TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

A

PROJECT REPORT

ON

NEPALI LANGUAGE AUTO-COMPLETION TOOL USING LSTM

SUBMITTED BY:

SATYASA KHADKA (PUL077BCT078)

SHUVANGI ADHIKARI (PUL077BCT080)

SUDIP TIWARI (PUL077BCT085)

SUBMITTED TO:

DEPARTMENT OF ELECTRONICS & COMPUTER ENGINEERING

March, 2024

Copyright

The author has agreed that the Library, Department of Electronics and Computer Engineering, Pulchowk Campus, and Institute of Engineering may make this report freely available for inspection. Moreover, the author has agreed that permission for extensive copying of this project report for scholarly purposes may be granted by the department head wherein the project report was conducted. It is understood that recognition will be given to the author of this report and the Department of Electronics and Computer Engineering, Pulchowk Campus, Institute of Engineering for any use of the material of this project report. Copying publication or the other use of this report for financial gain without the approval of to the Department of Electronics and Computer Engineering, Pulchowk Campus, Institute of Engineering, and the author's written permission is prohibited.
Request for permission to copy or to make any other use of the material in this report in whole or in part should be addressed to:

Head
Department of Electronics and Computer Engineering
Pulchowk Campus, Institute of Engineering, TU
Lalitpur, Nepal.

Acknowledgments

First and foremost, we would like to express our sincere gratitude towards Department of Electronics and Computer Engineering, Pulchowk Campus, Institute of Engineering, including the Head of the Department, Deputy Head of the Department, the members of the Project Management Team, incorporating IC Chair Asst. Prof. Santosh Giri and Asst. Prof. Bibha Sthapit and all the faculty members for efforts, constant guidance and helpful encouragement. We would also like to thank our Cluster Head, Dr. Aman Shakya, for his guidance, and support.

We would like to thank the Department of Electronics and Computer Engineering, for providing us opportunity of collaborative undertaking which has helped us to implement the knowledge as the Minor Project for Third Year and developing project of our own which will greatly enhance our knowledge and provide us a new experience of teamwork.

We are also grateful towards our respected seniors who have helped us with their knowledge, experience and suggestions. We would also like to thank all of our friends who have directly and indirectly helped us in doing this project. Last but not the least, we place a deep sense of appreciation to our family members who have been constant source of inspiration for us.

Any kind of suggestion or criticism will be highly appreciated and acknowledged.

Authors:
Satyasa Khadka
Shuvangi Adhikari
Sudip Tiwari

Abstract

This project introduces an LSTM-based Nepali Language Auto-Completion Tool, designed to bridge the gap in Nepali representation within Natural Language Processing (NLP) tools. Utilizing publicly available datasets, the project aims to navigate the intricate linguistic nuances inherent in the Nepali language. Its core objectives encompass evaluating the performance of different LSTM architectures across various categories of news datasets, subsequently identifying the optimal model for an Editorial category Auto-Completion tool. The overarching goal is to enhance typing efficiency and minimize errors by seamlessly integrating the LSTM model into an intuitive typing tool. Methodologically, the project emphasizes the utilization of diverse and meticulously curated datasets, ensuring rigorous evaluation of the model's performance, and seamless integration into existing tools. By prioritizing these methodologies, the project aims to deliver a robust and effective solution that meets the evolving needs of Nepali language processing. Ultimately, this project serves as both a research endeavor within the dataset made public by the NepBERTa paper and as a pioneering initiative in empowering the Nepali language within NLP tools. By merging technical expertise with user-centric solutions, the project strives to facilitate efficient language processing while contributing to the advancement of Nepali representation in the realm of NLP.

Keywords: *LSTM, Nepali Language, NLP, Editorial, category, dataset*

Contents

Copyright **i**

Acknowledgements **ii**

Abstract **iii**

Contents **v**

List of Figures **vi**

List of Abbreviations **vii**

1 Introduction **1**

1.1 Background 1

1.2 Problem statements 2

1.3 Objectives 2

1.4 Scope 3

2 Literature Review **4**

2.1 Related work 4

2.2 Related theory 5

2.2.1 LSTM Model 6

2.2.2 Working 6

3 Methodology **8**

4 Experimental Setup **10**

4.1 Data Selection 10

4.2 Removal of Whitespace and Non-Nepali words 10

4.3 Model Development, Training and Testing 11

4.4 Implementation as a Web Application 11

5 System design **12**

5.1 LSTM Architecture 12

5.2 Context Model 14

5.3 Use Case Diagram 15
5.4 DFD Diagram 16

6 Results & Discussion **17**
6.1 Model training Details 17
6.2 Final Result: 26
6.2.1 Homepage 27
6.2.2 Explorepage 28

7 Conclusions **29**

8 Limitations and Future enhancement **31**

References **31**

Appendices **34**

List of Figures

2.1 LSTM unit 6

3.1 Workflow 8

5.1 LSTM unit 12
5.2 Context Model 14
5.3 Use case diagram 15
5.4 DFD diagram 16

6.1 Graph Illustrating Loss from Epoch 50 to 150 for Setup 1 18
6.2 Graph Illustrating Loss from Epoch 30 to 60 for Setup 2 19
6.3 Graph Illustrating Loss for Setup 3 20
6.4 Graph Illustrating Loss from Epoch 0 to 10 for Setup 5 22
6.5 Graph Illustrating Loss for Setup 7 24
6.6 Graph Illustrating Loss for Setup 9 26
6.7 Model Prediction 26
6.8 Home page 27
6.9 Main page 28

8.1 Lstm model 34
8.2 Lstm model 35
8.3 Trainig loop 35
8.4 Validation loop 36
8.5 Model Prediction 36

List of Abbreviations

AI	Artificial Intelligence
BERT	Bidirectional Encoder Representations from Transformers
DFD	Data Flow Diagram
HTML	Hyper Text Markup Language
LSTM	Long Short-Term Memory
ML	Machine Learning
NLP	Natural Language Processing
RNN	Recurrent Neural Network
XLM	Cross-lingual Language Model

