observation and consists of information about the applications examined, workloads simulated and the tools and technologies used. And also this chapter describes the flow of this research.

Chapter 4 – Implementation

This chapter describes the implementation of the proposed hybrid model for Tamil morphological analysis. Additionally, it presents the algorithms used for morphological prediction, including the deep learning techniques for lemma prediction and the similarity-based method for grammatical feature assignment.

Chapter 5 – Results and evaluation

This chapter presents the results that were obtained through the experiments conducted with the evaluation and explanations of those results. And also, this chapter provides explanations for each and every result.

Chapter 6 – Conclusion

This dissertation concludes with this chapter by presenting the final outcomes of the research and a summary of contributions made through this research.

1.8 Scope Including Delimitation

- **Development of a Deep Learning Model for Tamil Morphological Analysis:**
  - Implementation of character-level neural network models (RNN, LSTM, GRU, Bi-RNN, Bi-LSTM, Bi-GRU) to perform morphological tagging and lemmatization.
- **Similarity-Based Approach:**
  - Design and application of a similarity-based algorithm using word embeddings to predict grammatical features for unseen words.
- **Data Collection and Annotation:**
  - Collection of Tamil words (nouns, verbs, adjectives, adverbs, pronouns, verbal nouns) with lemmas for training and evaluation.
- **Experimental Analysis:**
  - Testing of different deep learning architectures and configurations and evaluation of models using metrics like per-tag accuracy for grammatical features and exact string match for lemmas.

1.8.1 Out Scope

- **Context-Aware Morphological Analysis:**
  - This research does not consider context-dependent morphological analysis, focusing only on individual word forms without surrounding text context.
- **Word Segmentation:**

  - The study excludes word segmentation tasks and focuses solely on morphological tagging and lemmatization for complete word forms.

- **Development of Morphological Analyzers for Other Languages:**
  - This research is limited to Tamil and does not extend to developing or evaluating models for other languages.

- **Human-Centered Usability Testing:**
  - Usability testing or evaluation from a user experience perspective is not included, as this research is focused on model development and accuracy.

- **Real-Time or Production-Ready Systems:**
  - The study does not aim to develop real-time or deployable systems, as the focus is on research and experimental analysis rather than practical deployment.

Chapter 2

Literature Review

To understand possible gaps within the previous works of morphological analyzers a preliminary review has been conducted.

2.1 What is morphological analysis?

The process of examining a word's internal structure and how it is put together by joining various morphemes is known as morphological analysis. Word forms may be distinguished and many grammatical variations can be produced by using the morphological analysis of the words, which provides essential information about the word forms. To analyze the development of a word, it must be broken down into its individual morphemes. A sandhi splitter can be used to complete this work. Sandhi splitters identify morpheme borders and morphological changes brought about by sandhi, which allows them to separate the constituent morphemes inside a word.

Tamil classical morphology is incredibly complex. Like the other Dravidian languages, it is an agglutinative language. The lexical roots of classical Tamil words are followed by one or more affixes. The smallest meaningful units are referred to as morphemes and consist of lexical roots and affixes. As a result, morphemes are joined together in a sequence to form classical Tamil words. Lexical morphemes, also known as lexical roots, usually appear first in a structure. Other grammatical or functional morphemes may or may not follow this. Because it inflects to person, gender, and number markers as well as mixes with auxiliaries that express aspect, mood, causality, attitude, and other things in verbs, the morphological structure of classical Tamil is rather complex. One of the

main tasks in NLP is computational morphology. In this field, many methods have been put forth, particularly for European languages where morphological analysis is an easy undertaking. Nevertheless, morphological analysis and sandhi splitting are challenging tasks in agglutinative languages like Tamil, Malayalam, Telugu like Dravidian languages. On this subject, not much research has been done.

2.1.1 Evolution of Tamil morphological analysis

Initially rule based approaches were used for this work. Later the research has moved to the machine learning direction. The Anusaraka group started the process of building a morphological analyzer for the Tamil language. Ganesan created a morphological analyzer to examine the CIIL corpus in Tamil. This follows phonological and morphophonemic guidelines and considers Tamil's morphotactic restrictions while constructing. According to Menon et al. (2009) the second work has reported in 2010 which is a Rule-Based Approach to AMRITA Morph Analyzer and Generator for Tamil (2010): AMAG, a finite state transducer, was used by Drs. A.G. Menon, S. Saravanan, R. Loganathan, and K. Soman of Amrita University in Coimbatore to create a rule-based morphological analyzer and generator for Tamil. Lexicon and orthographic principles from a two level morphological system underpin the system's functionality. About 3,000 verbs, 50,000 nouns, and a somewhat smaller amount of adjectives make up the system. When compared to the current Tamil morph analyzer and generator, known as ATCHARAM, the suggested AMAG performed better.

The work ATCHARAM deals with the inflections and derivations those are not the same for all the nouns and verbs. The biggest challenge is the grouping of nouns and verbs in such a way that the members of the same group have similar inflections and derivations. Otherwise one has to make rules for each noun and verb, which is not feasible. This system, as mentioned earlier, works on rules and these rules are capable of solving this clumsiness. The system design involves building an exhaustive lexicon for noun, verb and other categories. The performance is directly related to this exhaustiveness. It is a laborious task.

According to Anand Kumar et al. (2010a) in 2010 An Ad Hoc Morphological Generator for Tamil The improvised database for morphological analysis and creation implemented on Apertium is the subject of this study, which is proposed by Parameswari K. of

CALTS, University of Hyderabad. The Word and Paradigm based database, along with the Finite State Transducers method for one-pass analysis and generation, are used by the makeshift MAG. The system's speed and accuracy are assessed and contrasted with those of the other Tamil morphological analyzers on the market, which were created at Anna University's AU-KBC Research Center and CALTS. The outcome of the experiment demonstrated that the suggested MAG outperforms the alternative.

A morphological analyzer for Tamil language based sequence labeling approach has been introduced which uses the first machine learning approach by Anand Kumar M, Dhanalakshmi V, Soman K.P and Rajendran S of AMRITA Vishwa Vidyapeetham, Coimbatore according to Anand Kumar et al. (2010b) The suggested work reframes the morphological analyzer problem as a classification problem, which is then solved through the application of machine learning techniques. This is a corpus-based method that uses support vector machine techniques for both testing and training. There are 130,000 verb words and 70,000 noun terms in the training corpus, respectively. Using 30,000 nouns and 40,000 verbs from the Amrita POS Tagged corpus, the method is tested. When the system's performance was compared to that of other systems created using the same corpus, it was found that the SVM-based technique performed better. They reported 92.95% accuracy in the analysis. The approach used the SVM for morphological analysis of Tamil words.

Lushanthan et al. (2014) have proposed a morphological analyser and generator for Tamil, which has been implemented using XFST. The authors have used transliteration to handle the Tamil script, given that the current version of XFST has rendering issues, although it supports Unicode internally. The authors have considered 2,000 noun and 96 verb stems as part of their analysis and generation. They have tested the system using their own data set consisting of 3,500 nouns and 500 verbs with a success rate of 78%. However, the data sets and XFST rules have not been made available.

There has also been an attempt to develop a Morphological analyser for Tamil using a support vector machine Mokanarangan et al. (2016). Although the writers have reported an accuracy of 98.73%, their system is not available, and it is not clear what data sets or analyses have been used for training and testing, as there is no data available in the paper, or online.

Sarveswaran et al. (2021) This paper presents an open source and extendable Morpho-

logical Analyser Generator (MAG) for Tamil named ThamizhiMorph which is considered as the recent development in Tamil morphological analysis. This paper describes how ThamizhiMorph is designed using a Finite-State Transducer (FST) and implemented using Foma. Although no benchmark resources exist for an evaluation of NLP applications developed for Tamil, they designed two evaluation experiments to test the coverage and accuracy of ThamizhiMorph. The results are very good in that identifed errors are either due to out-of-vocabulary items or derivational formations that have not as yet been implemented.

In spite of this topics greater significance in the computational processing in Tamil, no research has been reported using deep learning approaches.dataset gathered from the UD Tamil treebank

2.2 Recent advancement in morphological analysis

Recent advancements in deep learning have begun to address the limitations of rule-based and traditional machine learning approaches in Tamil morphological analysis. In other Dravidian languages, deep learning models have demonstrated promising results. The work reported in Malayalam language which introduces a deep learning approach for learning the rules for identifying the morphemes automatically and segmenting them from the original word. Then individual morphemes can be further analyzed to identify the grammatical structure of the word. Three different systems were developed for this analysis using RNN, LSTM and GRU and obtained accuracies 98.08%, 97.88% and 98.16% respectively according to Premjith et al. (2018).

Similarly, a bidirectional LSTM-based morphological analyzer for Gujarati has achieved high accuracy by predicting root words and morphological features without relying on language-specific rules according to Baxi and Bhatt. This approach also experimented with monolithic and individual label representations, which significantly improved the baseline accuracy. Figure 2.1 shows the architecture of the system for grammatical feature prediction task used in the study.These successes suggest that deep learning can offer more robust morphological analyzers for Tamil.

In addition, morphologically-guided embeddings represent a novel way to encode linguistic relationships. Traditional word embeddings typically capture semantic similarity

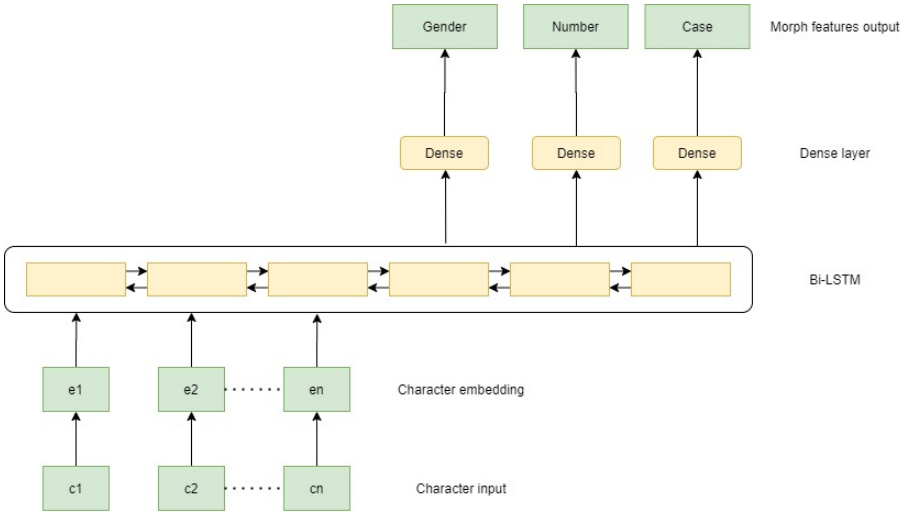

Figure 2.1: System architecture for the grammatical feature prediction task

but often ignore morphological structure. Cotterell and Schütze (2019) addressed this by using morphologically annotated data to guide embeddings, ensuring that morphologically similar words remain close in the embedding space. This approach is especially advantageous for morphologically rich languages like Tamil, where capturing internal word structure enhances the quality of language models.

To further address out-of-vocabulary (OOV) issues common in morphologically rich languages, FastText embeddings utilize character-level n-grams, representing words by a combination of subword units. This method not only handles OOV words more effectively but also enables the model to generalize morphological variations based on subword structures. Given Tamil's extensive morphological inflection, integrating subword embeddings like FastText could be particularly beneficial.

These advances in deep learning and morphology-aware embeddings offer new pathways for Tamil morphological analysis, enabling future models to achieve greater accuracy and better handle the complexity of Tamil's agglutinative morphology. By leveraging these modern techniques, Tamil NLP can move closer to the high-performance standards already observed in other languages, enhancing applications in text processing, machine translation, and beyond.

Tamil and Malayalam, both Dravidian languages, share an agglutinative structure and complex morphological systems, though they differ in aspects such as verb conjugation complexity, noun inflection patterns, and the influence of Sanskrit. While the deep learning approaches that have been effective for Malayalam can be adapted for Tamil,

significant adjustments are necessary. These include tailoring preprocessing steps for the Tamil script, annotating a substantial Tamil corpus, and incorporating Tamil-specific morphological rules. Additionally, a similarity-based approach using word embeddings could enhance the model's ability to handle OOV words by leveraging similarities to existing annotated words. By combining deep learning techniques with embedding-based similarity methods, a robust and efficient morphological analyzer for Tamil can be effectively developed, addressing both the challenges of resource limitations and Tamil's unique morphological characteristics.

Time period	Dominating approaches	Notable work
1980 - 1990	Two level morphology, Stemmer based approaches	Koskenniemi (1996), Karttunen and Wittenburg (1983), Lun (1983), Porter (1980)
1990 - 2000	Two level morphology, Finite state transducer	Oflazer (1994), Beesley (1998), Koskenniemi (1996)
2000 - 2010	Suffix Stripping approach, Paradigm based approach, Unsupervised morphology	Eryiğit and Adalı (2004), Batsuren et al. (2021), Jena et al. (2011), Goldsmith (2001), Hammarström and Borin (2011)
2010 - 2018	Supervised machine learning, Statistical methods	Kumar et al. (2009), Abeera et al. (2012), Malladi and Mannem (2013), Mokanarangan et al. (2016)
2018 onwards	Deep learning based methods	Cotterell and Heigold (2017), Adelani et al. (2021), Malaviya et al. (2018), Kondratyuk (2019)

Table 2.1: Overview of the approaches used during different time periods for developing computational morphology tools for various languages (Baxi and Bhatt (2024))

2.3 Theories and Concepts

RNN, LSTM, and GRU have been the methods employed in the majority of studies for morphological analysis in other languages. The majority of the investigations have produced outcomes that are extremely high. Taking those results into account, it suggests that deep learning techniques might also work better for Tamil language morphological analysis. Recurrent neural networks rely on previous factors in the sequence to determine their outcomes.

An artificial neural network class called RNN is frequently applied to sequential data. A directed graph with a data sequence is represented by the connections between nodes. As a result, it can exhibit considerable temporal behavior. Feedforward neural networks are the source of these networks. RNN generates future calculations and outcomes based on previous information.

Figure 2.2 illustrates how recurrent neural networks leverage prior information to

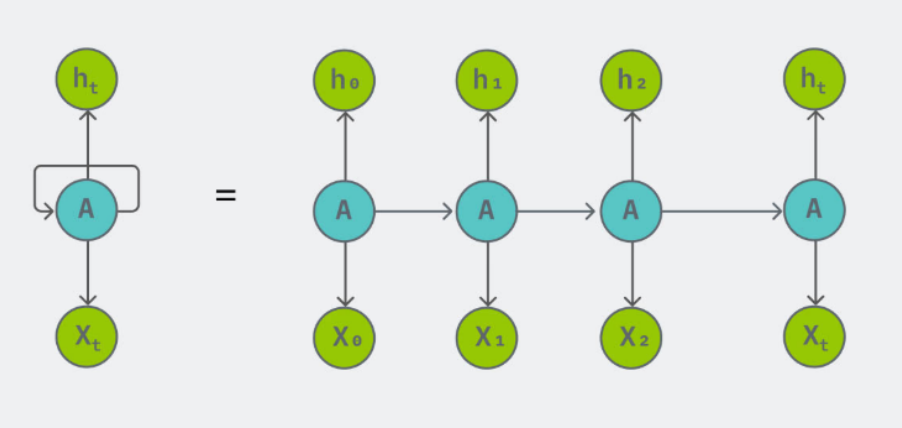

Figure 2.2: Structure of RNN

produce future outcomes. For supervised learning, RNNs are employed. This makes it simple to determine the relationship between the input data set and the labels that go with it.

Hochreiter proposed the LSTM (Hochreiter and Schmidhuber (1997)). An input gate, an output gate, a forget gate, and a cell make up an LSTM. The values are occasionally remembered by the cell. Information entering and leaving the cell is managed by three gates. The vanishing gradient issue that might arise during the training of conventional RNNs has been addressed by LSTMs. What information should be discarded or kept private is determined by the forget gate. The gate takes into account both information

about the current input and information from the prior concealed state for this. The cell state is updated from the input gate. The output gate determines the next hidden state, which holds the data from the previous input. Predictions are also made using the hidden state.

GRU (Cho et al. (2014)), proposed by Cho and B. Merrienboer. This is a gating mechanism in recurrent neural networks. This is also similar to the LSTMs and it has fewer parameters than the LSTMs. GRUs use a hidden state to transfer information and it has two gates (reset gate and update gate). Update gate acts similar to the forget in input gate of LSTMs. It decides what is the information that needs to keep and the information that can throw away. The reset gate is a gate that decides how much pass information to forget. Some studies have shown that GRU is faster than LSTM for processing the same dataset. Usually, researchers have used both of LSTMs and GRUs to determine which one works better for their use case. In this research also, all these architectures will be considered and the most suitable one will be selected. A graphical representation of LSTM and GRU has shown in Figure 2.3.

Bidirectional RNNs are also taken into consideration in this study. Rather than

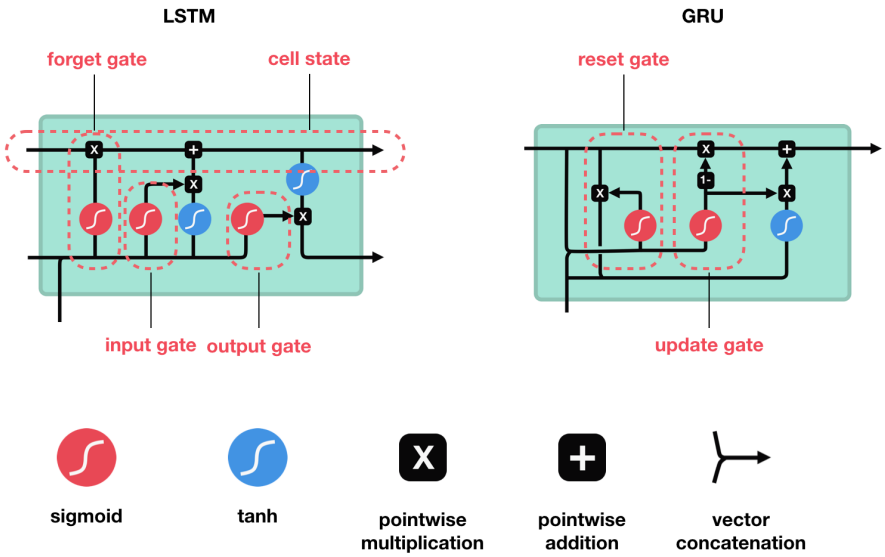

Figure 2.3: LSTM and GRU(Kostas (2023))

training a single model, these models train two models. The input sequence is read by the first model, and the opposite sequence is read by the second model. Bidirectional models have been employed in research such as Prasad et al. (2019), Prabha et al. (2018), Gilmullin et al. (2018), and Kim and Kim (2020).

In terms of deep learning in morphological analysis, the majority of comparable languages have similarly demonstrated excellent performance. A study on this subject is preferable because no research has utilized deep learning for morphological analysis in the Sinhala language.

As in the work Sarveswaran et al. (2021) has stated there is a problem in where few errors that did occur were mostly because of: Out-of-vocabulary (OOV) items – words that weren't in the system's dictionary, Unseen derivational forms – new or rare word formations not covered by the existing rules. Word representation is fundamental for NLP. Recently, continuous word-embeddings have gained traction as a general-purpose representation framework. While such embeddings have proven themselves useful, they typically treat words holistically, ignoring their internal structure. For morphologically impoverished languages, i.e., languages with a low morpheme-per-word ratio such as English, this is often not a problem. However, for the processing of morphologically-rich languages exploiting word internal structure is necessary.

Word-embeddings are typically trained to produce representations that capture linguistic similarity. The general idea is that words that are close in the embedding space should be close in meaning. A key issue, however, is that meaning is a multifaceted concept and thus there are multiple axes, along which two words can be similar. For example, ice and cold are topically related, ice and fire are syntactically related as they are both nouns, and ice and icy are morphologically related as they are both derived from the same root.

As discussed above the comparison between words can be used to morphologically analyses the Tamil words. But the problem we face is OOV words where every word couldn't be represented in a unique way. The work Bojanowski et al. (2017) introduces a solution for this. Continuous word representations, trained on large unlabeled corpora are useful for many natural language processing tasks. Popular models that learn such representations ignore the morphology of words, by assigning a distinct vector to each word. This is a limitation, especially for languages with large vocabularies and many rare words. In this paper, they propose a new approach based on the skipgram model, where each word is represented as a bag of character n-grams. A vector representation is associated to each character n-gram; words being represented as the sum of these representations. This method is fast, allowing to train models on large corpora quickly and

allows us to compute word representations for words that did not appear in the training data.

The embedding based method could be employed for the grammatical feature prediction since the embeddings of the words themselves represents morphological features where it could open a novel room for exploration in Morphological analysis using embedding based similarity method.

2.4 Summary

In terms of these deep learning architectures, RNN is a stage of input networks created when node connections quickly build a desired graph. RNNs may process long equivalent inputs by using their internal state, or memory. The gradient descent and momentum techniques are used to construct LSTMs. The GRU and LSTM are comparable as well. However, compared to LSTM, it contains fewer parameters. Employing this deep learning technique could help in the process of lemmatization task for Tamil words.

A similarity-based approach using word embeddings could enhance the model's ability to handle OOV words by leveraging similarities to existing annotated words. By combining deep learning techniques with embedding-based similarity methods, a robust and efficient morphological analyzer for Tamil can be effectively developed, addressing both the challenges of resource limitations and Tamil's unique morphological characteristics.

Chapter 3

Design

This research involves developing a morphological analyzer for Tamil using a combination of deep learning for lemma prediction and similarity-based approach with word embeddings for grammatical feature prediction. The approach is structured in the following steps in figure 3.1 where it will explained step by step in the latter sections.

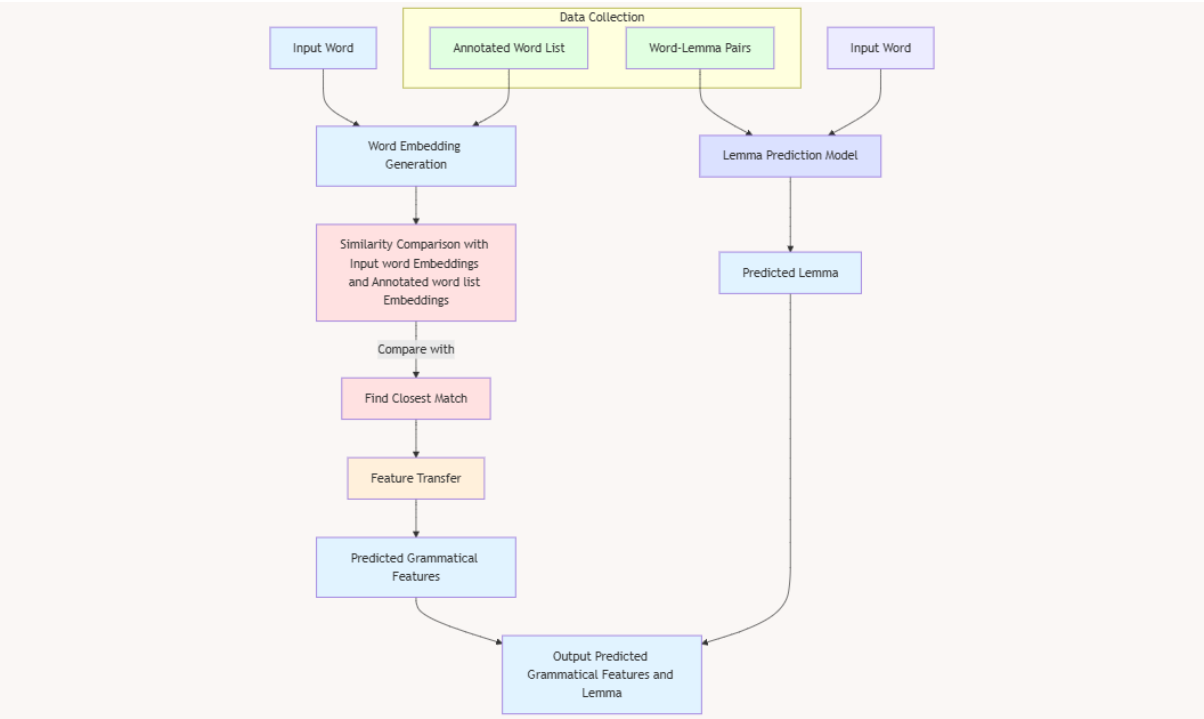

Figure 3.1: Research Design for Tamil Morphological Analysis Using Deep Learning and Similarity-Based Approaches

3.1 Define research problem

As mentioned in the section 1.2, initially, the problem is defined that is going to address in this research. We also came up with three major research questions to address in this research. Since this is an hybrid approach the research problem is splitted into two designs

1. Lemma prediction
2. Grammatical feature prediction

3.2 Lemma prediction

3.2.1 Data collection

The dataset is gathered from the UD Tamil treebank(Kengatharaiyer et al. (2020), Ramasamy and Žabokrtský (2012)) and simple verbs and nouns are collected from kaggle by K. Sarveswaran according to Sarveswaran. The words collected additionally after UD Tamil treebank are annotated manually. This dataset consists the Tamil words and their corresponding lemmas in Tamil script. This dataset contains 70,007 total number of words. Figure 3.2 shows some sample data in the Tamil script.

Considering this dataset, it contains different types of Tamil words. Considering

Word	Lemma
துரந்தவரை	துர
குனிவிக்கின்றர்	குனி
புகல்வவரை	புகல்
முயன்னவர்கள்	முயல்
தவிர்ந்தவனைக்	தவிர்
குழல்விக்கின்றவளை	குழல்
மிகுக்கவர்கள்	மிகு

Figure 3.2: Structure of the dataset (Tamil script)

the different part of speech it is possible to identify different types of root forms in this dataset (Nouns, Verbs, Adverbs, Adjectives, Adpositions, Pronouns, Determiners, Part, Propernouns etc). Table 3.1 shows the number of words available in the total dataset

with the part-of-speech types.

Even though the dataset contains a large number of words, some word forms are not

Values	**Number of Words**	**%**
Verb	48,112	68.72 %
Noun	20,787	29.69 %
Proper noun	505	0.72 %
Adjective	225	0.32 %
Adverb	111	0.15 %
Auxiliary	111	0.15 %
Pronoun	55	0.078 %
Adposition	53	0.075 %
Determiner	24	0.034 %
Part	24	0.034 %

Table 3.1: Different POS types with available number of words

covered in the dataset. In this study, we are considering to develop a deep learning model which is suitable to predict the lemma(root morpheme / base root) of any kind of word. To check this ability, need to find a set of words that is not available in this dataset. For that, a set of Tamil words extracted from UD Tamil tree bank considered as test data which are unseen since that is the dataset considered as the gold standard.

3.2.2 Construct a deep learning model

Morphological analysis in the Tamil language can be formulated as a sequence labeling problem, particularly for lemma prediction. Since a single word may have multiple morphemes, the model must learn to map the inflected form to its corresponding base form (lemma). Each input word is embedded into a fixed-length vector representation, as described in Chapter 4, allowing the model to process and predict lemmas efficiently.

For lemma prediction, a deep learning-based approach is used, inspired by prior research that has shown promising results in morphological analysis (Premjith et al. (2018), Prasad et al. (2019), Ekanayaka et al. (2023)). This research explores different neural architectures, including Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM), and Gated Recurrent Units (GRU), as well as bidirectional variations of these

models. Bidirectional architectures enhance the model's ability to capture morphological dependencies by processing the input sequence in both forward and backward directions. Previous studies (Silfverberg and Hulden (2018), Silfverberg and Tyers (2019)) have demonstrated varying performance across these architectures, making it essential to experiment with multiple models to determine the most effective approach for Tamil lemma prediction.

The deep learning model is trained using a manually annotated word-lemma dataset, ensuring that it learns robust morphological patterns. Initially, the same hyperparameters used in Premjith et al. (2018) are applied, followed by hyperparameter tuning to optimize performance. The effectiveness of the model is evaluated based on exact string match accuracy, ensuring that the predicted lemmas align with ground truth annotations. This approach aims to improve Tamil lemma prediction accuracy, contributing to the development of more scalable and data-efficient morphological analysis systems.

3.2.3 Train the dataset with deep learning model

Since this research explores multiple deep learning architectures for lemma prediction, the dataset is trained using different approaches to analyze the model's performance. From the total dataset, 80% is used for training, while the remaining 20% is reserved for testing. In the deep learning model, character-level Tamil words are used as input, following successful approaches from studies like Premjith et al. (2018), Prasad et al. (2019) and Ekanayaka et al. (2023). As this research employs a supervised learning approach, proper label representation is necessary. For this purpose, the dataset consists of word-lemma pairs, where each input word is mapped to its corresponding lemma label before being fed into the deep learning model.

To train the model, server named as "ANTPC" server is used (4 x GeForce RTX 2080 Ti 11GB of GDDR6 memory with 128GB RAM) to get more processing power. To monitor and improve model performance, learning curves are plotted for each experiment, helping identify overfitting or underfitting. If the model exhibits signs of overfitting, regularization techniques and hyperparameter tuning are applied to optimize training. The computational complexity of training increases with the number of unique word-lemma pairs, as each word is represented as a vector. Previous studies, such as Silfverberg and Hulden (2018), have addressed similar issues by refining label definitions to reduce com-

putational costs.

After training, the predicted lemmas are evaluated based on exact string match accuracy, ensuring that the deep learning model produces accurate lemma outputs for input words. This methodology ensures that the Tamil morphological analyzer achieves a balance between efficiency, scalability, and accuracy, making it suitable for real-world NLP applications. Table 3.2 presents examples of input words and corresponding lemma labels used in training.

Input	**Label**
['ம', 'ர', 'ி', 'ய', 'ர', 'ி','ர']	['ம', 'ர', 'ி', 'ய', 'ர', 'ி']

Table 3.2: Labels used for training

3.2.4 Hyper-parameter tuning

Building a deep learning model is an iterative process, starting with an initial architecture and refining it through multiple configurations to achieve optimal performance in terms of accuracy, computational efficiency, and training stability. This process, known as hyperparameter tuning, involves selecting the best values for various model parameters to improve learning efficiency and generalization.

In this research, hyperparameter tuning is applied to optimize the deep learning model for lemma prediction. Several key hyperparameters are considered, including the number of neurons, activation function, optimizer, batch size, embedding size, and number of epochs. To identify the most effective combination, an exhaustive search method is used, where a predefined set of possible values is assigned to each hyperparameter. All possible combinations of these values are systematically tested to determine which configuration yields the highest lemma prediction accuracy. Table 3.3 shows the values considered for all combinations.

For each experiment, a combination of hyperparameters is selected (e.g., ['relu', 'Adam', 32, 64, 64]), and the deep learning model is trained with these settings. The accuracy of lemma prediction is evaluated, and the best-performing combination is selected as the final hyperparameter configuration. This ensures that the deep learning model is well-optimized for Tamil lemma prediction, balancing both computational efficiency and accuracy for real-world applications.

Hyper-parameter	**Selected values**
Activation function	relu, sigmoid, softmax, tanh
Optimizer	Adam, SGD, RMSprop
Number of neurons	32, 64, 128, 256, 512
Batch size	32, 64, 128, 256, 512
Embedding size	32, 64, 128, 256, 512

Table 3.3: Hyper-parameter selection

3.2.5 Find best performing model

This step focuses on identifying the most suitable input data representation, deep learning architecture, and model configuration for lemma prediction in Tamil morphological analysis. To determine the best-performing model, multiple deep learning architectures are evaluated through systematic experiments, as outlined in Section 3.2.3.

Since lemma prediction is formulated as a sequence labeling task, various deep learning architectures are tested, including Simple RNN, LSTM, GRU, and their bidirectional variants. The performance of these architectures is compared, and the model that achieves the highest lemma prediction accuracy is selected as the most effective approach.

Additionally, to ensure optimal performance, hyperparameter tuning is conducted (as described in Section 3.2.4), refining key parameters such as number of neurons, activation function, optimizer, batch size, embedding size, and epochs. The best-performing model is identified based on exact string match accuracy, ensuring that the predicted lemmas closely align with ground truth annotations. The final model configuration is proposed as the most efficient deep learning solution for Tamil lemma prediction, balancing accuracy, computational efficiency, and generalization.

3.2.6 Provide most suitable deep learning algorithm

After completing the model evaluation and selection process, the most suitable deep learning algorithm for lemma prediction in Tamil morphological analysis is identified. This section presents a step-by-step breakdown of the method used to implement the best-performing deep learning model, ensuring clarity and reproducibility.

By leveraging the insights gained from experiments and hyperparameter tuning, the

proposed model is optimized for accuracy, efficiency, and generalization. This structured approach provides a well-defined methodology for future research and practical applications in Tamil NLP, serving as a foundation for further advancements in morphological analysis.

3.3 Grammatical Feature Prediction

3.3.1 Data Collection

The primary dataset used for grammatical feature prediction in this research is the Universal Dependencies (UD) Tamil Treebanks, specifically UD_Tamil-TTB (which is Ramasamy and Žabokrtský (2012)) and UD_Tamil-MWTT(which is Kengatharaiyer et al. (2020)). These treebanks serve as the only publicly available gold-standard dataset for Tamil, providing pre-annotated morphological and syntactic information. Since the objective of this component is to predict grammatical features for unseen words, using an already annotated dataset ensures a reliable benchmark for training and evaluation.

While the UD Tamil Treebanks cover a wide range of part-of-speech (POS) categories, this research focuses on Nouns, Verbs, Proper Nouns, Pronouns, and Auxiliary Verbs. The analysis had to be limited to these categories because other parts of speech, such as Adjectives (ADJ), Numerals (NUM), Determiners (DET), Coordinating Conjunctions (CCONJ), Punctuation (PUNCT), and Adverbs (ADV), have little or no morphological tags with grammatical features. This lack of comprehensive morphological annotations and the variability in data quality restrict the study to only a subset of word categories, ensuring a more reliable and meaningful analysis.

The dataset contains 5290 unique words, covering train, test, dev dataset of UD_Tamil-TTB and test dataset of UD_Tamil-MWTT (Only one file available which is named as test dataset). Table 3.4 shows the count of each word category specific to the dataset files of UD_Tamil-TTB and UD_Tamil-MWTT.

Each word entry is annotated with morphosyntactic features, providing rich linguistic information that can be leveraged for feature prediction. Following shows an sample entry from the dataset.

வந்தான் *வா* VERB Gender=Masc|Number=Sing|Person=3|Tense=Past|VerbForm=Fin

Dataset	Sentences	Words	Verb	Noun	Pron	Propn	Aux
ta_mwtt-test	536	914	283	316	68	34	41
ta_ttb-train	400	2684	483	1022	55	505	111
ta_ttb-test	120	988	227	333	26	154	50
ta_ttb-dev	80	704	155	275	16	155	31

Table 3.4: Word category distribution in UD_Tamil-TTB and UD_Tamil-MWTT datasets

Each entry in the UD Tamil Treebanks follows the above CoNLL-U format, where each word is annotated with:

1. **Word**
2. **Lemma** – The base form of the word.
3. **UPOS (Universal POS Tag)** – The general part-of-speech category (e.g., NOUN, VERB).
4. **XPOS (Language-Specific POS Tag)** – Tamil-specific POS tagging conventions.

The dataset annotates words with a variety of grammatical features, depending on their word category. Table 3.5 shows the features represented by each word category.

Word Category	Grammatical Features
Verbs	Tense, Person, Gender, Number, Mood, Voice, Form, Animacy, Polarity, VerbForm, Polite, Case
Nouns	Case, Gender, Number, Person, Animacy, Polite, Polarity, Tense, VerbForm
Pronouns	Case, Gender, Number, Person, Animacy, Polite, PronType, Reflex
Proper Nouns	Case, Gender, Number, Person, Animacy, Polite
Auxiliary Verbs	Case, Gender, Number, Person, Animacy, Polite, Polarity, Voice, VerbForm, Tense, Mood

Table 3.5: Grammatical Features for Different Word Categories

3.3.2 Word Embedding Generation

Word embeddings are essential in natural language processing (NLP), representing words as continuous vectors in a high-dimensional space. Traditional word embedding models, such as Word2Vec and GloVe, assign a unique vector to each word, failing to capture morphological variations. This is particularly problematic for morphologically rich and agglutinative languages like Tamil, where words can have multiple inflections and derivations.

To address this limitation, this research utilizes FastText, a subword-based embedding model that represents words as a bag of character n-grams. Instead of treating each word as an atomic unit, FastText breaks words down into smaller overlapping character sequences (n-grams) and generates a vector representation for each n-gram. The final word embedding is computed as the sum of these subword vectors.