1. Introduction

1.1 Background

Machine learning delves into artificial intelligence, focusing on crafting and analyzing statistical algorithms capable of performing tasks without direct instructions by effectively generalizing information. Lately, there have been advancements where generative artificial neural networks have outperformed numerous prior methods in terms of performance. Machine learning methods have found applications in various fields like large language models, computer vision, speech recognition, email filtering, agriculture, and medicine. This is particularly beneficial in scenarios where developing specific algorithms for required tasks proves to be overly expensive. Natural Language Processing is the part of machine learning used to generate natural language from data input to it. It involves developing machines that can manage human language or data resembling human language, covering its written, spoken, and structural components. Unlike computational linguistics, which uses computer science to understand language principles, NLP stands apart with its emphasis on real-world uses, striving to create technology that executes practical functions instead of solely concentrating on theoretical structures.Language modeling stands as a cornerstone task within NLP. At its core, it involves predicting the word succeeding a sequence of words, aiming to identify the most probable next word based on preceding context.

Nepali is the official language of Nepal, and it is widely spoken in Nepal, India, Myanmar, and Bhutan, as well as by the Nepalese diaspora abroad. The Nepali language is spoken by an estimated 40 million people worldwide. It is based on the Devanagari script, which has thirty-six consonants (thirty-three separate consonants and three consonant combinations), thirteen vowels, and ten numbers. Apart from such alphabets, there are other half alphabets in Devanagari scripts. Nepali writing is done from left to right, and there are no capital letters in Nepali alphabets. In Nepali, we can write the same word in different ways. This makes automatic language processing more challenging for the Nepali language in comparison to other languages (**Prasain et al. 2008**[1]). To this date, multilingual models such as Multilingual BERT, XLM and XLM-RoBERTa haven't been able to achieve promising results in Nepali NLP tasks, and there does not exist any such a large-scale monolingual corpus.

Natural language processing (NLP) in Nepali language began in 2004 with the introduction of a first Nepali lexicon. The main purpose was to build a spell checker and machine

translation system. The Bhasa Sanchar project and Nepali National Corpus further developed NLP activities in Nepali. Research activities include Nepali spell checker, grammar checker, Dobhase-A machine translation system, online dictionary, Nepali text to speech, and morphological analyzer (**T. B. Shahi & Sitaula, 2021**[2]).

1.2 Problem statements

The Nepali language, spoken by a comparatively smaller community of about 40 million worldwide, has been overlooked by researchers and practitioners. Written in the Devnagari script, Nepali boasts rich semantics and a complex grammatical structure that differs significantly from widely used languages like English. In English, the sentence structure typically follows Subject-Verb-Object, while Nepali structures commonly conclude with the verb, adhering to a standard form of Subject-Object-Verb (**Bal, 2004**[3] offers a more in-depth exploration).

Nepali is considered a low-resource language (**Rajan and Salgaonkar, 2022; Basu and Majumder, 2020**[4]), receiving minimal attention within the realm of Natural Language Processing (NLP). Despite the advancements made in NLP for English, the Nepali NLP domain has seen limited progress due to insufficient pre-training data, standardization of resources, and computational limitations.

The absence of an adequate tool for Nepali language typing has significantly impacted user productivity. This gap has led to a substantial decline in Nepali language usage, affecting typing practices across news articles, governmental office applications, and content creation endeavors.

1.3 Objectives

While language-specific models have excelled over multilingual models in several languages, the absence of Nepali language representation in such platforms has motivated us to address this gap. Consequently, our aim is to contribute to the Nepali language dataset and develop a model that effectively bridges this disparity. The primary objectives of our project includes:

1. To study LSTM architecture and create a model for experimental analysis, utilizing a self-collected Nepali dataset comprising thesis papers and news articles.

2. To apply the LSTM architecture to the publicly available dataset curated by **Timilsina et al.**[5] aiming to assess our methodology within the Nepali Language Understanding Evaluation (Nep-gLUE) Benchmark.

3. To create an assistive tool to enhance user typing efficiency by reducing keystrokes and optimizing typing time, particularly beneficial for both native and non-native language

users.

4. To improve typing accuracy by minimizing spelling errors and facilitate language learning for non-native users through suggested and accurate word recommendations, fostering vocabulary expansion with the creation of an assistive tool.

5. To acquire practical experience in contributing to Nepali language research methodologies, and assistive tool development thereby establishing a foundational milestone in our ongoing pursuit of advancements in AI/ML within the foreseeable future.

1.4 Scope

The project's focal point involves the implementation and refinement of an LSTM-based model dedicated to next-word prediction in the Nepali language. Emphasizing a diverse dataset, our efforts consider the intricate nuances and complexities inherent in Nepali grammar and vocabulary.

While previous endeavors on our proposed dataset have centered on the BERT architecture, employing 110 million parameters resulting in an accuracy of 93.55, our project diverges from aiming to compete with established models and benchmark data. Instead, it constitutes a research endeavor examining the performance of a comparatively simpler LSTM model, equipped with fewer parameters and computational demands, on the same dataset. This exploration aims to provide insight into how the LSTM architecture performs within the dataset published by **Timilsina et al.**[5]

By minimizing spelling errors and optimizing typing time, this tool will cater to diverse tasks such as news article composition and Nepali letter typing in governmental settings, among others.

2. Literature Review

Natural Language Processing is a methodical and crucial process aimed at creating coherent text that humans can comprehend. To generate this text, information is gathered from various sources or provided as input by users. In the past, Natural Language Processing (NLP) techniques relied on shallow machine learning models that incorporated manual feature engineering, consuming a significant amount of time. There has been a drastic change in the field of NLP over the past few years.(**Perera and Nand ,2017**[6]). With the rising prominence of word embeddings, neural networks have surpassed traditional machine learning models, showcasing notably enhanced success in language tasks.