Example of Subword-Based Representation in Tamil

Consider the Tamil word **வந்தார்கள்**(vandārkaḷ), which means "they came". In a traditional word embedding model, if this exact word form is not seen during training, it would be treated as an out-of-vocabulary (OOV) word and have no meaningful vector representation.

However, FastText breaks this word into character n-grams, such as:

<வ>, <வந>, <வந்த>, <ந்தா>, <ந்தார்>, <தார்க>, <ர்கள்>, <ிகள்>

Each of these n-grams has an associated vector, and the final word representation is the sum of all these vectors.

This subword-level modeling provides several advantages:

- **Handles OOV Words:** If a new word shares common morphemes with known words, it can still get a meaningful representation. For example, the unseen word "**வந்தாயா**" (vandāyā – "Did you come?") will have overlapping n-grams with "**வந்தார்கள்**", allowing the model to generate a relevant embedding.

- **Captures Morphological Information:** Since Tamil words undergo inflectional changes based on tense, gender, and number, FastText preserves morphological patterns through n-gram similarity.

- **Enhances Similarity-Based Feature Prediction:** Words with similar grammatical structures are mapped closer in the embedding space, improving the accuracy of grammatical feature transfer for unseen words.

To evaluate the quality of these embeddings, cosine similarity is used to check how well n-grams from unseen words match known words. For instance, the word "சிறியதாய்" (siṟiyatāy – "small in size") is expected to align with "சிறிய" (siṟiya – "small") in the vector space due to overlapping n-grams, despite their different suffixes.

By leveraging FastText's subword-based embeddings, this research enhances grammatical feature prediction by enabling efficient feature transfer for unseen words, reducing dependence on large annotated datasets.

3.3.3 Similarity-Based Feature Prediction Algorithm

A similarity-based approach is employed to compare the embedding of an input word with embeddings of words in a pre-annotated word list. This method enables the transfer of grammatical features from known words to unseen or out-of-vocabulary (OOV) words, improving the system's ability to generalize without requiring extensive manually labeled data.

As we discussed in the section 3.3.1 the data extracted from the Universal Dependencies (UD) Tamil Treebank serves as the primary gold-standard dataset for this approach.

Methodology

- The annotated word list consists of words that have pre-labeled grammatical features (e.g., tense, gender, number, case).
- By finding the closest word in this list, we can transfer its grammatical features to the input word.
- This enables feature prediction for unseen words using embedding-based similarity rather than requiring large annotated corpora.

Algorithm for Similarity-Based Feature Prediction

1. *Train a word embedding model using* ***FastText*** *with the annotated word list as the corpus (*`gensim.models.FastText(corpus, vector_size=50, min_count=1, sg=1)`*).*

2. *Generate an embedding (vector representation) for the input word using the trained FastText model.*

3. *Compare the input word's embedding with embeddings of all words in the annotated word list using* ***cosine similarity****.*

4. *Identify the most similar word in the annotated dataset based on the similarity score.*

5. *Transfer grammatical features (such as tense, gender, number, case) from the closest matching word to the input word.*

6. *Output the predicted grammatical features for the input word.*

This approach allows for efficient feature prediction while addressing the challenge of handling OOV words, making it an effective solution for Tamil morphological analysis in low-resource NLP settings. We attempt to intrinsically determine whether it is indeed true that words similar in the embedding space are morphologically related. Qualitative evaluation, shown in figure 3.3, indicates that this is the case.

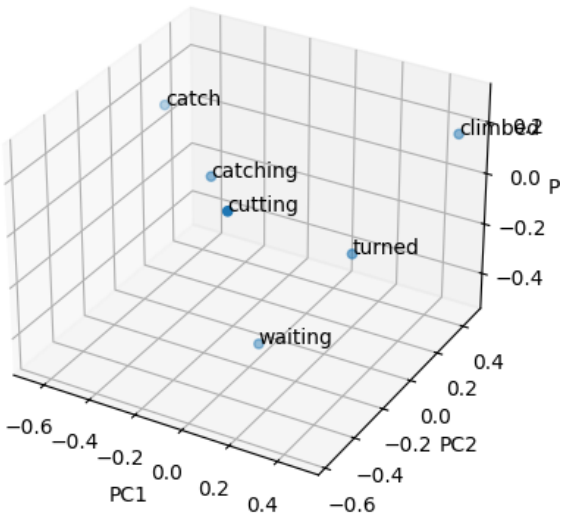

Figure 3.3: Projections of our 50 dimensional embeddings onto R^3

Key observations from the figure 3.3

1. Words with common stems vs. different stems:

- *catching* and *catch* appear close together, indicating that their embeddings capture their root word similarity despite the suffix difference.
- *cutting* and *catching* are also relatively near, suggesting some similarity in how their embeddings capture morphological relationships.

2. Effect of suffixes in standard embeddings:
   - *turned* and *climbed* (both past tense verbs) are positioned separately, likely because their base forms are different.
   - *waiting* is slightly distant from other words, possibly because "wait" has a different semantic or contextual usage in the corpus.
3. Lack of explicit morphological grouping:
   - Without suffix weighting, words are grouped based on their overall semantic and contextual usage rather than their morphological similarities.
   - Words with different base forms (e.g., catch vs. turned) are not necessarily close together, even if they share tense or suffix patterns.

3.3.4 Optimizing Similarity-Based Approach

In our similarity-based feature prediction approach, we compare the embedding of an unseen word with a list of pre-annotated words and transfer grammatical features from the closest match. However, using standard word embeddings like FastText introduces two key challenges:

1. Semantic Dominance Over Morphology
   - FastText embeddings capture semantic and contextual similarities well but do not explicitly prioritize morphological relationships.
   - For example, in standard embeddings, "running" may be closer to "*walking*" (due to similar meaning) rather than "*run*".
   - This affects grammatical feature prediction because our method should prioritize words with similar suffix patterns rather than just semantic neighbors.
2. Inconsistent Handling of Inflections

- Words with the same suffix (e.g., -ing, -ed, -s) may end up far apart in embedding space because FastText does not explicitly weight suffix information.
- Example: *jumping* (from jump) might be closer to walking rather than jump.
- This causes incorrect predictions since the most morphologically relevant match might not always be the closest in standard embedding space.

To overcome these challenges, we introduce suffix-weighted embeddings, where we modify the standard word embeddings by:

1. Assigning Higher Weight to the Suffix in Word Representation
    - Instead of treating the whole word uniformly, we apply a higher weight to the suffix, ensuring that morphological similarity is emphasized.
    - Example: In "*waiting*", the suffix "-ing" should contribute more to the similarity score when comparing it to other "-*ing*" words.
2. Considering Suffix Length
    - Longer suffixes usually indicate more complex morphological transformations (e.g., "-*ation*" vs. "-s").
    - Short suffixes (like "-s") should not dominate similarity scores, as they are more general.
    - Example: "*writes*" (from "*write*") and "*jumps*" (from "*jump*") share "-s", but the base form similarity should still matter.

By incorporating suffix-weighted embeddings, we ensure that morphological transformations (like tense, aspect, and derivation) are better represented in the embedding space. This significantly improves our ability to predict the correct grammatical features for unseen words, making our similarity-based approach more robust and reliable.

The Figure 3.4 shows how the same words plotted in Figure 3.3 has plotted after the suffix weighted embedding.

Key observations from the figure 3.4

1. Clustering of Morphologically Similar Words

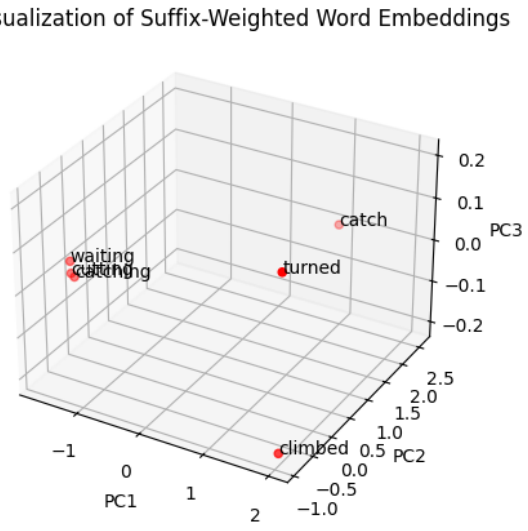

Figure 3.4: Projections of our 50 dimensional embeddings after suffix-embedding modification onto R^3

(a) "*cutting*", "*catching*", and "*waiting*" are grouped closely together in the 3D space.

(b) These words share the common "-ing" suffix, which influences their embeddings and brings them closer.

(c) This is a direct result of applying suffix-weighting, which ensures that words with similar suffixes get more similar embeddings.

2. Separation of Words with Different Suffixes

(a) "*climbed*" and "*turned*" are positioned farther apart from the "-ing" words but closer to each other.

(b) This suggests that past-tense suffixes ("-ed") contribute to a different cluster.

(c) "catch" remains somewhat distant, indicating that it does not share the same morphological structure.

3. Effectiveness of Suffix-Weighting in Morphological Representation

(a) The clustering of "*cutting*", "*catching*", and "*waiting*" highlights that suffixes play a significant role in word relationships.

(b) Without suffix-weighting, embeddings would focus only on semantic similarity, ignoring morphology.

(c) By incorporating suffix length and weight into the embeddings, we ensure that words with similar inflections are grouped together.

3.3.5 Hyper-parameter tuning

Building an effective model for grammatical feature prediction involves iterative refinement through hyperparameter tuning. This process optimizes model performance by selecting the best suffix length and suffix weight configurations, ensuring high accuracy in predicting morphological features while maintaining computational efficiency.

In this research, suffix-based hyperparameter tuning is performed to enhance grammatical feature prediction. Two key hyperparameters are explored:

- **Suffix Length (SL)**: The number of characters from the end of the word used for feature encoding. Values considered: 4,6,8,10.
- **Suffix Weight (SW)**: The relative importance assigned to suffixes in the embedding space. Values considered: 0.5, 0.6, 0.7, 0.8

To determine the optimal configuration, an exhaustive search method is employed, where each combination of SL and SW is systematically tested on a dataset containing grammatical features of noun words. The evaluation is based on prediction accuracy.

For each experiment, a specific pair (SL,SW) (e.g., (6,0.7)) is selected, and the model is trained with these settings. The accuracy of grammatical feature prediction is then assessed. The combination yielding the highest accuracy is chosen as the final suffix configuration for feature prediction.

This approach ensures that the model effectively captures morphological variations in Tamil nouns, leading to improved generalization and robust predictions in real-world applications.

Chapter 4

Implementation

4.1 Deep Learning Model for Lemma Prediction

In Premjith et al. (2018), have proposed an algorithm to do the morphological analysis in the Malayalam language. Using that algorithm they have gathered high results for their experiments. In Prasad et al. (2019), Ekanayaka et al. (2023) also have used similar kind or approach to do morphological inflection generation in the Sanskrit language. Because of these reasons, this algorithm is used to create a deep learning model in this study and did some changes according to the experiments of this research. In this research several deep learning architectures have considered. Simple RNN, LSTM, GRU and their bi-directional models. To build the deep learning model, Python Keras (version 3.8.0) library is used in this research. All the deep neural networks can be built with 'model' package in the Keras library. The models are defined as a sequence of layers. The sequential model is created using the 'Sequential()' function and each layer is added to the model one at a time as necessary.

The input layer of this model is the word embedding layer. The input dimension of the embedding layer is the vocabulary size which is the number of unique characters (48) and the output dimension of the embedding layer is set to 32 (from hyper parameter tuning). The next layer of the model is the layer that consist the deep learning architecture (RNN, LSTM, GRU). Keras provides the ability to apply all these deep learning architectures to a deep learning model. To apply bidirectional architecture a bidirectional wrapper is used which is available in keras for these deep learning architectures.

The final layer of this model is a fully connected (Dense) layer. The output dimension

of this layer is the number of possible class labels (max_label). The 'softmax' activation function is used in this output layer for multi-class classification, converting the output logits into probability distributions. The model is trained with the 'adam' optimizer (from hyper parameter tuning) and uses categorical cross-entropy loss, which is suitable for handling multi-class classification problems.

Steps to build the deep leaning model,

- Define the first layer of the network as a fully connected layer
- Define an LSTM/RNN/GRU network (network is dense)
- Define the network as bidirectional or not
- Set the activation function to Softmax
- Assign necessary parameters (Loss function, Optimizer, Batch size)

Several experiments have been conducted in this study, and the outcomes are compared with one another. It is feasible to offer an algorithm based on the procedure for using deep learning for Lemma prediction.

As mentioned in the algorithm 1, it is possible to apply deep learning for lemma prediction and obtain predictions for a given Tamil word. This model can be trained to generate different types of predictions based on the input representation and target labels.

For example, characterized words can be used as input, while their corresponding lemmas serve as the labels. This allows the model to predict the lemma for any given word. Depending on the target prediction, necessary modifications can be made to the algorithm. By simply adjusting the input representation and output labels, this deep learning model can be adapted for different morphological analysis tasks while maintaining its core structure.

Python code 1 shows the implementation of the deep learning model for lemma prediction.

4.1.1 Data Preprocessing

Data should have corresponding labels since supervised learning is being employed. Characterized words are used as input data in this study. Characterized words cannot be used

Algorithm 1 Lemma Prediction Using Deep Learning

Require: Characterized words with corresponding lemma labels
Ensure: Predicted lemma for each input word

Step 01: Obtain characterized words with their corresponding lemma labels.

Step 02: Create a unique list of characters from input words.

Step 03: Encode both input words and their lemma labels using character embeddings.

Step 04: Pad the encoded values to a common length to ensure uniform input and output sizes.

Step 05: Build the deep learning model:

Set embedding size to 32 and hidden size to 256.(from hyper-parameter tuning) Define the first layer as an embedding layer.

Use a GRU network with a hidden size of 32 and a dense output layer.

Set the activation function to Softmax.

Train the model with:

- Loss function: categorical crossentropy

- Optimizer: Adam

- Batch size: 256

Step 06: Predict the lemma for each input word.

directly as inputs into a deep learning model. All input and output variables for deep learning models must be numerical. As a result, utilizing numbers to represent the inputs and labels is crucial. Encoded described words and the labels that go with them are employed for that.

The process of converting meaningful linguistic data into numerical or vector representations, known as word encoding, is essential for preserving the contextual and relational properties of words. This transformation enables a deep learning model to identify relevant patterns in textual data and accurately capture word relationships.

In this research, character-based word representations are used as input to the deep learning model. The model learns to associate these encoded word representations with their corresponding lemma forms through a structured learning process. By leveraging deep learning, the system effectively generalizes across various word forms, enabling accurate lemma prediction even for previously unseen words.

As we discussed in the Section 3.2.1 the word-lemma pair dataset is used for the lemma prediction task. In the process of encoding the inputs, first, a list of unique words from the dataset is taken. Then characterize those words and create a list of characterized words. From that it is possible create a list of unique characters. Then it is possible to provide a unique value for each character available in that unique character list. After that we encode the characterized words using these values according to the each character value. From these steps it is possible to obtain set of encoded unique word list. Since these encoded characterized words are in different lengths, need to pad these inputs to get same length (maximum word length is the padding length) inputs.

As shown in the Figure 4.1, input data is converted into set of vectors. A similar approach is used to encode the labels as well. In Figure 4.2 have shown the encoding process of the labels.

Similar to encoding character-based words, each character in this unique list is assigned a distinct numerical value. Using these values, the labels (lemmas) are encoded accordingly. Each label is then converted into a 2D vector representation (similar to a one-hot matrix), ensuring a uniform label size for all input words. The maximum unique value serves as the matrix width, while the maximum label length determines the matrix height.

Since both input words and labels use character embeddings, a predefined list of all

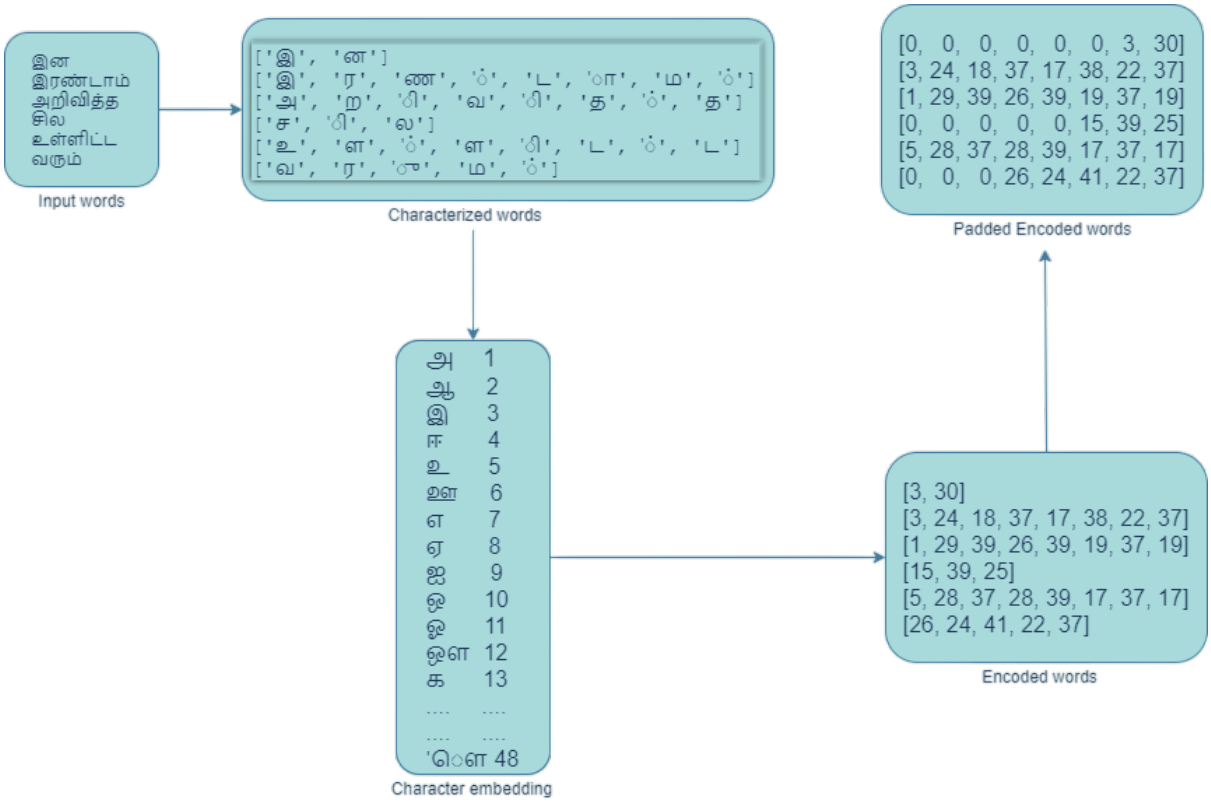

Figure 4.1: Encoding process of input data

possible Tamil characters (consonants, vowels, and modifiers) is used to ensure that the model can represent any character encountered during lemma prediction. The Tamil character vocabulary in this research consists of 48 unique characters, effectively capturing the linguistic diversity required for accurate lemma prediction.(Figure 4.3)

4.1.2 Hyperparameter Tuning

As mentioned in the section 3.2.4, a hyper-parameter tuning is used to improve the performance of the deep learning model. Since a single dense layer is considered in the deep learning model as shown in the study of Premjith et al. (2018), didn't consider the number of layers parameter for the tuning. As mentioned in the Table 3.3 initially, defined set of values for each parameter. Considering the all possible parameter combinations, is created a python list with all the results. Following algorithm 2 shows the implementation of the hyper-parameter tuning.

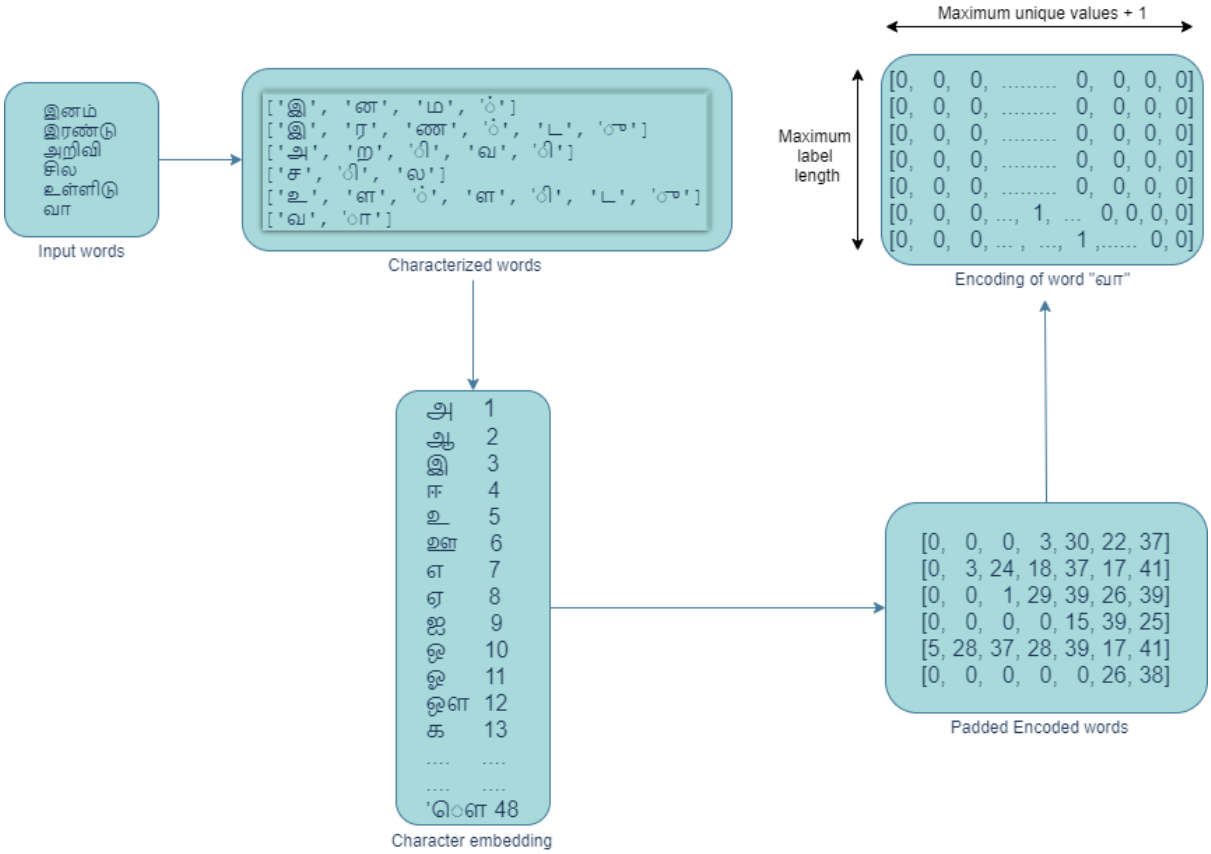

Figure 4.2: Encoding process of labels

```
vocabulary = [
    "அ", "ஆ", "இ", "ஈ", "உ", "ஊ", "எ", "ஏ", "ஐ", "ஒ", "ஓ", "ஔ",
    "க", "ங", "ச", "ஞ", "ட", "ண", "த", "ந", "ப", "ம", "ய", "ர", "ல",
    "வ", "ழ", "ள", "ற", "ன", "ஜ", "ஷ", "ஸ", "ஹ", "க்ஷ", "ஃ",
     "்", "ா", "ி", "ீ", "ு", "ூ", "ெ", "ே", "ை", "ொ", "ோ", "ௌ"
]
```

Figure 4.3: Python list created using all the possible characters (consonants and modifiers) in the Tamil language

Algorithm 2 Hyperparameter Tuning for GRU Model

Require: Set of hyperparameter combinations
Ensure: Optimal set of hyperparameters based on validation accuracy

1: Initialize empty results list
2: **for** each combination in hyperparameter set **do**
3: Define and compile the GRU-based deep learning model
4: Train the model with early stopping to prevent overfitting
5: Evaluate the model on test data
6: Store the accuracy and corresponding hyperparameters
7: **end for**

Python code 2 shows the implementation of hyper-parameter tuning..

4.1.3 Extracting the Predicted Lemma

Once the deep learning model is trained and optimized through hyperparameter tuning, the next crucial step is extracting the predicted lemma for given input words. This step is essential for evaluating the model's performance in morphological analysis, particularly in identifying the base form (lemma) of inflected words in Tamil. The model, once trained, takes an input word in its encoded representation and processes it through the trained neural network layers. The output is a sequence that represents the predicted lemma. The figure 4.4 shows the process of lemma prediction.

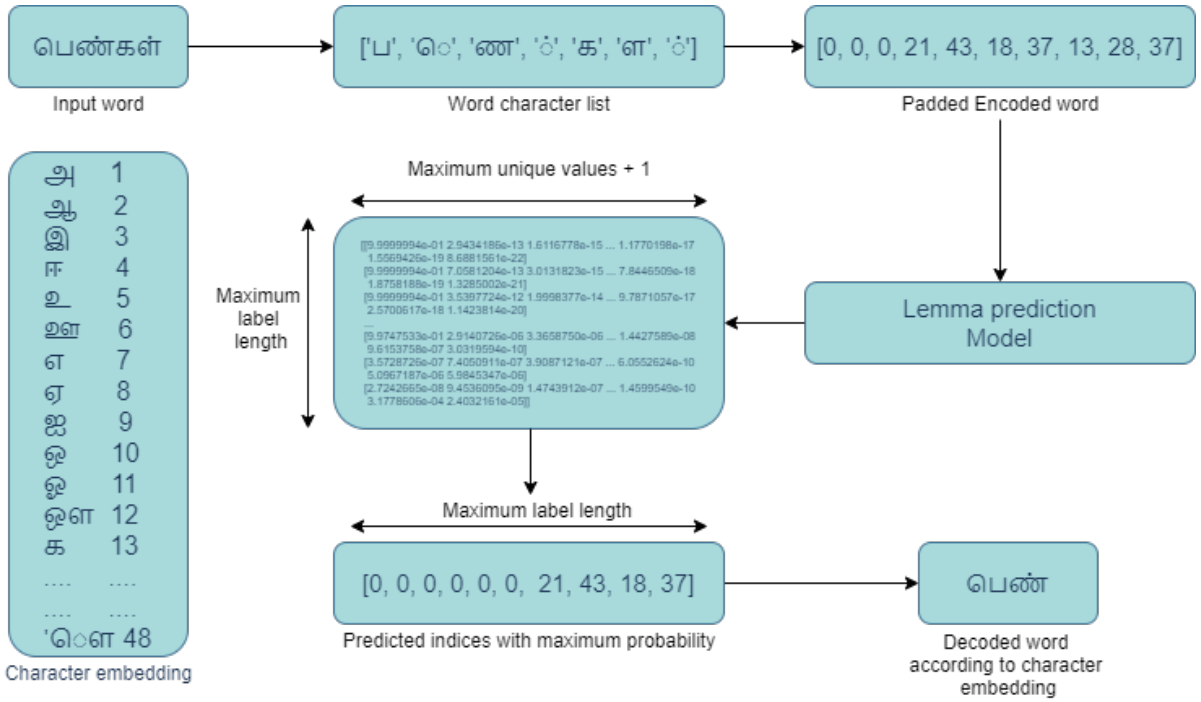

Figure 4.4: Encoding process of input data

Following Algorithm 3 shows the lemma extraction process.

Python code 3 shows the implementation of the Lemma prediction process.

Algorithm 3 Lemma Extraction Process

Require: Encoded lemma predictions from the model, character mapping dictionaries
Ensure: Decoded lemma as a readable Tamil word

```
function DECODE_LEMMA(encoded_lemma, ind2char)
    Initialize empty character list
    for each index in encoded_lemma do
        if index > 0 and index exists in ind2char then
            Append corresponding character from ind2char
        end if
    end for
    return Concatenated characters as the predicted lemma
end function
```

4.2 Embedding-Based Approach for Grammatical Feature Prediction

The embedding-based approach for grammatical feature prediction leverages word embeddings to infer the grammatical properties of unseen Tamil words. Unlike rule-based or purely deep learning models, this method relies on pre-trained word embeddings(Bojanowski et al. (2016)) and similarity-based feature transfer to predict grammatical attributes effectively.

Word representation is a fundamental aspect of natural language processing (NLP), with continuous word embeddings emerging as a powerful framework for capturing semantic relationships between words. While these embeddings have demonstrated their effectiveness in various applications, they typically treat words as atomic units, disregarding their internal morphological structure. This limitation is particularly evident in morphologically rich languages, where words consist of multiple morphemes that contribute to their meaning and grammatical role. Unlike morphologically impoverished languages such as English, where a low morpheme-per-word ratio makes holistic word representations sufficient, languages like Tamil require embeddings that account for internal word structure to accurately capture linguistic properties.

Traditional word embeddings are trained to position words close to each other in a high-dimensional space based on their semantic and contextual similarity. However, similarity in language is a multi-faceted concept, encompassing not only semantic relationships but also syntactic and morphological connections. For instance, words like ”ice” and ”cold” are semantically related, ”ice” and ”fire” share a syntactic relationship

as nouns, while ”ice” and ”icy” exhibit morphological similarity due to their shared root. In morphologically rich languages, capturing this morphological dimension is essential for effective word representation. To address this, the proposed approach enhances traditional word embeddings by integrating morphological awareness, ensuring that words with similar inflectional and derivational patterns are meaningfully clustered in the embedding space. This refinement is particularly crucial for Tamil morphological analysis, where suffixation and word-internal variations play a key role in determining grammatical features.

4.2.1 Generating Word Embeddings

As discussed in Section 3.3.2, FastText, a subword-based embedding model, is used to generate vector representations for Tamil words. Unlike traditional word embedding models such as Word2Vec, which treat words as atomic units, FastText decomposes words into character n-grams, enabling it to capture morphological variations effectively. This is particularly beneficial for agglutinative languages like Tamil, where words undergo complex inflections, derivations, and compounding.

Although pre-trained FastText embeddings for Tamil are available, they are not used in this research due to several reasons:

1. Domain-Specificity
    - Pre-trained embeddings are often trained on general-purpose corpora such as Wikipedia or Common Crawl, which may not contain rich morphological variations relevant to Tamil grammar and NLP tasks.
    - The dataset used in this research contains annotated words with detailed grammatical features, which are not captured in pre-trained models.
2. Out-of-Vocabulary (OOV) Words
    - Pre-trained embeddings may not include rare or specialized words present in Universal Dependencies(UD) Tamil Treebank (UD-Tamil-TTB UD-Tamil-MWTT).
    - Training on the dataset ensures coverage of all words relevant to morphological analysis.

Since no pre-trained Tamil embeddings are used, a FastText model is trained on the dataset using the annotated corpus. The model is initialized as follows:

```
import gensim

model = gensim.models.FastText(corpus, vector_size=50, min_count=1, sg=1)
```

Listing 4.1: Training the Fasttext model on Universal dependancies annotated word list

- **corpus**: The collection of Tamil words from the dataset.
- **vector_size=50**: Each word is represented as a 50-dimensional vector.
- **min_count=1**: Ensures even rare words are included in the training.
- **sg=1**: Uses the Skip-gram model for better representation of rare words.

Skip-gram Predicts surrounding words for a given target word and it is suitable for Low-resource languages, rare words, morphologically rich languages (Tamil, Finnish, etc.). Since Tamil has complex word formations, Skip-gram (sg=1) is preferred for Tamil morphological analysis as it helps learn embeddings for infrequent and out-of-vocabulary (OOV) words more effectively.

4.2.2 Generating suffix-weighted Word Embeddings

In order to enhance the representation of Tamil words for grammatical feature prediction, a suffix-weighted embedding approach is employed. Unlike traditional word embeddings that treat words as atomic units, this method considers both the root and suffix components of a word, capturing the morphological structure more effectively. This is particularly crucial for Tamil, an agglutinative language, where suffixes play a key role in defining grammatical properties such as tense, case, number, gender, and mood.

The suffix-weighted embedding is computed by decomposing a word into two parts: the root (initial segment) and the suffix (final segment). Each of these segments is assigned an individual embedding using a FastText model trained on the annotated dataset. The final word embedding is then derived as a weighted sum of the root and suffix embeddings, where a predefined suffix weight hyperparameter determines the contribution of

the suffix. The formula for computing the suffix-weighted embedding is given as follows:

$$E_{word} = (1 - w).E_{root} + w.E_{suffix}$$

where E_{word} represents the final word embedding, E_{root} is the embedding of the root, E_{suffix} is the embedding of the suffix, and w is the suffix weight hyperparameter that controls the influence of the suffix.

To determine the optimal values for suffix length and suffix weight, hyperparameter tuning(Section 4.2.3) is conducted by testing different combinations of these parameters and evaluating their impact on grammatical feature prediction accuracy. This ensures that the model effectively captures both the semantic meaning and morphological properties of Tamil words, leading to improved performance in predicting grammatical features, especially for unseen words.

Following code segment 4.2 in python show the implementation of suffix weighted embedding return function.

```
def suffix_weighted_embedding(word, model, suffix_length, suffix_weight):
    if len(word) > suffix_length:
        root_part = word[:-suffix_length]
        suffix_part = word[-suffix_length:]
    else:
        root_part = word
        suffix_part = word

    root_vec = model.wv[root_part]
    suffix_vec = model.wv[suffix_part]

    combined_vec = (1 - suffix_weight) * root_vec + suffix_weight *
    suffix_vec
    return combined_vec
```

Listing 4.2: Suffix-Weighted Embedding generation

4.2.3 Hyperparameter Tuning

As mentioned in the section 3.3.5, a hyper-parameter tuning is used to improve the performance of the grammatical feature prediction model. As mentioned two key hyper-

parameters are explored:

- **Suffix Length (SL)**: The number of characters from the end of the word used for feature encoding. Values considered: 4,6,8,10.
- **Suffix Weight (SW)**: The relative importance assigned to suffixes in the embedding space. Values considered: 0.5, 0.6, 0.7, 0.8

. Considering the all possible parameter combinations, is created a python list with all the results. Following algorithm 4 shows the implementation of the hyper parameter tuning process of .

Algorithm 4 Optimizing Suffix-Weighted Embeddings for Grammatical Feature Prediction

Require: Annotated corpus, trained FastText model, suffix lengths SL, suffix weights SW, test words

Ensure: Best suffix length and suffix weight for highest accuracy

$best\_params \leftarrow None$
$best\_accuracy \leftarrow 0$
for each $suffix\_length$ in SL **do**
    for each $suffix\_weight$ in SW **do**
        Initialize $word\_embeddings$ as an empty dictionary
        for each word in corpus **do**
            Compute suffix-weighted embedding:
            $embedding \leftarrow SuffixWeightedEmbedding(word, model, suffix\_length, suffix\_weight)$
            Store $(word, embedding)$ in $word\_embeddings$
        end for
        $total\_accuracy \leftarrow 0$, $total\_words \leftarrow 0$
        for each word in test words **do**
            $true\_features \leftarrow$ actual grammatical features
            Predict features using similarity-based method
            $predicted\_features \leftarrow PredictGrammaticalFeatures(word, suffix\_length, suffix\_weight, word\_embeddings)$
            if $predicted\_features$ is not None **then**
                Compute accuracy:
                $accuracy \leftarrow CalculateAccuracy(predicted\_features, true\_features)$
                Update $total\_accuracy$ and $total\_words$
            end if
        end for
        Compute average accuracy:
        $avg\_accuracy \leftarrow total\_accuracy/total\_words$
        if $avg\_accuracy > best\_accuracy$ **then**
            $best\_accuracy \leftarrow avg\_accuracy$
            $best\_params \leftarrow (suffix\_length, suffix\_weight)$
        end if
    end for
end for
return $best\_params, best\_accuracy$

Python code 4 shows the implementation of the hyper parameter tuning process. The arguments passed to the *tune_parameters* function are,

1. **corpus** :- The list of words in training dataset of UD-Tamil-TTB(Only test dataset is available on UD-Tamil-MWTT dataset so ignored).
   Ex. ['தண்டனை', 'குறை', 'முஸ்லீம்', 'நீட்டிப்பு', 'அமெரிக்கர்', 'அதிகாலை, ...]