2.1 Related work

It has been studied that Interactive keyboard uses the sequence of past keystrokes to predict the most likely succeeding keystroke.(**Witten, 1992**[7]). This approach of keystrokes was determined statistically. After that, almost all of the predictive models developed.These models helped us to increase writing fluency and save time.

Although English is the language being used worldwide, Nepali is the language used by the majority of the citizens for communication. A greater part of the population has little or no knowledge of English language. Hence, a machine learning model which generates text in the Nepali language would help such citizens to connect with the world digitally.The current layouts of Nepali keyboard lack scientific optimization and disregard essential factors like key frequency distribution, hand alternation, and other crucial elements. Consequently, they impose excessive and uneven strain on fingers, potentially leading to various long-term adverse effects. Prajapati, Shrestha, and Jha's paper examines how the traditional Nepali Unicode keyboard layout demands significant typing effort and diminishes typing speed.**(Prajapati, Shrestha, and Jha,2008**[8])

Several scholars have documented their research findings on next-word prediction, presenting them in published journals. Each researcher employed distinct models in their predictive frameworks. Consequently, the outcomes varied significantly between these studies, showcasing the diversity in approaches and methodologies utilized for predicting subsequent words. This diversity underscores the importance of exploring multiple models and methodologies within this domain. Among these,"Next Words Prediction Using Recurrent Neural Networks" (**Ambulgekar et al ,2021**[9]) and "Pre Training Federated Text Models for Next Word Prediction" ((**Stremmel & Singh,2021)**[10]) stand out.

In the domain of Next Word Prediction in the Nepali language employing LSTM models, our research revealed that the work of Nepali Language Auto-Completion Using LSTM is a nobel work, and no prior work has been done in this field before. The closest parallel endeavor we encountered two close works: "Next Word Prediction in Hindi Using Deep Learning Techniques"(**Sharma et al ,2019**[11]) involved the utilization of LSTM for Next Word Prediction in Hindi Language, alongside a research paper titled "Next Word Prediction in Bodhi Language Using an LSTM-based Approach."(Kumar et al, 2023) Noteworthy is the latter study's focus on the Bodhi language, prevalent in the Tibetan and Ladakh regions. The research reported the utilization of a sequence length of 10, resulting in a loss of 0.0003 and an accuracy of 44% with a batch size of 64 across 500 epochs. The corpus size for this exploration was approximately 5000 words.

Furthermore, the study titled "Next Word Prediction in Hindi Using Deep Learning Techniques"(**Sharma et al ,2019**[11]) undertook analogous efforts. The authors trained their models on a dataset comprising 2615 sentences, with an average of 7 words per sentence, totaling approximately 18000 words. Employing LSTM and Bi-LSTM architectures, the LSTM model achieved an accuracy of 59.46% utilizing a learning rate of 0.01 over 197 epochs.

Additionally, the study "NPVec1: Word Embeddings for Nepali - Construction and Evaluation" by **Koirala and Niraula, 2021**[12] was examined, wherein the authors introduced NPVec1. This resource comprises 25 state-of-the-art Word Embeddings for Nepali, derived from a substantial corpus utilizing Glove, Word2Vec, FastText, and BERT methodologies. The pre-trained model for word embeddings provided a valuable reference point for our research endeavors.

2.2 Related theory

The application of Neural Networks gained popularity in recent times over the traditional methods due to their correctness in generating the text. Neural Networks are inspired by the functioning of the brain and are proving to be efficient models in generating accurate real-world conversational or written context format. RNN was popularly used for text generation because of its ability to process sequential data. But due to its limitation of vanishing gradients, it is being replaced by other neural networks. LSTM, and other versions of LSTM, i.e., Bi-LSTM, GRU are being popularly used nowadays for generating text in most of the Indian languages. These models are also being used for other NLP-related tasks like Query auto-completion, story generation.(**Srinagesh et al ,2021**[13])

2.2.1 LSTM Model

Long short-term memory (LSTM) network is a Recurrent Neural Network(RNN), aimed to deal with the vanishing gradient problem present in traditional RNNs. LSTMs are specialized neural networks that are well-suited for tasks involving sequences of data, thanks to their ability to capture and remember information over extended periods. LSTMs mitigate the vanishing gradient problem by allowing information to be retained or discarded based on the gates, facilitating better learning of sequential patterns. It is applicable to classification, processing and predicting data based on time series, such as in handwriting, speech recognition, machine translation,speech activity detection, robot control, video games, and healthcare.