2. **morph_analysis** :- The dictionary contains key-value pairs where each word in the corpus are keys and their corresponding morphological tags are values.
   Ex. {'தண்டனை': 'Case': 'Nom', 'Gender': 'Neut', 'Number': 'Sing', 'Person': '3', 'Animacy': None, 'Polite': None, 'Polarity': None, 'Tense': None, 'VerbForm': None,
   'குறை': 'Case': 'Nom', 'Gender': 'Neut', 'Number': 'Sing', 'Person': '3', 'Animacy': None, 'Polite': None, 'Polarity': None, 'Tense': None, 'VerbForm': None,
   'முஸ்லீம்': 'Case': 'Nom', 'Gender': 'Neut', 'Number': 'Sing', 'Person': '3', 'Animacy': None, 'Polite': None, 'Polarity': None, 'Tense': None, 'VerbForm': None,
   'நீட்டிப்பு': 'Case': 'Nom', 'Gender': 'Neut', 'Number': 'Sing', 'Person': '3', 'Animacy': None, 'Polite': None, 'Polarity': None, 'Tense': None, 'VerbForm': None,
   'அமெரிக்கர்': 'Case': 'Nom', 'Gender': 'Com', 'Number': 'Sing', 'Person': '3', 'Animacy': None, 'Polite': 'Form', 'Polarity': None, 'Tense': None, 'VerbForm': None,
   .... }

3. **test_words** :- The dictionary contains key-value pairs where each word in the test dataset of UD-Tamil-TTB and UD-Tamil-MWTT are keys and their corresponding morphological tags are values. (Used as ground truth component to calculate the accuracy of predicted features)
   Ex. {'மக்கள்': 'Case': 'Nom', 'Gender': 'Com', 'Number': 'Plur', 'Person': '3', 'Animacy': 'Anim', 'Polite': None, 'Polarity': None, 'Tense': None, 'VerbForm': None,
   'உடல்நிலை': 'Case': 'Nom', 'Gender': 'Neut', 'Number': 'Sing', 'Person': '3', 'Animacy': None, 'Polite': None, 'Polarity': None, 'Tense': None, 'VerbForm': None,
   ... }

4. **suffix_lengths**: The number of characters from the end of the word used for feature encoding. Values considered: 4,6,8,10.

   - Tamil suffixes typically range from 3–10 characters.

- Multiples of 2 allow systematic coverage of short to long morphological patterns while keeping the model efficient and the feature space manageable.

5. **suffix_weights**: The relative importance assigned to suffixes in the embedding space. Values considered: 0.5, 0.6, 0.7, 0.8

    - All the values more than or equal to 0.5 has been considered in order to increase the weight of the suffix embedding.

4.2.4 Predicting grammatical feature

The prediction process relies on a pre-annotated corpus, specifically the Universal Dependencies Tamil Treebank (UD-Tamil-TTB UD-Tamil-MWTT), which serves as a gold standard dataset containing words with predefined grammatical features. The FastText model is trained on this dataset to generate embeddings that preserve morphological relationships between words. Given an unseen word, its embedding vector is computed and compared against words in the annotated dataset using cosine similarity. The word with the highest similarity score is identified, and its grammatical features are transferred to the input word.

Following algorithm 5 shows the implementation of the grammatical feature prediction.

Algorithm 5 Suffix-Weighted Embedding for Grammatical Feature Prediction

Require: Annotated corpus, trained FastText model, suffix length SL, suffix weight SW
Ensure: Predicted grammatical features for unseen words

function SuffixWeightedEmbedding($word, model, SL, SW$)
    if length of word > SL **then**
        $root\_part \leftarrow word[: -SL]$
        $suffix\_part \leftarrow word[-SL :]$
    else
        $root\_part, suffix\_part \leftarrow word$
    end if
    Compute word vector:
    $root\_vec \leftarrow model.wv[root\_part]$
    $suffix\_vec \leftarrow model.wv[suffix\_part]$
    Compute final weighted vector:
    $embedding \leftarrow (1 - SW) \times root\_vec + SW \times suffix\_vec$
    return *embedding*
end function
function PredictGrammaticalFeatures($new\_word, SL, SW, word\_embeddings$)
    $new\_word\_embedding \leftarrow SuffixWeightedEmbedding(new\_word, model, SL, SW)$
    if $new\_word\_embedding$ is None **then**
        return None, None
    end if
    for each word in corpus **do**
        $embedding, feature\_vector \leftarrow word\_embeddings[word]$
        Compute similarity:
        $similarity \leftarrow CosineSimilarity(new\_word\_embedding, embedding)$
        Store $(word, similarity)$
    end for
    $most\_similar\_word \leftarrow$ word with highest similarity
    $predicted\_features \leftarrow morph\_analysis[most\_similar\_word]$
    return $predicted\_features, most\_similar\_word$
end function

Python code 5 shows the implementation of the grammatical feature prediction.

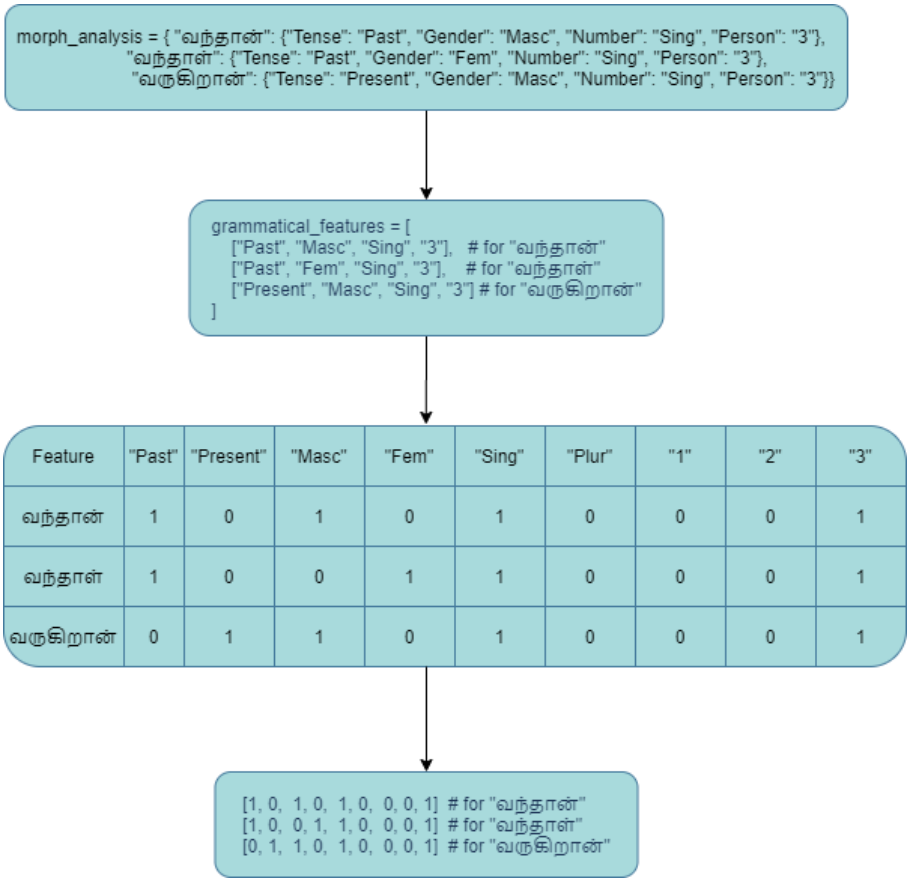

Figure 4.5: Creation process of feature vectors

The arguments passed to the *predict_grammatical_features* function are,

1. **Word** :- The input word
2. **suffix_length** :- The number of characters from the end of the word used for feature encoding in order to pass to the suffix_weighted_embedding function.
3. **suffix_weight** :- The relative importance assigned to suffixes in the embedding space in order to pass to the suffix_weighted_embedding function.
4. **word_embeddings** :- Word_embeddings is a contains key-value pairs where each word in the training dataset is a key. Value contains a tuple with corresponding (suffix weighted embedding of the word, feature vector which represents the morphological features)
   (a) **Suffix weighted embedding** (Section 4.2.2)
   (b) **Feature vector** :- The feature vector is created as explained in the figure 4.5.

Chapter 5

Results and Evaluation

This section presents the experiments conducted in this research and the corresponding results obtained. The study focuses on Tamil morphological analysis, specifically addressing lemma prediction using deep learning and grammatical feature prediction using an embedding-based similarity approach. The evaluation measures the effectiveness of these approaches in accurately predicting word lemmas and morphological attributes such as tense, gender, number, case, mood etc.

To achieve this, different deep learning architectures were explored for lemma prediction, including RNN, LSTM, GRU and bidirectional variants of them. The performance of these models was assessed using exact-string match accuracy, ensuring that the predicted lemma aligns with the actual root form. Additionally, a suffix-weighted embedding approach was employed for grammatical feature prediction, leveraging word embeddings trained on the Universal Dependencies Tamil Treebank (UD-Tamil-TTB UD-Tamil-MWTT). The evaluation involved testing different suffix lengths and suffix weights to optimize the performance of the similarity-based approach.

The evaluation metrics include exact-string match accuracy, per-tag accuracy for grammatical features, overall classification accuracy, exact grammatical feature match performance etc will be more deeply explained in latter section. By analyzing these results, this study provides insights into the strengths and limitations of each method, contributing to the advancement of Tamil NLP and computational morphology.

5.1 Lemma Prediction Evaluation

5.1.1 Experimental Setup

The evaluation of lemma prediction was conducted using a dataset of word-lemma pairs which is discussed in section 3.2.1, where each word was mapped to its corresponding root form. The dataset was carefully curated to include various verb conjugations, noun inflections, and derived forms to ensure a diverse and representative evaluation.

5.1.1.1 Dataset Preparation

The dataset was divided into training (80%) and testing (20%) splits to train and evaluate the deep learning model effectively. Each word in the dataset was represented using character-level encoding as discussed in 4.1, allowing the model to learn morphological transformations without relying on large annotated corpora as lookup table.

5.1.1.2 Model Architecture and Training

For lemma prediction, character-based deep learning models were implemented, including RNN, LSTM, and GRU architectures. Each model was trained on the encoded dataset using an embedding layer followed by bidirectional recurrent layers to capture both forward and backward dependencies in word structures. The final model was optimized based on categorical cross-entropy loss and trained using the Adam optimizer. The table 5.1 shows the training configuration used to train the model.

Parameter	**Value**
Embedding size	32 (from Section 5.1.2)
Hidden size	256 (from Section 5.1.2)
Batch size	256 (from Section 5.1.2)
Optimizer	Adam (from Section 5.1.2)
Activation	Softmax (from Section 5.1.2)
Loss function	Categorical Cross-Entropy
Epochs	30 (with early stopping)

Table 5.1: Training configuration

5.1.2 Hyperparameter Tuning

Since a set of parameter combinations are considered to identify the most suitable values for the each hyper-parameter (Section 3.2.4), the performance of the each combinations provided in the predictions should be analyzed.

Embedding size	Hidden size	Activation	Optimizer	Batch size	Score
32	32	relu	SGD	512	0.8648
32	32	relu	SGD	256	0.8648
32	32	relu	SGD	128	0.8648
32	32	relu	SGD	64	0.8648
32	32	relu	SGD	32	0.8769
32	32	relu	Adam	256	0.8816
32	32	relu	Adam	64	0.8950
32	32	relu	Adam	512	0.8905
32	32	relu	Adam	32	0.8976
32	32	relu	Adam	128	0.9081
256	512	softmax	Adam	32	0.9887
32	256	softmax	Adam	32	0.9886
32	256	softmax	Adam	512	0.9883
32	256	softmax	Adam	64	0.9890
256	512	softmax	Adam	64	0.9898
32	128	softmax	Adam	32	0.9891
32	256	softmax	Adam	128	0.9897
256	512	softmax	Adam	128	0.9900
32	128	softmax	Adam	64	0.9896
32	128	softmax	Adam	128	0.9902
32	128	softmax	Adam	512	0.9904
32	128	softmax	Adam	256	0.9909
32	256	softmax	Adam	256	0.9912

Table 5.2: Sample results of hyper-parameter tuning

In the Table 5.2 the score means the results provided by the ”keras.model”. From this

value, it is possible to get an idea about the accuracy of each result. Considering the results of hyper-parameter tuning the highest accuracy was obtained, when the morphological analyzer used embedding_size = 32, units = 256 activation = softmax, optimizer = Adam and batch_size = 256 as the hyper-parameters. Using these hyper-parameters the model was trained and analyzed the predictions.

5.1.3 Evaluation Metrics

5.1.3.1 Exact-String Match

Exact-String Match is one of the primary evaluation metrics used for lemma prediction. It measures how often the predicted lemma exactly matches the actual lemma in the test dataset. This metric is particularly important because in morphological analysis, small variations in predictions (such as missing or extra characters) can lead to incorrect interpretations of the word's meaning and structure.

Since lemma prediction involves transforming an inflected form into its root form, an exact match ensures that the model accurately reconstructs the base form without any errors. Higher Exact-String Match accuracy indicates that the model has learned the correct morphological transformations, making it more reliable for real-world applications in Tamil NLP.

Let L_p be the predicted lemma and L_a be the actual lemma. The Exact-String Match Accuracy (ESM) is defined as:

$$ESMAccuracy = \frac{Number\ of\ Correctly\ Predicted\ Lemmas}{Total\ Number\ of\ Words\ in\ Test\ Set} * 100$$

If $L_p = L_a$ for a given word, it is counted as correct; otherwise, it is incorrect.

5.1.4 Results and Analysis

5.1.4.1 Different deep learning architectures

Since several deep learning architectures are considered to identify the most suitable one, the dataset was trained using same deep learning model with different deep learning architectures. Here, same hyper-parameters were used and the dataset size (Both training and testing) for each experiment and calculated the accuracy of each experiment. Study

of Morphological Inflection Generation in Sanskrit language (Prasad et al. (2019)) has also used this approach to identify the best performing deep learning architectures for their study. The same approach was used to identify the best performing deep learning architecture for this study. To do this six different deep learning architectures were selected. RNN, LSTM, GRU, Bi-RNN, Bi-LSTM and Bi-GRU are the six deep learning architectures that have selected for this study. Studies like Premjith et al. (2018), Prasad et al. (2019) and Ekanayaka et al. (2023) have used these deep learning architectures to do the morphological analysis.

Since different deep learning architectures are considered, it helps to identify the behaviour of the morphological analyzer with different deep learning architectures. Then the results of these experiments were analyzed against each other. From that, it is possible to identify the most suitable deep learning architecture for our morphological analyzer.

Sample input: ['ഩ', 'ഠ◌', 'ന', '◌ി', 'ധ', '◌ി', 'ല', '◌', 'ല']

Sample label: ['ഩ', 'ഠ◌', 'ന', '◌ി', 'ധ', '◌ി', 'ല', '◌്']

Considering the results in Table 5.3, bidirectional LSTM has provided the highest

Architecture	**Total**	**Train**	**Test**	**Correct**	**Accuracy**
RNN	70,007	56,005	14,002	12,370	88.34 %
GRU	70,007	56,005	14,002	12,688	90.61 %
LSTM	70,007	56,005	14,002	12,438	88.83 %
BI-RNN	70,007	56,005	14,002	12,306	87.88 %
BI-GRU	70,007	56,005	14,002	12,884	92.01 %
BI-LSTM	70,007	56,005	14,002	13,050	**93.20** %

Table 5.3: Performance of the Lemma prediction with different deep learning architectures for test dataset

results for the lemma prediction. Using bidirectional LSTM as the deep learning architecture has provided the accuracy around 93.20%.

Unidirectional models only stores past information because the only input the model has seen is old information. But in the bidirectional models, it will use the input in two ways, one from the past to the future and the other from the future to the past. What distinguishes this method from the unintentional is that running back you to store information from the future. Using two hidden regions together you can save information from both past and future at any time. Bidirectional models use the past and the future

information for the predictions because of that the performance of the morphological analyzer is getting increased when we are using the bidirectional deep learning architectures.

Comparing the deep learning architecture without the bidirectional wrappers, GRU has provided the best results. Considering the training parameters GRU uses less training parameters, because of that it uses less memory, perform faster and train faster. Since this study is considering words, and morphological analysis of words, considering about long sequences are not needed. Because of that using GRU as the deep learning architecture can improve the performance. Because of that GRU architecture has provided the highest accuracy among the selected deep learning architectures in the case of bidirectional wrapper absence.

5.1.4.2 Predictions of entire new set of data

Good morphological analyzer should be able to predict any kind of word. To check this ability of a morphological analyzer the predictions of the model needed to be checked with a different set of words. In this study 70,006 total number of words were used to train and test the deep learning model. As we have discussed in the section 3.2.1 a set of Tamil words extracted from UD Tamil tree bank considered as test data which are unseen since that is the dataset considered as the gold standard. 4102 total number of Tamil words has been extracted from UD Tamil tree bank which is completely unseen. To compare the results, the BI-LSTM model with the maximum test accuracy has tested against these words.

Sample input: ['ஷ', 'ெ◌', 'ண', '◌ி', 'ய', '◌ி', 'ட', '◌', 'ட']

Sample label: ['ஷ', 'ெ◌', 'ண', '◌ி', 'ய', '◌ி', 'ட', '◌ு']

Upon inspecting the results in detail, several key factors contribute to the errors in

No. of words	Correct	Accuracy
4102	1241	30.25 %

Table 5.4: Performance of the Lemma prediction model(BI-LSTM model) with set of unseen words

lemma prediction. Since Tamil is a morphologically rich language, the training dataset does not cover all possible inflected and derived forms, leading to gaps in the model's ability to generalize to unseen words. Additionally, a significant portion of errors arises

due to single-character or few character mispredictions, where a minor mistake in predicting one or few characters alters the entire lemma, affecting the overall accuracy. Another observation is that some of the misclassified words belong to the category of proper nouns, which often have unique morphological patterns and may not follow the same transformation rules as other word categories. These challenges highlight the limitations of the current deep learning model in handling out-of-vocabulary (OOV) words and rare morphological variations, emphasizing the need for improved generalization strategies.

5.1.5 Error analysis

The analysis of errors in lemma prediction is crucial to understanding the strengths and limitations of our Bi-LSTM model in Tamil morphological analysis. While the model successfully predicts lemmas for most words, it exhibits low accuracy on unseen words due to morphological complexity and out-of-vocabulary (OOV) challenges. This section presents a detailed error analysis, examining mispredictions and their underlying causes.

Table 5.5 presents a subset of unseen words where the model's predictions were incorrect compared to the ground truth.