2.2.2 Working

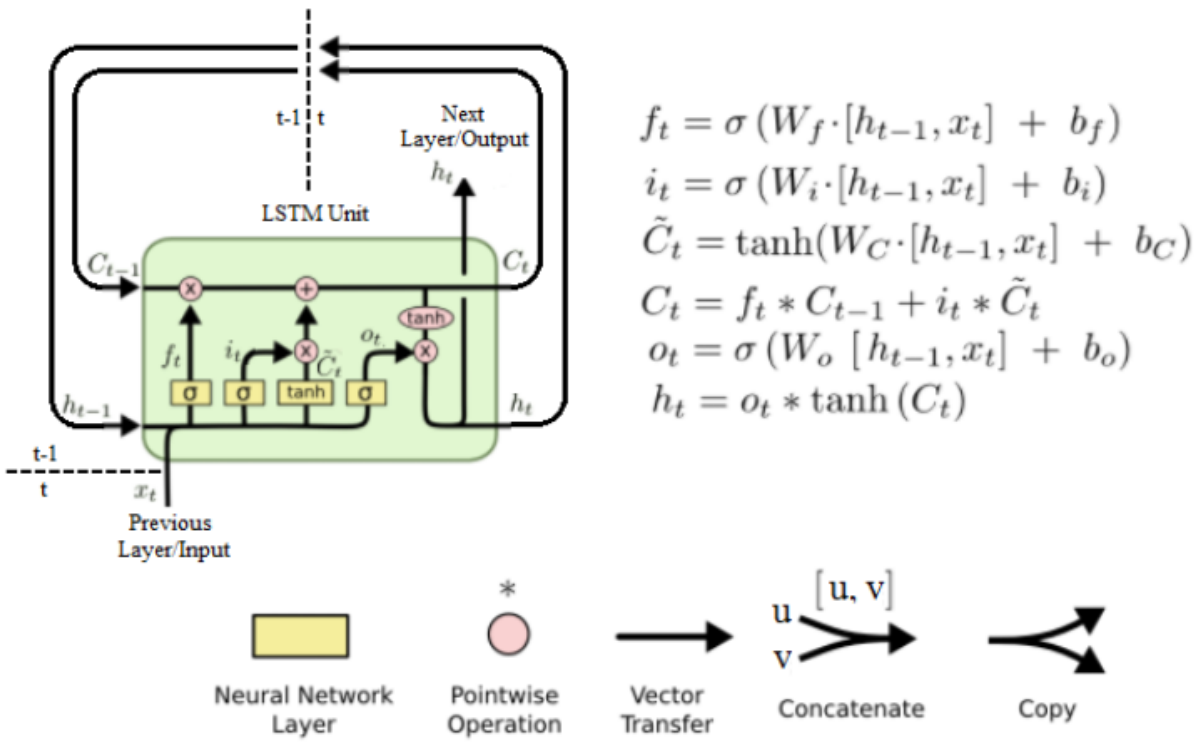

Figure 2.1: LSTM unit

The Long Short-Term Memory (LSTM) model is a type of recurrent neural network (RNN) architecture designed to address the vanishing gradient problem inherent in traditional RNNs. It achieves this by introducing gated mechanisms that selectively allow information to be retained or discarded over time. Let's delve into the workings of an LSTM unit:

Let us consider an LSTM unit at time step t, which takes as input the previous hidden state h_{t-1}, the current input x_t, and the previous cell state C_{t-1}.
The LSTM unit consists of three main gates: the forget gate, the input gate, and the output gate, each governed by sigmoid activation functions. These gates control the flow of information through the unit.

1. **Forget Gate (f_t)**: The forget gate decides which information from the previous cell state C_{t-1} to discard. It takes as input the concatenation of h_{t-1} and x_t and outputs a forget gate vector $f_t \in [0, 1]$.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

2. **Input Gate (i_t)**: The input gate determines which new information to incorporate into the cell state. It takes as input the concatenation of h_{t-1} and x_t and outputs an input gate vector $i_t \in [0, 1]$ and a candidate cell state $\tilde{C}_t$.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

3. **Update Cell State (C_t)**: The update cell state operation combines the previous cell state C_{t-1} with the information selected by the forget gate and the input gate.

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

4. **Output Gate (o_t)**: The output gate determines which information to output from the cell state. It takes as input the concatenation of h_{t-1} and x_t and outputs an output gate vector $o_t \in [0, 1]$.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

Finally, the current hidden state h_t is computed by applying the output gate to the updated cell state.

$$h_t = o_t \cdot \tanh(C_t)$$

The LSTM architecture's ability to retain information over long sequences makes it particularly effective for tasks such as next word prediction.

3. Methodology

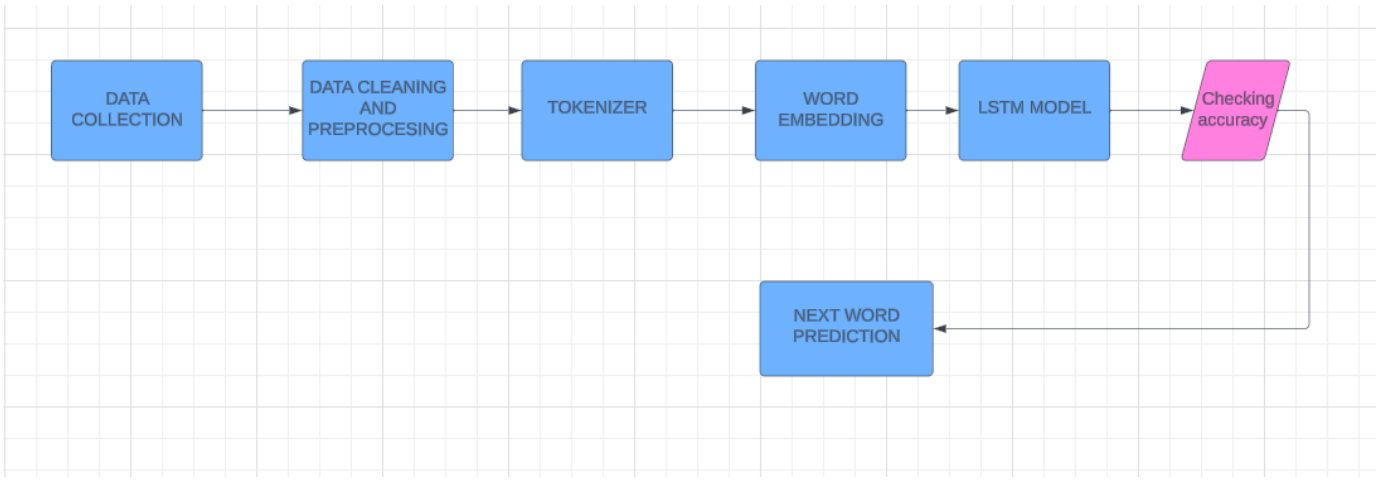

Figure 3.1: Workflow

The basic roadmap of our project was:

1. **Data collection**: A substantial dataset of text in Nepali language was collected. We curated datasets of varying sizes for different models, including news articles across different categories.

2. **Data preprocessing**: The text data underwent cleaning and preprocessing, which involved removing new line characters, Non-Nepali words and symbols etc.

3. **Tokenizer**: The text was broken down into individual words or subword units.

4. **Word embedding**: Tokens were converted into numerical representations (Word embeddings or integer encodings) suitable for input into the LSTM model.

5. **LSTM model training**: The LSTM model was trained using the training dataset. Iterations over epochs were performed, adjusting hyperparameters to minimize the loss function.