Index	**Word**	**Correct Lemma**	**Predicted Lemma**	**Identified Error**	**Reason**
0	கோவில்	கோவில்	கோகில்	Character-Level Error	Incorrect character replacement
2	கிராமத்தில்	கிராமம்	கிராாரம்	Character-Level Error	Incorrect character formation ராம -> ராார
4	எப்போது	எப்போது	எப்போடு	Character-Level Error	Incorrect character substitution
8	வரவில்லை	வா	வவல்	lemmatization fail	Insufficient training dataset

31	புத்தகத்தைக்	புத்தகம்	பதுதகம்	Modifier Substitution Error	Incorrect modifier prediction ◌ -> ◌
34	உயரமான	உயரம்	யரமம்	Character-Level Error	Incorrect character prediction உ -> ய
48	சிறியதாக	சிறியது	சிறி	Morphological Truncation Error	Over-truncation of 'சிறியது'
207	உயரமாக	உயரம்	உயரரம்	Character-Level Error	Incorrect character insertion (ர)
227	இந்தத்	இந்த	◌ாத்	Incorrect segmentation of suffix	Presence of Consequent modifiers
382	வெயிலில்	வெயில்	◌ெயில்	Character-Level Error	Incorrect character formation (Absence of character 'வ')
388	திரும்பி	திரும்பு	திரும்பி	Character-Level Error	Incorrect modifier prediction ◌ -> ◌
516	உடல்நிலையய்	உடல்நிலை	உடல்ந்லை	Character-Level Error	Incorrect modifier prediction ◌ -> ◌
788	துறையாகும்	துறை	◌ுறை	Character-Level Error	Presence of Consequent modifiers
1030	முடிவடையும்	முடிவடை	டவடை	lemmatization fail	Loss of critical inflection

1467	மலேசிய	மலேசியா	மமமயமி	OOV Error	Unseen word in training (Proper noun)
2003	கருணாகரனின்	கருணாகரன்	உக்ர்ாரர்	OOV Error	Unseen word in training (Proper noun)
2537	பாகிஸ்தானுக்குக்	பாகிஸ்தான்	ஸஆாஸமதான்	OOV Error	Unseen word in training (Proper noun)
2494	இந்தியாவுக்க்	இந்தியா	இநிதியாய்	OOV Error	Unseen word in training (Proper noun)

Table 5.5: Error analysis for Lemma prediction

By manually analyzing these mispredictions, we identify the following key error categories:

1. Character-Level Errors:
    - The model incorrectly modifies single or multiple characters(Characters also includes the modifiers), leading to incorrect lemmas.
    - The challenging part in the lemma prediction is finding the suffix part of the lemma correctly. In that case most of the words' suffixes are correctly predicted while wrongly predicting few character at the beginning of the words.
    - An additional component of post-processing part could be added to only predicting the suffixes while repeating the same few characters present in the prefix of the input word.
    - Example: கோவில் -> கோகில் (Incorrect character replacement)
2. Morphological Truncation Errors:

- The model over-truncates or under-truncates the word, leading to an incomplete lemma.
- Example: சிறியதாக -> சிறி (Incorrect truncation of 'சிறியது')

3. Out-of-Vocabulary (OOV) Errors:
   - Words that are not frequently present in the training data result in inaccurate lemma predictions.
   - The model poorly performs for the proper nouns where model suffers to identify similar pattern while it is not available.
   - Example: பாகிஸ்தானுக்குச் -> ஸஆாஸமதான் (Completely incorrect lemma due to lack of OOV handling)(Ground truth :- பாகிஸ்தான்)
   - In the above example you can see the suffix('தான்') has been correctly predicted so if there is a post-processing component as we discussed before, which identifies the correct prefix from the input word and joins with the predicted suffix a correct lemma can be obtained.
     Example: பாகிஸ்(Post-processed prefix) + தான்(predicted suffix)

4. Incorrect segmentation of suffix:
   - Presence of consequent modifiers in the predicted word.
   - Presence of consequent modifiers is not allowed in a Tamil word.
   - As we have discussed the model predicts the character in the figure 4.3. Since modifiers also considered as independent characters there are possibilities where a predicted word can contain consequent modifiers.
   - To avoid this, all possible combinations with tamil characters and modifiers could be considered as the vocabulary but in that case even though we minimize the possibility of consequent modifiers' occurrence but in the same time, it reduces the accuracy since number of possible characters can be predicted are more.
   - Example of the modified vocabulary is shown in the figure 5.1

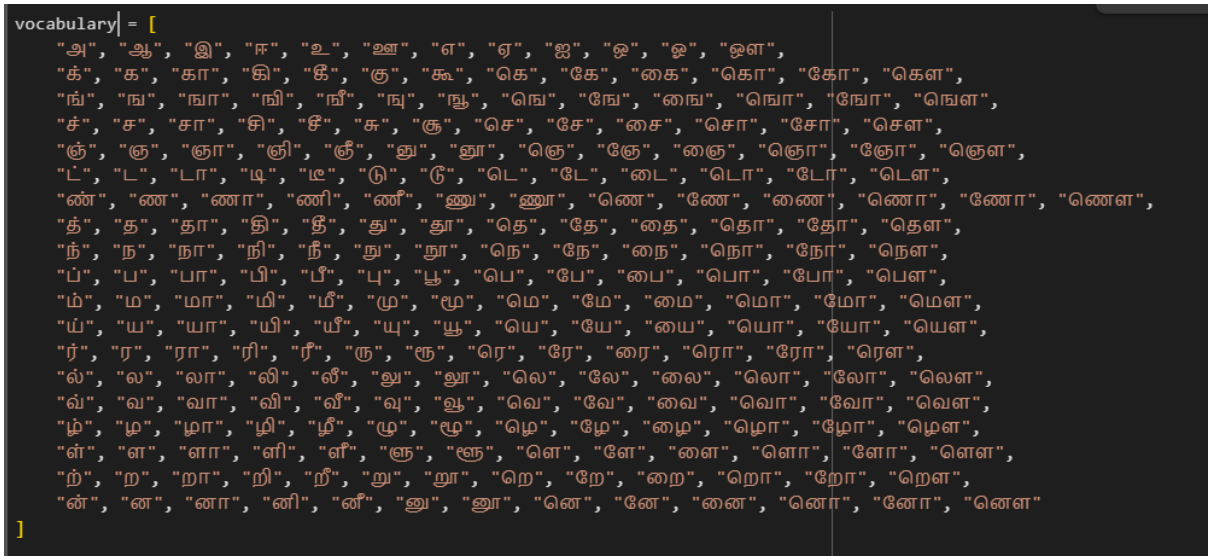

Figure 5.1: This list includes all the independent vowels and consonant modifications

5.2 Grammatical feature Prediction Evaluation

5.2.1 Experimental Setup

For the evaluation of grammatical feature prediction, we utilize the ta_ttb-train dataset for the similarity comparison. The evaluation is performed using the ta_mwtt-test, ta_ttb-test, and ta_ttb-dev datasets which are universal dependancies(UD) datasets. These datasets serve as test data against our evaluation metrics to analyze the performance of the embedding-based approach for grammatical feature prediction. The dataset contains lack of comprehensive morphological annotations and the variability in data quality restrict the study to only a subset of word categories which are Nouns, Verbs, Auxiliary, Pronouns and Propernouns, ensuring a more reliable and meaningful analysis.

5.2.2 Hyperparameter Tuning

We employ hyperparameter tuning to determine the optimal values for Suffix Length and Suffix Weight as we have discussed in Section 3.3.5, which influence feature prediction accuracy. The tuning process involves selecting configurations that maximize accuracy. The overall accuracy is calculated using the below method which is one of the evaluation metrics we use to analyze the results of the grammatical feature prediction

component.(Overall Accuracy (Across All Words)).

$$OverallAccuracy = \frac{\sum WordAccuracies}{NumberofWords}$$

Example:

- Word 1 Accuracy: 0.67
- Word 2 Accuracy: 0.80
- Overall Accuracy: (0.67 + 0.80) / 2 = 0.735 (73.5%)

The section 5.2.3 explains more on how does the word accuracy is calculated. The table 5.6 shows the Hyperparameter tuning results of grammatical feature prediction. The

Suffix Length	Suffix Weight	Accuracy
4	0.5	0.89
4	0.6	0.89
4	0.7	0.89
4	0.8	0.89
6	0.5	0.86
6	0.6	0.87
6	0.7	0.88
6	0.8	0.89
8	0.5	0.85
8	0.6	0.86
8	0.7	0.87
8	0.8	0.87
10	0.5	0.86
10	0.6	0.86
10	0.7	0.86
10	0.8	0.86

Table 5.6: Hyperparameter tuning result to determine the optimal suffix length and suffix weight to consider for maximum accuracy

hyper parameter tuning results recorded in the table 5.6 is represented in the graph 5.2

for the convenience of easy reading. Based on the graph, the optimal suffix length and

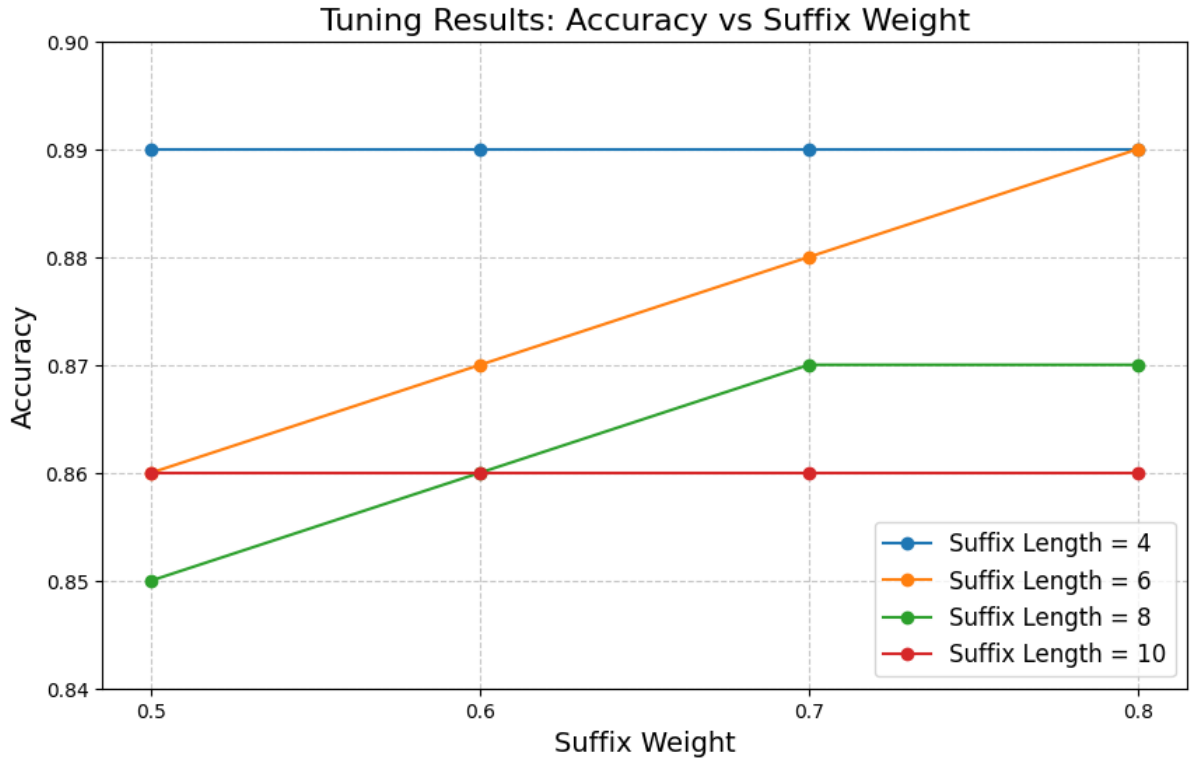

Figure 5.2: Hyper parameter tuning results for grammatical feature prediction

suffix weight should be chosen to maximize accuracy. Here's the analysis:

- Suffix Length = 4: Consistently achieves the highest accuracy (0.89) across all suffix weights.
- Suffix Length = 6: Improves as the suffix weight increases and reaches 0.89 at weight 0.8.
- Suffix Length = 8 and 10: Show lower overall accuracy (0.87) across all suffix weights.

Since Suffix Length = 4 achieves the best accuracy across all suffix weights without needing higher suffix weights, the optimal configuration is Suffix Length = 4 and Suffix Weight = 0.8, as it ensures the highest accuracy while keeping the suffix length minimal.

5.2.3 Evaluation Metrics

To assess the performance of grammatical feature prediction, we employ the following evaluation metrics:

1. **Word-Level Feature Prediction Accuracy**

   The work Pawar et al. (2023) also uses the same evaluation metric to calculate the word level accuracy. Accuracy for a word is computed as:

$$Accuracy = \frac{NumberOfCorrectlyPredictedFeatures}{TotalNumberOfFeatures}$$

   Example:

   - Predicted: Case: Nom, Gender: Masc, Number: Sing
   - True: Case: Nom, Gender: Masc, Number: Plur
   - Correct Predictions: $2/3 \rightarrow 67\%$

2. **Overall Accuracy (Across All Words)**

   Overall accuracy is the extended version of word-level feature prediction accuracy. Overall accuracy of all the words are computed as:

$$OverallAccuracy = \frac{\sum WordAccuracies}{NumberofWords}$$

   Example:

   - Word 1 Accuracy: 0.67
   - Word 2 Accuracy: 0.80
   - Overall Accuracy: (0.67 + 0.80) / 2 = 0.735 (73.5%)

3. **Exact Match Accuracy**

   Measures the percentage of words where all grammatical features were correctly predicted. This is an additional evaluation metric which is not reported in any of the previous works. This evaluation metric shows how good the grammatical prediction is in predicting all the grammatical features correctly.

4. **Per-Tag Accuracy**

   As same as for the overall accuracy the work Pawar et al. (2023) uses the per-tag accuracy for each grammatical feature (e.g., Case, Gender, Tense), the proportion of correct predictions is calculated. The type of grammatical features varies according to the word categories as we discussed in Table 3.5.

5.2.4 Results and Analysis

5.2.4.1 Overall Performance Summary

This section records the overall accuracy and exact match accuracy of all word categories we are considering. The Table 5.7 shows the overall performance of the new words that are extracted from the test and dev dataset from the universal dependency as we discussed before. The figure 5.3 shows a clear representation for the comparison between different word categories.

Word Category	Total Words	Overall Accuracy	Exact Match Accuracy
Noun	838	0.90	0.50
Verb	601	0.76	0.38
Pronoun	80	0.70	0.30
Proper Noun	275	0.85	0.56
Auxiliary	97	0.78	0.45

Table 5.7: Grammatical feature prediction overall performance

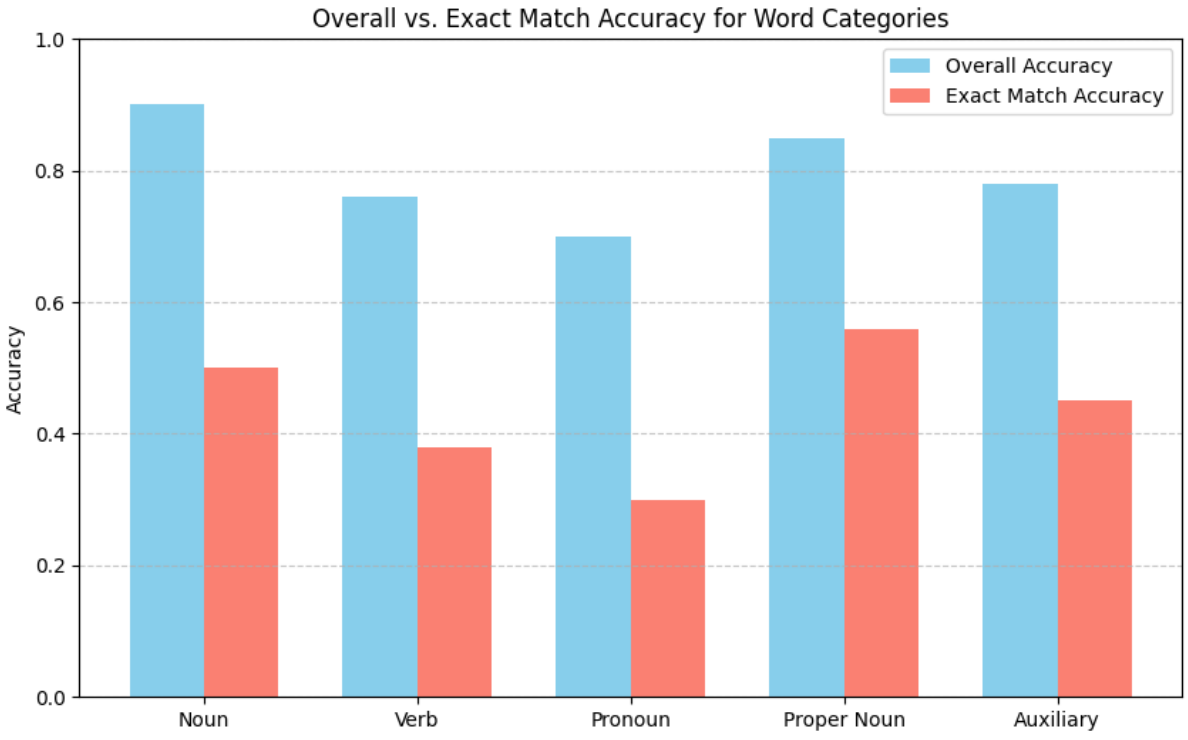

Figure 5.3: Bar chart represents the overall accuracy and exact-match accuracy among different word categories

Based on the figure 5.3 we can conclude that the overall Accuracy is consistently higher than Exact Match Accuracy across all categories, meaning that the model is able to assign close but not always perfect labels. Nouns and Proper Nouns have the highest overall accuracy (above 0.85) because of the less number of grammatical features represented by them. Usually nouns are the word category with less inflected or derivated form in Tamil. Verbs and Pronouns have lower accuracy, likely due to their morphological complexity.

5.2.4.2 Per-Tag Accuracy for Each Word Category

This section records the accuracy calculated for the each word category against its grammatical features. The Table 5.8 shows the per-tag accuracy of the new words that are extracted from the test and dev dataset from the universal dependency as we discussed before. The heatmap 5.4 shows a clear representation for the comparison between different word categories.