6. **Evaluation of model**: The model's performance metrics (accuracy, perplexity) were assessed using a test dataset separate from the training set.

7. **Generation of predictions**: The trained model was used to predict the next word given a sequence of tokenized words.

8. **Fine-tuning and optimization of model**: Dropout, which is a Regularization Technique was implemented to enhance the model's accuracy and prevent overfitting. Adam optimizer, which is an optimisation algorithm was used to minimize the loss function during the training. Similarly, fine-tuning of the hyperparameters was conducted to attain optimal performance and accuracy on the testing dataset, aiming to pinpoint the most effective model.

9. **Web Application Development**: Finally, a web application was developed as a tangible manifestation of our diligent efforts, experimentation, and research, effectively translating our research outcomes into practical use.

4. Experimental Setup

Our project was built on Google Colab, utilizing its extensive infrastructure, notably its GPU-enabled runtime environment. This allowed us to execute our source code more efficiently while also using the computing capabilities of GPUs to speed up our model training process. Our experimental setup included rigorous curation of datasets of differing sizes, which included news items from various categories. We improved the efficiency and scalability of our model training by using the GPU resources provided by Google Colab.

4.1 Data Selection

Our dataset was selected to be the dataset made public by **Timilsina et al.**[5] of size 12.5 GB containing approximately 0.8 Billion words with 2.75 million documents with an average of 291 words in each document. Given the computational complexities and time constraints associated with training on a large corpus of nearly 0.8 Billion words, we opted to work with a subset of data. Specifically, we curated different size of datasets for different models comprising a certain number of news articles from different categories. Each news article was represented as a distinct row within the dataset. We made various adjustments in dataset size, the categories of news articles to train our models. The details about these adjustments are also elucidated in the 'Results and Discussion' section. These adjustment was made to ensure that the model's predictions remained more general in nature, thus enhancing its overall utility and applicability.

4.2 Removal of Whitespace and Non-Nepali words

To ensure the integrity and relevance of our dataset, we embarked on a process of removing non-contextual and non-Nepali words. Initially, we addressed newline and whitespace characters by employing the following Python code:

```
subset_of_text = subset_of_text.str.replace('\n', ' ')
```

Subsequently, we encountered instances of English words such as "Source: https://website.com" and other non-Nepali terms within the dataset. To eliminate these extraneous elements, we utilized the following code, which leverages regular expressions:

```
import re
subset_of_text_string = re.sub('[A-Za-z]+', ' ', subset_of_text_string)
```

This approach effectively stripped away non-Nepali words, ensuring that our dataset remains focused and linguistically appropriate for the task at hand.

4.3 Model Development, Training and Testing

The development of the model pipeline followed a methodical approach, capitalizing on the PyTorch framework for the implementation of the LSTM-based language model. During the construction of our model pipeline, we conducted experiments with various LSTM architectures, manipulating the number of LSTM layers stacked atop one another, ranging from 2 to 4. Additionally, we explored different dimensions for the embedding and hidden units, spanning from 128 to 768, drawing inspiration from the architecture of the BERT model for experimentation purposes.

Subsequently, diverse architectures were trained across distinct datasets within our corpus, encompassing categories such as 'Politics,' 'Bichar,' 'Blog,' 'Sampadakiya,' and 'Sahityarkala.' The training procedure entailed the random partitioning of the dataset into training and validation sets in an 80:20 ratio. This process was meticulously orchestrated, incorporating a well-crafted training loop that integrated the Cross-Entropy Loss function alongside the Adam optimizer.

Hyperparameter tuning emerged as a pivotal aspect in refining the configuration of the model. Parameters including sequence length, batch size, learning rate, and epochs were meticulously adjusted to strike an optimal balance between training efficiency and generalization performance.

Additional details regarding the diverse LSTM architectures employed, along with their respective outcomes in terms of hyperparameters, will be expounded upon in the forthcoming Results and Discussion section.

4.4 Implementation as a Web Application

Following the identification of the most accurate model, our attention turns towards seamlessly incorporating this model into a real-world application—an add-on or extension. This application serves as a tangible manifestation of our diligent efforts and experimentation, translating our research into practical use. Initially, we explored the integration of our model into Google Docs through Google Apps Script. However, since programming within Google Apps Script confines us to Google's workspace, necessitating deployment of the backend on the web, we encountered time and economic constraints. Consequently, we redirected our efforts towards developing a standalone web application. For the web application, we utilized FastAPI as the backend framework and React as the frontend framework.

5. System design

5.1 LSTM Architecture

The LSTM architecture consists of one unit, the memory unit (also known as LSTM unit). The LSTM unit is made up of four feedforward neural networks. Each of these neural networks consists of an input layer and an output layer.
Three of the four feedforward neural networks are responsible for selecting information. They are the forget gate, the input gate, and the output gate. These three gates are used to perform the three typical memory management operations: the deletion of information from memory (the forget gate), the insertion of new information in memory (the input gate), and the use of information present in memory (the output gate). The fourth neural network, the candidate memory, is used to create new candidate information to be inserted into the memory.

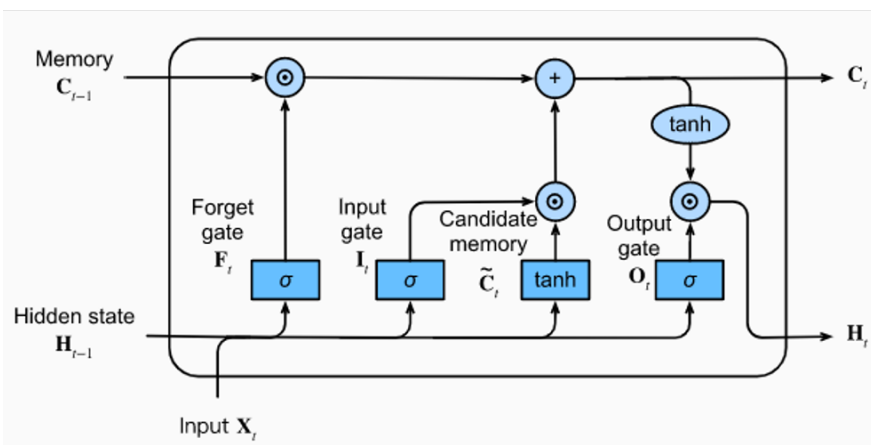

Figure 5.1: LSTM unit

An LSTM unit receives three vectors (three lists of numbers) as input. Two vectors come from the LSTM itself and were generated by the LSTM at the previous instant (instant t-1). These are the cell state (C) and the hidden state (H). The third vector comes from outside. This is the vector X (called input vector) submitted to the LSTM at instant t. Given the three input vectors (C, H, X), the LSTM regulates, through the gates, the internal flow of information and transforms the values of the cell state and hidden state vectors.