Based on the figure 5.4 we can conclude that the component performs well on Per-

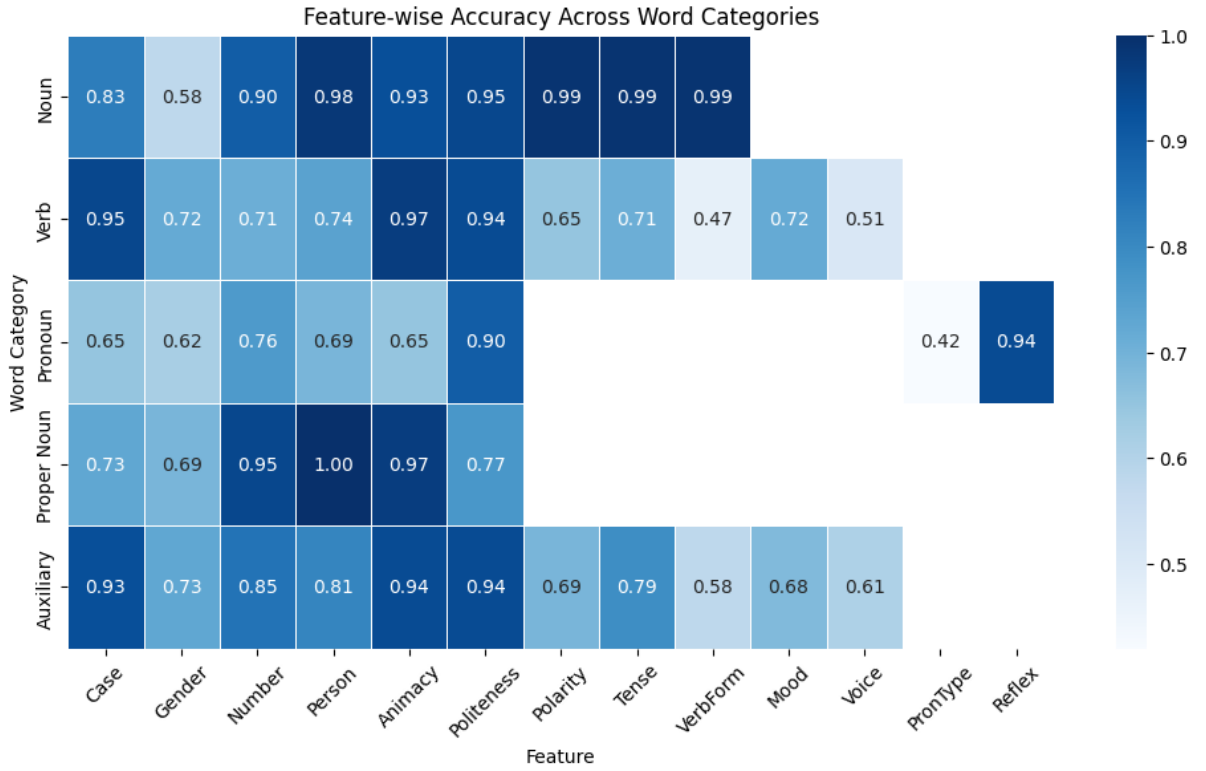

Figure 5.4: Heatmap to display feature-wise accuracy for each word category.(Missing values are left blank for clarity)

son, Politeness, Animacy, and Tense across multiple word categories. Pronouns and Verbs

Feature	Noun	Verb	Pronoun	Proper Noun	Auxiliary
Case	0.83	0.95	0.65	0.73	0.93
Gender	0.58	0.72	0.62	0.69	0.73
Number	0.90	0.71	0.76	0.95	0.85
Person	0.98	0.74	0.69	1.00	0.81
Animacy	0.93	0.97	0.65	0.97	0.94
Politeness	0.95	0.94	0.90	0.77	0.94
Polarity	0.99	0.65	-	-	0.69
Tense	0.99	0.71	-	-	0.79
VerbForm	0.99	0.47	-	-	0.58
Mood	-	0.72	-	-	0.68
Voice	-	0.51	-	-	0.61
PronType	-	-	0.42	-	-
Reflex	-	-	0.94	-	-

Table 5.8: Grammatical feature prediction per-tag accuracy

exhibit lower accuracy in specific features, like PronType and VerbForm. Gender classification is weaker for nouns and pronouns, possibly due to data imbalance or morphological complexity in Tamil. And it is clear that Proper nouns are easier to classify, especially for Number and Person features because unlike common nouns and verbs, proper nouns do not inflect for different numbers or persons in Tamil.e.g. "சென்னை" (Chennai) remains the same regardless of singular/plural usage, "ராமன்" (Raman) does not change based on grammatical number or person.

5.2.5 Error analysis

Error analysis for the grammatical feature prediction is done in order to understand the limitations and strength of the component. While overall accuracy for all the word categories achieves nearly 80% but still we have to manually inspect the predictions where the prediction has gone wrong and the root cause for that.

Table 5.9 presents a subset of unseen words where the component's predictions were incorrect compared to the ground truth.

Word	Similar Word (based to cosine similarity)	Predicted Features	True Features	Accuracy	Identified Error	Reason
நிலையங்களுக்குத் (Noun)	செயலுக்குத்	Case=Dat, Gender=Neut, Number=Sing, Person=3	Case=Dat, Gender=Neut, Number=Plur, Person=3	0.89	Number mismatch	Failed to identify plural suffix "கள்"
கட்சிக்கு (Noun)	எதிர்க்கட்சிக்கு	Case=Dat, Gender=Neut, Number=Sing, Person=3	Case=Dat, Gender=Neut, Number=Sing, Person=3	1.00	-	Correct prediction
சங்கடங்கள் (Noun)	வாழ்ந்தவர்கள்	Case=Nom, Gender=Com, Number=Plur, Person=3, Polite=Form, Polarity=Pos, Tense=Past, Verb-Form=Part	Case=Nom, Gender=Neut, Number=Plur, Person=3	0.44	Extra features predicted (Tense, Verb-Form, etc.)	Misclassified as verb-derived noun due to suffix "ங்கள்" being similar to participle endings
ஆகியவற்றில் (Noun)	வரலாற்றில்	Case=Loc, Gender=Neut, Number=Sing, Person=3	Case=Loc, Gender=Neut, Number=Plur, Person=3	0.89	Number mismatch	Ignored plural indicator "வைகள்" from "வைகள் + இல்"

இருந்தவர் (Noun)	தெரிந்தவர்	Case=Nom, Gender=Com, Number=Sing, Person=3, Polite=Form, Polarity=Pos, Tense=Past, VerbForm=Part	Case=Nom, Gender=Com, Number=Sing, Person=3, Polite=Form, Polarity=Pos, Tense=Pres, VerbForm=Part	0.89	Tense mismatch	Confused present participle with past participle due to similar structure
கடந்த (Verb)	கேட்க	Tense=None, Person=None, Gender=None, Number=None, Mood=None, Voice=Act, Form=None, Animacy=None, Polarity=Pos, VerbForm=Inf	Tense=None, Person=None, Gender=None, Number=None, Mood=None, Voice=None, Form=None, Animacy=None, Polarity=None, VerbForm=None, Polite=None, Case=None	0.75	VerbForm, Polarity, Voice	Verb was incorrectly tagged as infinitive; actual features missing
ஒதுக்கியது (Verb)	வெளிப்படுத்தியது	Tense=Past, Person=3, Gender=Neut, Number=Sing, Mood=Ind, Voice=Act, Polarity=Pos, VerbForm=Fin	Tense=Past, Person=3, Gender=Neut, Number=Sing, Mood=None, Voice=Act, Polarity=Pos, VerbForm=Ger, Case=Nom	0.75	Mood, VerbForm, Case	Confused finite and gerund forms; added mood and omitted case

நடத்தப் (Verb)	உயர்த்தப்	Tense=None, Person=None, Gender=None, Number=None, Mood=None, Voice=Act, Polarity=Pos, VerbForm=Inf	Same as predicted	1.00	-	Correct prediction
பெறும் (Verb)	பங்காற்றும்	Tense=Fut, Person=3, Gender=Neut, Number=Plur, Mood=Ind, Voice=Act, Polarity=Pos, VerbForm=Fin	Tense=Fut, Polarity=Pos, VerbForm=Part	0.50	Person, Gender, Number, VerbForm, Mood	Misidentified participle as finite verb; assumed unnecessary features
இருந்தது (Verb)	நிகழ்ந்தது	Tense=Past, Person=Number3, Gender=Neut, Number=Sing, Mood=Ind, Voice=Act, Polarity=Pos, VerbForm=Fin	Tense=Past, Person=3, Gender=Neut, Number=Sing	0.67	Mood, Voice, Polarity, VerbForm	Overfitted features; added unnecessary voice and mood
தமிழ்நாட்டின் (propn)	தென்னாப்பிரிக்காவின்	Case=Gen, Gender=Neut, Number=Sing, Person=3, Animacy=None, Polite=None	Same as predicted	1.00	-	Correct prediction

ரத்தன்-குமார் (propn)	குமார்	Case=Nom, Gender=Com, Number=Sing, Person=3, Polite=Form	Case=Nom, Gender=Neut, Number=Sing, Person=3, Polite=None	0.67	Gender, Polite	Assumed politeness and misgendered a name
அவசர (propn)	சென்னை-ய்	Case=Nom, Gender=Neut, Number=Sing, Person=3	Case=None, Gender=Neut, Number=Sing, Person=3	0.83	Case	Erroneously assigned case to a proper noun modifier
சசி (propn)	இட	Case=None, Gender=Neut, Number=Sing, Person=3	Case=Nom, Gender=Com, Number=Sing, Person=3, Polite=Form	0.50	Case, Gender, Polite	Failed to identify name as common gender; missed politeness
பலவற்றை (pron)	அவற்றை	Case=Acc, Gender=Neut, Number=Plur, Person=3, PronType=Prs	Case=Acc, Gender=None, Number=Sing, Person=3, PronType=Ind	0.62	Number, Gender, PronType	Plural mistaken for singular; pronoun type confusion
தன்னுடைய (pron)	என்னுடைய	Case=Gen, Gender=Com, Number=Sing, Person=1, Animacy=Anim, PronType=Prs	Same as predicted	1.00	-	Correct prediction
தனக்கு (pron)	அவர்களுக்கு	Case=Dat, Gender=Com, Number=Plur, Person=3, Animacy=Anim, PronType=Prs, Reflex=None	Case=Dat, Gender=Com, Number=Sing, Person=1, Animacy=Anim, PronType=Prs, Reflex=Yes	0.62	Number, Person, Reflex	Repeated error: plural and third person wrongly chosen

முடிந்தது (aux)	இருந்தது	Tense=Past, Person=3, Gender=Neut, Number=Sing, Mood=Ind, Voice=Act, Polarity=Pos, VerbForm=Fin	Tense=Past, Person=3, Gender=Neut, Number=Sing	0.64	VerbForm, Voice, Polarity, Mood	Added auxiliary-specific features not present in gold data
போகிறது (aux)	இருக்கிறது	Tense=Pres, Person=3, Gender=Neut, Number=Sing, Mood=Ind, Voice=Act, Polarity=Pos, VerbForm=Fin	Tense=Pres, Person=3, Gender=Neut, Number=Sing	0.64	VerbForm, Voice, Polarity, Mood	Auxiliary behavior misinterpreted as full finite verb form
கொள்வது (aux)	வருவது	Case=Nom, Tense=Fut, Person=3, Gender=Neut, Number=Sing, Polarity=Pos, Voice=Act, VerbForm=Ger	Case=Nom, Tense=Fut, Person=3, Gender=Neut, Number=Sing, Polarity=Pos, Voice=Act, VerbForm=Ger	1.00	-	Correct prediction
கொண்டனர் (aux)	பட்டனர்	Tense=Past, Person=3, Gender=Com, Number=Plur, Mood=Ind, Voice=Pass, Polarity=Pos, VerbForm=Fin, Polite=Form	Tense=Past, Person=3, Gender=Com, Number=Plur, Mood=Ind, Voice=Act, Polarity=Pos, VerbForm=Fin, Polite=Form	0.91	Voice	Passive misclassified instead of active

முயல்கி-ன்றன (aux)	படுகின்றன	Tense=Pres, Person=3, Gender=Neut, Number=Plur, Mood=Ind, Voice=Pass, Polarity=Pos, VerbForm=Fin	Tense=Pres, Person=3, Gender=Neut, Number=Plur, Mood=Ind, Voice=Act, Polarity=Pos, VerbForm=Fin	0.91	Voice	Mistook active voice for passive in auxiliary context

Table 5.9: Error analysis for Grammatical feature prediction

By manually analyzing these mispredictions, we identify the following key points:

1. Inconsistencies in the Gold Standard Dataset
    - Some words have gold labels that appear incorrect or incomplete.
    - In a few cases, the model's predicted features actually match the expected features better than the ground truth.
    - Example: Words like முடிந்தது, போகிறது — predicted features include auxiliary traits like VerbForm=Fin, Mood=Ind, and Voice=Act which are missing in the gold data but should arguably be present.
2. Partial Feature Accuracy Across Predictions
    - While the overall exact match is low, most individual morphological features (like Tense, Number, Gender) are correctly predicted.
    - This indicates strong generalization of core morphological traits, even if fine-grained features (like Voice, Polarity, VerbForm) occasionally differ.
3. Confusion Between Verbal Nouns and Regular Nouns
    - The model struggles to distinguish between verbal noun forms and common nouns.
    - Example:

- சங்கடங்கள் (difficulties) is a noun, but sometimes misclassified as a verbal noun.
- வாழ்ந்தவர்கள் (those who lived) is a verbal noun with embedded tense and voice, but might get treated like a regular noun.

- This is likely due to, Ambiguity in Tamil morphology, Overlapping surface forms, Limited or inconsistent annotation in training data.

4. Auxiliary Verbs Treated as Full Verbs

   - Words like இருந்தது, முடிந்தது, போகிறது are often auxiliary in function but treated as main verbs.
   - This creates annotation ambiguity and prediction inconsistency.
   - Fixing this would require clearer annotation guidelines for auxiliary constructions.

5. High Feature-Level Match Even When Overall Match is No

   - Even when the exact match = No, features like Gender, Number, and Tense are usually correct.

6. Polarity, Voice, and VerbForm are Most Error-Prone

   - These features are frequently mispredicted or missing in gold labels.

Chapter 6

Conclusion

Considering all the experiments and knowledge that we have gathered in this research, we can elaborate the conclusion of this study. Addressed problem of this research is to applying deep learning and embedding based techniques for morphological analysis in the Tamil language.

6.1 Conclusion about the research questions

We continued our research with three major research questions. During the research we did several experiments to identify the most suitable answers for those questions. We can summaries the answers for each and every research question.

6.1.1 What approaches are most effective for Tamil morphological analysis in low-resource settings with limited annotated corpora?

In low-resource settings, the most effective approach for Tamil morphological analysis is a hybrid method that combines deep learning and similarity-based techniques. Deep learning models, particularly Bidirectional LSTM, showed the highest accuracy (around 93%) according to table 5.3 for lemma prediction by leveraging context from both directions. For grammatical feature prediction, a similarity-based approach using word embeddings achieved over 85% according to table 5.7 accuracy across word categories like nouns, verbs, pronouns, and auxiliaries. This method reduces reliance on large annotated datasets by

using a pre-existing gold-standard dataset and embedding similarity, making it highly suitable for morphologically rich languages like Tamil with limited annotated resources.

6.1.2 How effectively can word embedding techniques predict/identify morphological features for unseen Tamil words?

Word embedding techniques, particularly FastText, have proven effective in predicting morphological features for unseen Tamil words. By leveraging subword-level representations, these embeddings capture phonetic and morphological patterns, enabling accurate feature transfer from known to unknown words. This approach addresses the out-of-vocabulary (OOV) challenge better than traditional rule-based or fully supervised methods, especially in low-resource settings. While some limitations remain, the majority of grammatical features for unseen words were predicted correctly according to 5.2.4, highlighting the potential of embedding-based methods to enhance Tamil morphological analysis with minimal annotated data.

6.1.3 What are the challenges in building a comprehensive annotated corpus for Tamil morphology, and how can they be addressed?

Building a comprehensive annotated corpus for Tamil morphology is difficult due to the language's morphological complexity, scarcity of data, and the need for expert manual annotation. Tamil's rich inflectional system means words can have multiple interpretations, often depending on context, which complicates consistent annotation. The lack of large, publicly available datasets further limits progress. To overcome these challenges, this research manually annotated word-lemma pairs and leveraged an existing gold-standard dataset for grammatical feature prediction. By combining human annotation with embedding-based similarity methods, the approach reduces the need for large-scale manual labeling while enhancing scalability and accuracy.

6.2 Conclusion about the research problem

The research addresses the challenge of developing an effective Tamil morphological analyzer in low-resource settings, where annotated corpora are scarce. By adopting a hybrid approach that combines deep learning models for lemma prediction with word embedding-based similarity methods for grammatical feature prediction, the study offers a scalable solution that minimizes reliance on extensive manual annotation. Bidirectional LSTM models demonstrated high accuracy in lemma prediction, while embedding-based methods effectively predicted grammatical features for unseen words. This approach not only improves performance across various word categories but also provides a practical and efficient framework for handling morphological complexity in Tamil, making it suitable for real-world NLP applications.

6.3 Limitations

The limitations of this work lie primarily in the manually annotated dataset used. The word-lemma pair corpus does not sufficiently cover all possible patterns of inflected and derived forms, and it is somewhat imbalanced, with a majority of entries being verbs. Additionally, the grammatical feature prediction component is limited to a subset of word categories for which proper annotations are available. Another key limitation is the absence of context-aware analysis—since this research focuses solely on word-level morphological analysis, it does not account for context-dependent variations in grammatical features, which may affect the accuracy of predictions in real-world scenarios.

6.4 Implications for further research

This research highlights the potential of combining deep learning and similarity-based approaches to perform effective morphological analysis in low-resource languages like Tamil. The success of subword-level embeddings in predicting grammatical features for unseen words suggests that embedding-based techniques can reduce dependency on large annotated corpora. However, the limitations identified open up several avenues for future research.

In future work, expanding and balancing the dataset to include more diverse in-

flectional and derivational forms across all word categories will be essential. Incorporating context-aware morphological analysis using sentence-level or sequence modeling approaches (e.g., transformer-based models) could improve accuracy where grammatical features depend on surrounding words. Additionally, semi-supervised or active learning techniques could be explored to automatically annotate and correct ambiguous cases, reducing manual effort. Finally, extending the current system into a full morphological generator and analyzer pipeline would enhance its applicability to real-world NLP tasks such as machine translation, speech recognition, and syntactic parsing in Tamil.

Bibliography

VP Abeera, S Aparna, RU Rekha, M Anand Kumar, V Dhanalakshmi, KP Soman, and S Rajendran. Morphological analyzer for malayalam using machine learning. In *Data Engineering and Management: Second International Conference, ICDEM 2010, Tiruchirappalli, India, July 29-31, 2010. Revised Selected Papers*, pages 252–254. Springer, 2012.