In practice, the LSTM unit uses recent past information (the short-term memory, H) and new information coming from the outside (the input vector, X) to update the long-term memory (cell state, C). Finally, it uses the long-term memory (the cell state, C) to update the short-term memory (the hidden state, H). The hidden state determined in instant t is also the output of the LSTM unit in instant t.

Working of each gates and vectors in next word prediction:

- **Forget Gate:** The forget gate looks at the previous hidden state and the current input word. It decides which information from the past should be kept or forgotten. This gate helps the LSTM remember relevant information from previous words.

- **Input Gate:** The input gate considers the current input word and the previous hidden state. It decides what new information is important to add to the memory. In the context of predicting the next word, this gate helps the LSTM capture the meaning of the current word and how it contributes to the overall context.

- **Cell state:** The combination of the forget gate and input gate updates the cell state. This cell state serves as the memory of the LSTM. It retains information about the relationships between words, allowing the model to understand the context and dependencies.

- **Output gate:** The output gate uses the updated cell state and the current input word to produce the new hidden state. The hidden state contains information about the context of the sentence in Nepali. The output gate determines what part of this information is relevant for predicting the next word. It helps the LSTM focus on the most important aspects of the context.

During training, the LSTM adapts its gates based on a dataset of Nepali sentences. It learns to recognize the patterns and relationships between words specific to the Nepali language. The gates are adjusted to make accurate predictions in the context of Nepali sentence structure.

5.2 Context Model

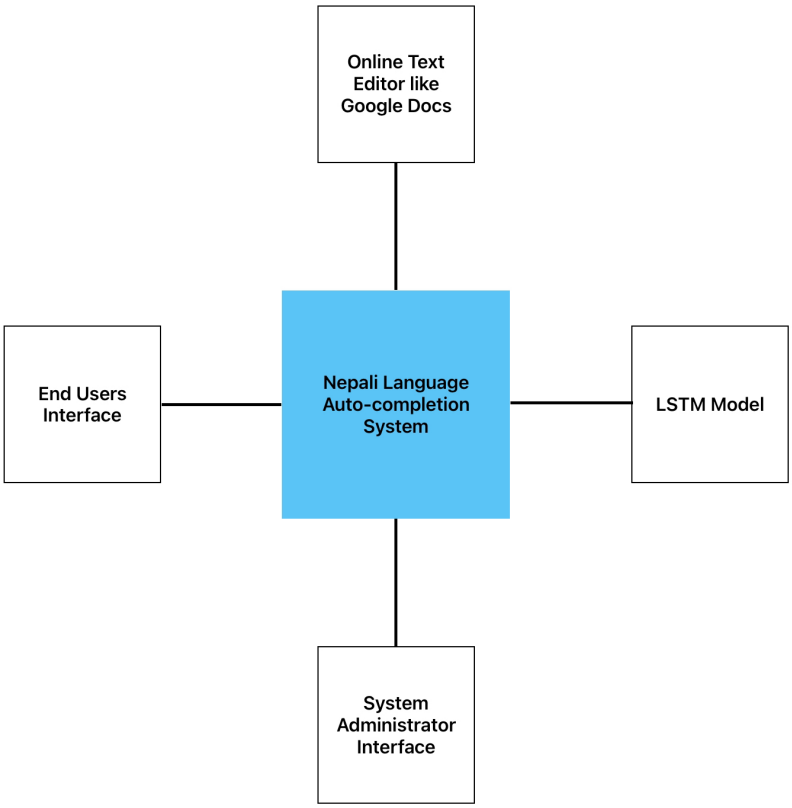

Figure 5.2: Context Model

The figure above represent the Context Model of the system. This model present the bird's-eye view of the system. The end user inputs text via the end user interface, which undergoes prediction using the LSTM model. Additionally, the system administrator can train and optimize the system model via the dedicated interface, ensuring its smooth operation.