David Ifeoluwa Adelani, Jade Abbott, Graham Neubig, Daniel D'souza, Julia Kreutzer, Constantine Lignos, Chester Palen-Michel, Happy Buzaaba, Shruti Rijhwani, Sebastian Ruder, et al. Masakhaner: Named entity recognition for african languages. *Transactions of the Association for Computational Linguistics*, 9:1116–1131, 2021.

M Anand Kumar, V Dhanalakshmi, RU Rekha, KP Soman, and S Rajendran. A novel data driven algorithm for tamil morphological generator. *International Journal of Computer Applications*, 975:8887, 2010a.

M Anand Kumar, V Dhanalakshmi, KP Soman, and S Rajendran. A sequence labeling approach to morphological analyzer for tamil language. *International Journal on Computer Science and Engineering*, 2(06):1944–1951, 2010b.

Khuyagbaatar Batsuren, Gábor Bella, and Fausto Giunchiglia. Morphynet: a large multilingual database of derivational and inflectional morphology. In *Proceedings of the 18th sigmorphon workshop on computational research in phonetics, phonology, and morphology*, pages 39–48, 2021.

Jatayu Baxi and Brijesh Bhatt. A bidirectional lstm-based morphological analyzer for gujarati. *Natural Language Processing*, pages 1–17.

Jatayu Baxi and Brijesh Bhatt. Recent advancements in computational morphology: A comprehensive survey. *arXiv preprint arXiv:2406.05424*, 2024.

Kenneth R Beesley. Arabic morphology using only finite-state operations. In *Computational Approaches to Semitic Languages*, 1998.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomás Mikolov. Enriching word vectors with subword information. *CoRR*, abs/1607.04606, 2016. URL `http://arxiv.org/abs/1607.04606`.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the association for computational linguistics*, 5:135–146, 2017.

Costanza Conforti, Matthias Huck, and Alexander Fraser. Neural morphological tagging of lemma sequences for machine translation. In *Proceedings of the 13th Conference of the Association for Machine Translation in the Americas (Volume 1: Research Track)*, pages 39–53, 2018.

Ryan Cotterell and Georg Heigold. Cross-lingual, character-level neural morphological tagging. *arXiv preprint arXiv:1708.09157*, 2017.

Ryan Cotterell and Hinrich Schütze. Morphological word embeddings. *arXiv preprint arXiv:1907.02423*, 2019.

Yasas D Ekanayaka, Randil Pushpananda, Viraj Welgama, and Chamila Liyanage. Applying deep learning for morphological analysis in the sinhala language. *The International Journal on Advances in ICT for Emerging Regions*, 16(2), 2023.

Gülşen Eryiğit and Eşref Adalı. An affix stripping morphological analyzer for turkish. In *Proceedings of the iasted international conference on artificial intelligence and applications*, pages 16–18, 2004.

Rinat Gilmullin, Bulat Khakimov, and Ramil Gataullin. A neural network approach to morphological disambiguation based on the lstm architecture in the national corpus of the tatar language. In *Proc. Intern. Workshop CMLS*, volume 2303, page 32, 2018.

John Goldsmith. Unsupervised learning of the morphology of a natural language. *Computational linguistics*, 27(2):153–198, 2001.

Harald Hammarström and Lars Borin. Unsupervised learning of morphology. *Computational Linguistics*, 37(2):309–350, 2011.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Itisree Jena, Sriram Chaudhury, Himani Chaudhry, and Dipti M Sharma. Developing oriya morphological analyzer using lt-toolbox. In *International Conference on Information Systems for Indian Languages*, pages 124–129. Springer, 2011.

Lauri Karttunen and Kent Wittenburg. A two-level morphological analysis of english. In *Texas Linguistic Forum Austin, Tex*, number 22, pages 217–228, 1983.

Sarveswaran Kengatharaiyer, Parameswari Krishnamurthy, and Keerthana Balasubramani. Universal dependencies dataset, 2020. Data available since UD v2.7.

Hongjin Kim and Harksoo Kim. Integrated model for morphological analysis and named entity recognition based on label attention networks in korean. *Applied Sciences*, 10 (11):3740, 2020.

Dan Kondratyuk. Cross-lingual lemmatization and morphology tagging with two-stage multilingual bert fine-tuning. In *Proceedings of the 16th workshop on computational research in phonetics, phonology, and morphology*, pages 12–18, 2019.

Kimmo Koskenniemi. Finite state morphology and information retrieval. *Natural Language Engineering*, 2(4):331–336, 1996.

Kahraman Kostas. Lstm based iot device identification, 04 2023.

Arun Kumar, V Dhanalakshmi, RU Rekha, KP Soman, S Rajendran, et al. Morphological analyzer for agglutinative languages using machine learning approaches. In *2009 International Conference on Advances in Recent Technologies in Communication and Computing*, pages 433–435. IEEE, 2009.

Matthieu Labeau, Kevin Löser, and Alexandre Allauzen. Non-lexical neural architecture for fine-grained pos tagging. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 232–237, 2015.

S Lun. A two-level morphological analysis of french. In *Texas Linguistic Forum Austin, Tex*, number 22, pages 271–278, 1983.

Sivaneasharajah Lushanthan, AR Weerasinghe, and DL Herath. Morphological analyzer and generator for tamil language. In *2014 14th International Conference on Advances in ICT for Emerging Regions (ICTer)*, pages 190–196. IEEE, 2014.

Chaitanya Malaviya, Matthew R Gormley, and Graham Neubig. Neural factor graph models for cross-lingual morphological tagging. *arXiv preprint arXiv:1805.04570*, 2018.

Deepak Kumar Malladi and Prashanth Mannem. Statistical morphological analyzer for hindi. In *Proceedings of the Sixth International Joint Conference on Natural Language Processing*, pages 1007–1011, 2013.

AG Menon, S Saravanan, R Loganathan, and K Soman. Amrita morph analyzer and generator for tamil: a rule based approach. In *Proceedings of Tamil Internet Conference*, pages 239–243, 2009.

T Mokanarangan, T Pranavan, U Megala, N Nilusija, Gihan Dias, Sanath Jayasena, and Surangika Ranathunga. Tamil morphological analyzer using support vector machines. In *International conference on applications of natural language to information systems*, pages 15–23. Springer, 2016.

Garrett Nicolai and Grzegorz Kondrak. Morphological analysis without expert annotation. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 211–216, 2017.

Kemal Oflazer. Two-level description of turkish morphology. *Literary and linguistic computing*, 9(2):137–148, 1994.

Siddhesh Pawar, Pushpak Bhattacharyya, and Partha Talukdar. Evaluating cross lingual transfer for morphological analysis: a case study of indian languages. In *Proceedings of the 20th SIGMORPHON workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 14–26, 2023.

Martin F Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

Greeshma Prabha, PV Jyothsna, KK Shahina, B Premjith, and KP Soman. A deep learning approach for part-of-speech tagging in nepali language. In *2018 international conference on advances in computing, communications and informatics (ICACCI)*, pages 1132–1136. IEEE, 2018.

Vidya Prasad, B Premjith, Chandni Chandran, KP Soman, and Prabaharan Poornachandran. Deep learning based character-level approach for morphological inflection generation. In *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*, pages 1423–1427. IEEE, 2019.

B Premjith, KP Soman, and M Anand Kumar. A deep learning approach for malayalam morphological analysis at character level. *Procedia computer science*, 132:47–54, 2018.

Loganathan Ramasamy and Zdeněk Žabokrtský. Prague dependency style treebank for Tamil. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Mehmet Uğur Doğan, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of Eighth International Conference on Language Resources and Evaluation (LREC 2012)*, pages 1888–1894, İstanbul, Turkey, 2012. ISBN 978-2-9517408-7-7. URL `http://www.lrec-conf.org/proceedings/lrec2012/summaries/456.html`.

K. Sarveswaran. Tamil-verbs.

Kengatharaiyer Sarveswaran, Gihan Dias, and Miriam Butt. Thamizhi morph: A morphological parser for the tamil language. *Machine Translation*, 35(1):37–70, 2021.

Miikka Silfverberg and Mans Hulden. Initial experiments in data-driven morphological analysis for finnish. In *Proceedings of the fourth international workshop on computational linguistics of Uralic languages*, pages 98–105, 2018.

Miikka Silfverberg and Francis Tyers. Data-driven morphological analysis for uralic languages. In *Proceedings of the fifth international workshop on computational linguistics for Uralic languages*, pages 1–14, 2019.

A Implementation of the deep learning model for lemma prediction

```
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Embedding, Bidirectional, LSTM,
    Dense, Activation
from tensorflow.keras import callbacks

# Define the model
model = Sequential()
model.add(Input(shape=(max_len,)))
model.add(Embedding(
    input_dim=max_label,
    output_dim=embedding_size,
    mask_zero=False
))
model.add(Bidirectional(
    LSTM(hidden_size, return_sequences=True)
))
model.add(Dense(max_label))
model.add(Activation(activation_function))
model.compile(
    loss='categorical_crossentropy',
    optimizer=optimizer,
    metrics=['accuracy']
)
model.summary()

# Train the model
earlystopping = callbacks.EarlyStopping(monitor="val_loss",
                                        mode="min",
                                        patience=5,
                                        restore_best_weights=True)

history = model.fit(X_train, y_train,
                    batch_size=32,
```

```
                epochs=30,
                validation_data=(X_test, y_test),
                callbacks=[earlystopping])

# Evaluate the predictions
score = model.evaluate(
    X_test,
    y_test,
    batch_size=batch_size
)

history_df = pd.DataFrame(history.history)
history_df.loc[:, ['loss', 'val_loss']].plot()
history_df.loc[:, ['accuracy', 'val_accuracy']].plot()
plt.show()

print('Deep learning model: Bidirectional LSTM')
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Listing 1: Deep Learning Model Training using Bidirectional LSTM

B Implementation of hyper parameter tuning for lemma prediction model

```
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Embedding, Bidirectional, GRU,
    Dense, Activation
from tensorflow.keras import callbacks

resultsList = []  # Contains all the results
tempResultsList = []

# Perform hyperparameter tuning
for i in range(len(combinations)):
    # Define the model
```

```
    model = Sequential()
    model.add(Input(shape=(max_len,)))
    model.add(Embedding(
        input_dim=max_label,
        output_dim=combinations[i][0],
        mask_zero=False
    ))
    model.add(Bidirectional(
        GRU(combinations[i][1], return_sequences=True)
    ))
    model.add(Dense(max_label))
    model.add(Activation(combinations[i][2]))
    model.compile(
        loss='categorical_crossentropy',
        optimizer=combinations[i][3],
        metrics=['accuracy']
    )
    model.summary()

    # Train the model
    earlystopping = callbacks.EarlyStopping(monitor="val_loss",
                                            mode="min",
                                            patience=5,
                                            restore_best_weights=True)

    history = model.fit(X_train, y_train,
                    batch_size=combinations[i][4],
                    epochs=100,
                    validation_data=(X_test, y_test),
                    callbacks=[earlystopping])

    # Evaluate the predictions
    score = model.evaluate(
        X_test,
        y_test,
        batch_size=combinations[i][4]
    )

    history_df = pd.DataFrame(history.history)
```

```
    history_df.loc[:, ['loss', 'val_loss']].plot()
    history_df.loc[:, ['accuracy', 'val_accuracy']].plot()
    plt.show()

    # Store results
    tempResultsList = [
        combinations[i][0],  # Embedding size
        combinations[i][1],  # GRU neurons
        combinations[i][2],  # Activation function
        combinations[i][3],  # Optimizer
        combinations[i][4],  # Batch size
        score[1]             # Accuracy
    ]
    print(tempResultsList, '\n\n')
    resultsList.append(tempResultsList)

    # Clear temporary results
    tempResultsList = []
```

Listing 2: Hyperparameter Tuning for Lemma Prediction

C Implementation of the Lemma prediction process

```
import numpy as np
from tensorflow.keras.preprocessing.sequence import pad_sequences

def decode_lemma(encoded_lemma, ind2char):
    decoded_chars = [ind2char[index] for index in encoded_lemma if index > 0 and index in ind2char]
    return ''.join(decoded_chars)

def preprocess_new_word(word, char2ind, max_len):
    sequence = [char2ind[char] for char in word if char != '\n']
    padded_sequence = pad_sequences([sequence], maxlen=max_len, padding='pre')
    return padded_sequence

def predict_new_word(word, model, char2ind, ind2char, max_len):
```

```
    processed_word = preprocess_new_word(word, char2ind, max_len)
    predictions = model.predict(processed_word)
    predicted_indices = [np.argmax(char_vector) for char_vector in
predictions[0]]
    predicted_lemma = decode_lemma(predicted_indices, ind2char)
    return predicted_lemma
```

Listing 3: Lemma Prediction and Decoding

D Implementation of the hyper parameter tuning process of grammatical prediction component

```
from collections import defaultdict
import gensim
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.preprocessing import OneHotEncoder
import numpy as np

def calculate_accuracy(predicted_features, true_features):
    correct_count = sum(1 for k, v in true_features.items() if
predicted_features.get(k) == v)
    total_count = len(true_features)
    return correct_count / total_count if total_count > 0 else 0

def predict_grammatical_features(new_word, suffix_length, suffix_weight,
word_embeddings=[]):
    new_word_embedding = suffix_weighted_embedding(new_word, model,
suffix_length, suffix_weight) if new_word in model.wv else None
    if new_word_embedding is None:
        return None, None

    similarities = []
    for word in corpus:
        embedding, feature_vector = word_embeddings[word]
        similarity = calculate_embedding_similarity(new_word_embedding,
embedding)
        similarities.append((word, similarity))

```

```
    most_similar_word = max(similarities, key=lambda x: x[1])[0]
    predicted_features = morph_analysis[most_similar_word]
    return predicted_features, most_similar_word

def suffix_weighted_embedding(word, model, suffix_length, suffix_weight):
    if len(word) > suffix_length:
        root_part = word[:-suffix_length]
        suffix_part = word[-suffix_length:]
    else:
        root_part = word
        suffix_part = word

    root_vec = model.wv[root_part]
    suffix_vec = model.wv[suffix_part]

    combined_vec = (1 - suffix_weight) * root_vec + suffix_weight *
    suffix_vec
    return combined_vec

def tune_parameters(corpus, morph_analysis, test_words, suffix_lengths,
    suffix_weights):
    best_params = None
    best_accuracy = 0

    for suffix_length in suffix_lengths:
        for suffix_weight in suffix_weights:
            word_embeddings = {}
            for word in corpus:
                word_embedding = suffix_weighted_embedding(word, model,
    suffix_length, suffix_weight)
                feature_vector = encoded_features[corpus.index(word)]
                word_embeddings[word] = (word_embedding, feature_vector)

            print(f"Tuning: Suffix Length={suffix_length}, Suffix Weight={
    suffix_weight}")
            total_accuracy = 0
            total_words = 0

            for word in test_words:
```

```
                true_features = test_words[word]
                predicted_features, _ = predict_grammatical_features(word, suffix_length, suffix_weight, word_embeddings)

                if predicted_features:
                    accuracy = calculate_accuracy(predicted_features, true_features)
                    total_accuracy += accuracy
                    total_words += 1

            average_accuracy = total_accuracy / total_words if total_words > 0 else 0
            print(f"Accuracy: {average_accuracy:.2f}\n")

            if average_accuracy > best_accuracy:
                best_accuracy = average_accuracy
                best_params = (suffix_length, suffix_weight)

    print(f"Best Parameters: Suffix Length={best_params[0]}, Suffix Weight={best_params[1]}")
    print(f"Best Accuracy: {best_accuracy:.2f}")
    return best_params, best_accuracy
```

Listing 4: Hyperparameter Tuning for Suffix-Weighted Embeddings

E Implementation of the grammatical feature prediction

```
from collections import defaultdict
import gensim
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.preprocessing import OneHotEncoder
import numpy as np

feature_keys_verb = ["Tense", "Person", "Gender", "Number", "Mood", "Voice", "Form", "Animacy", "Polarity", "VerbForm", "Polite", "Case"]
feature_keys_noun = ["Case", "Gender", "Number", "Person", "Number", "Animacy", "Polite", "Polarity", "Tense", "VerbForm"]
```

```
feature_keys_pronouns = ["Case", "Gender", "Number", "Person", "Animacy", "Polite", "PronType", "Reflex"]
feature_keys_proper_noun = ["Case", "Gender", "Number", "Person", "Animacy", "Polite"]
feature_keys_auxiliary = ["Case", "Gender", "Number", "Person", "Animacy", "Polite", "Polarity", "Voice", "VerbForm", "Tense", "Mood"]
word_not_in_model = []

def load_morphological_data(filename):
    corpus = []
    morph_analysis = {}

    with open(filename, 'r', encoding='utf-8') as file:
        for line in file:
            parts = line.strip().split()
            word = parts[0]
            features_str = parts[2]
            features = {key: None for key in feature_keys}

            if features_str != "_":
                for feature_pair in features_str.split('|'):
                    key, value = feature_pair.split('=')
                    if key not in feature_keys:
                        print(key)

            parsed_features = features if features_str == "_" else dict(feature.split('=') for feature in features_str.split('|'))
            for key, value in parsed_features.items():
                features[key] = value

            corpus.append(word)
            morph_analysis[word] = features

    return corpus, morph_analysis

def suffix_weighted_embedding(word, model, suffix_length, suffix_weight):
    if len(word) > suffix_length:
        root_part = word[:-suffix_length]
        suffix_part = word[-suffix_length:]
```

```
    else:
        root_part = word
        suffix_part = word

    root_vec = model.wv[root_part]
    suffix_vec = model.wv[suffix_part]

    combined_vec = (1 - suffix_weight) * root_vec + suffix_weight *
    suffix_vec
    return combined_vec

def calculate_embedding_similarity(embedding1, embedding2):
    embedding_similarity = cosine_similarity([embedding1], [embedding2])
    [0][0]
    return embedding_similarity

def predict_grammatical_features(new_word, suffix_length, suffix_weight,
    word_embeddings):
    new_word_embedding = suffix_weighted_embedding(new_word, model,
    suffix_length, suffix_weight) if new_word in model.wv else None
    if new_word_embedding is None:
        return None, None

    similarities = []
    for word in corpus:
        embedding, feature_vector = word_embeddings[word]
        similarity = calculate_embedding_similarity(new_word_embedding,
    embedding)
        similarities.append((word, similarity))

    most_similar_word = max(similarities, key=lambda x: x[1])[0]
    predicted_features = morph_analysis[most_similar_word]
    return predicted_features, most_similar_word
```

Listing 5: Suffix-Weighted Embedding and Grammatical Feature Prediction