5.3 Use Case Diagram

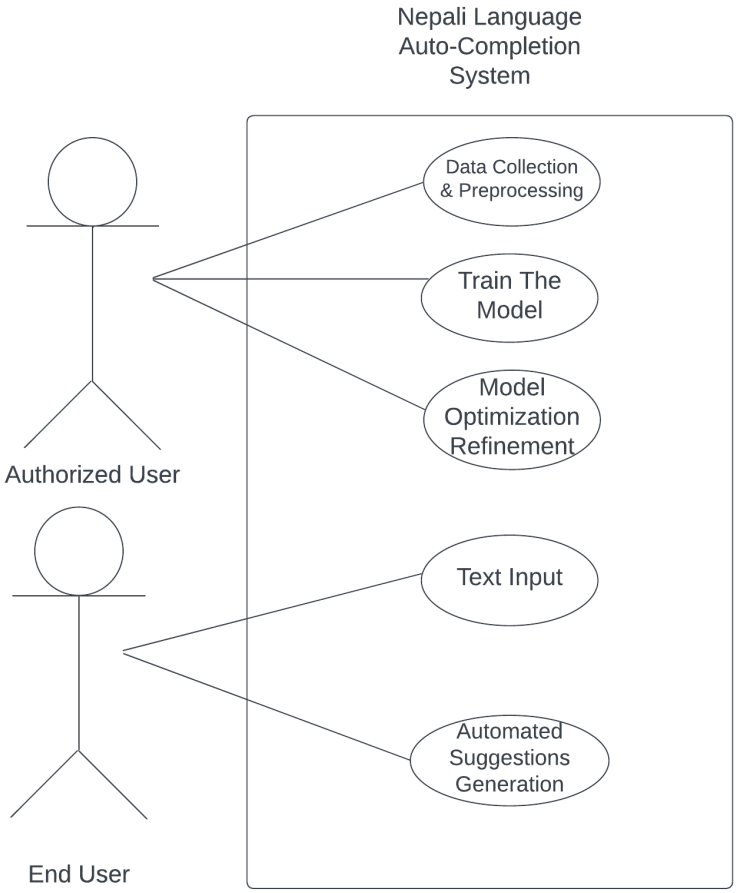

Figure 5.3: Use case diagram

The above figure showcases the Use case Diagram of the system. The diagram captures the essential interactions between users and the system. It demonstrates the functionalities available to users for text input, initiating the prediction process, and obtaining the predicted word as output.

5.4 DFD Diagram

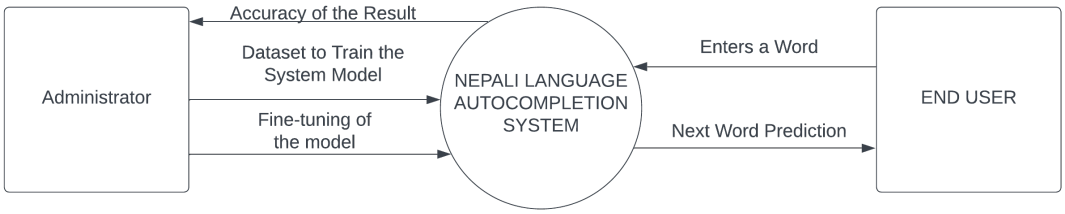

Figure 5.4: DFD diagram

The above figure represent the Level 0 DFD of our system. This diagram represents a basic flow where the user inputs text, the system processes it using the LSTM model, and then provides the predicted next word back to the user.It is a high-level overview.The internal workings or complexities of the LSTM model or the components within the Next Word Predictor system is not shown in the diagram.

6. Results & Discussion

6.1 Model training Details

1. **Model 1:** Initially we commenced the training of our model with the following hyperparameters as per the paper by **Sharma et al ,2019**[11]. We used the dataset made public by **Timilsina et al.**[5]. We incorporated news articles from the Bichar Category to train our first model. Below is the rundown of the hyperparameters and the architecture of the LSTM model:

   - Sequence Length: 10
   - Batch Size: 64
   - Learning Rate: 0.01
   - Number of Epochs: 150
   - Hidden Size (The number of features in the hidden state h) : 128
   - Embedding Dimension: 128
   - LSTM Layers: 3

   Results after 150 epochs:

   - Training Loss: 3.90
   - Validation Loss: 3.45

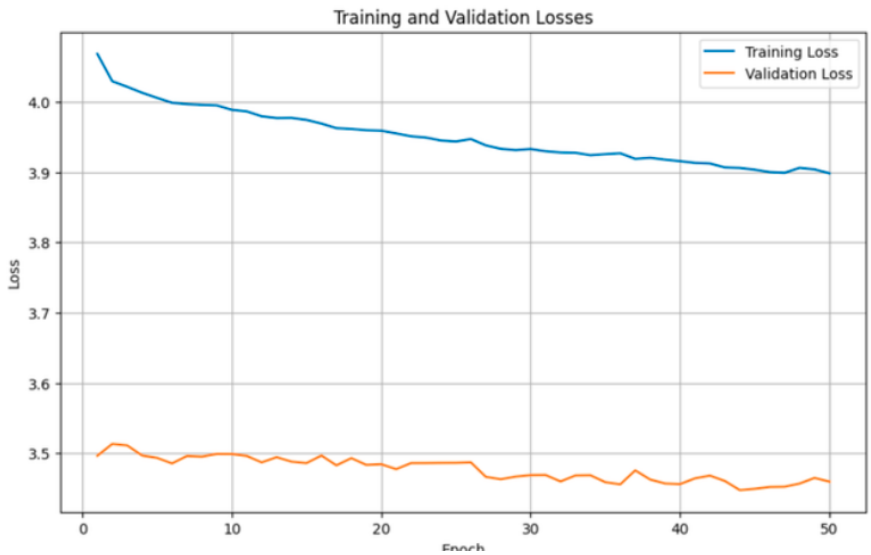

Figure 6.1: Graph Illustrating Loss from Epoch 50 to 150 for Setup 1

2. **Model 2:** Modifications were implemented on the hyperparameters, notably altering the **LSTM Layers to 2,** aiming to streamline the model's complexity and computational efficiency, thereby attaining even more optimal results. Below is the rundown of the hyperparameters and the architecture of the LSTM model:

   - Sequence Length: 10
   - Batch Size: 64
   - Learning Rate: 0.01
   - Number of Epochs: 60
   - Hidden Size (The number of features in the hidden state h) : 128
   - Embedding Dimension: 128
   - LSTM Layers: 2

   Results after 60 epochs:

   - Training Loss: 4.80
   - Validation Loss: 4.55

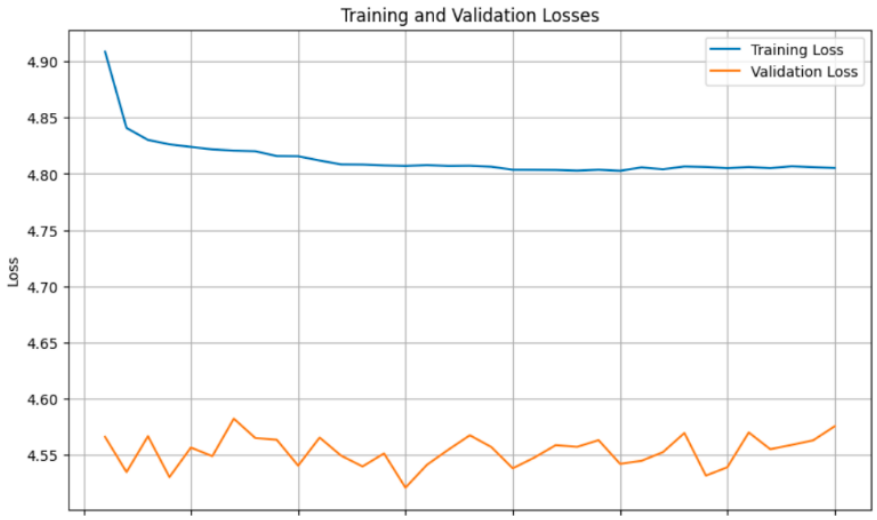

Figure 6.2: Graph Illustrating Loss from Epoch 30 to 60 for Setup 2

3. **Model 3:** In our third model, the dropout technique was applied to the embedded input instead of the LSTM layer in the Blog category dataset.The **dropout technique** was applied to the embedded input instead of the LSTM layer as a means of preventing overfitting. By randomly setting some of the embedding dimensions to zero during training, dropout helps mitigate overfitting by preventing the model from learning the training data too well, including its noise and outliers. We also decreased the LSTM layer to decrease overfitting. Below is the rundown of the hyperparameters and the architecture of the LSTM model:

   - Sequence Length: 10
   - Batch Size: 32
   - Learning Rate: 0.001
   - Number of Epochs: 50
   - Hidden Size (The number of features in the hidden state h) : 128
   - Embedding Dimension: 128
   - LSTM Layers: 1
   - dropout : 0.2

   Results after 100 epochs:

   - Training Loss: 3.8384
   - Validation Loss: 3.7066

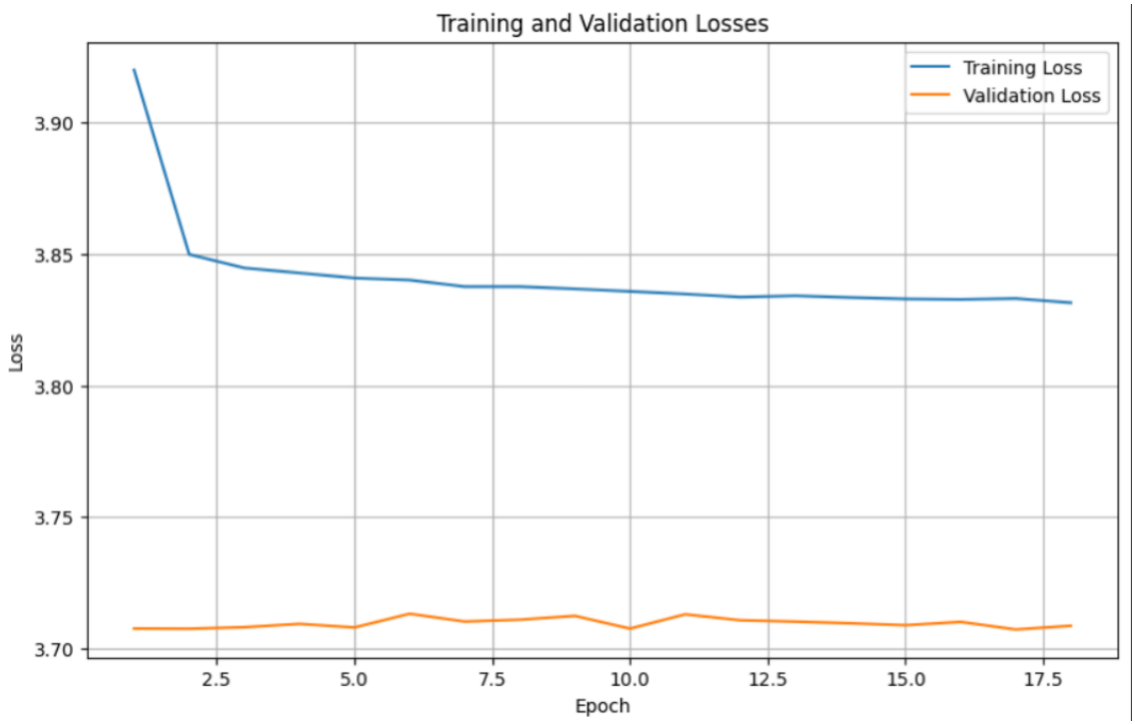

Figure 6.3: Graph Illustrating Loss for Setup 3

The application of dropout to the embedded input and the reduction of LSTM layers in our model proved effective in mitigating overfitting. The model demonstrated stable performance with a minimal validation loss of 3.7076. However, it still struggled with generalising new data.

4. **Model 4:** For our fourth model we made significant enhancements to improve its performance and generalization capability. This included increasing the **number of LSTM layers to 3**, increasing the learning rate for faster convergence, and augmenting the training dataset by incorporating news articles from the **Sampadakiya category** alongside the existing Blog category. Additionally, the introduction of perplexity measurement provided a valuable metric for evaluating the model's prediction accuracy. The dropout was added to the embed input. Below is the rundown of the hyperparameters and the architecture of the LSTM model:

   - Sequence Length: 10
   - Batch Size: 32
   - Learning Rate: 0.05
   - Number of Epochs: 50

- Hidden Size (The number of features in the hidden state h) : 128
- Embedding Dimension: 128
- LSTM Layers: 3
- dropout = 0.2

Results after 45 epochs:

- Training Loss: 4.3199
- Validation Loss: 4.28
- Perplexity: 72.2404

5. **Model 5:** Fine-tuning the hyperparameters, particularly amplifying the **Hidden Size and Embedding Dimensions,** yielded significantly enhanced performance, particularly notable when dealing with a **smaller dataset** comprising only 878 words. Below is the rundown of the hyperparameters and the architecture of the LSTM model:

   - Sequence Length: 20
   - Batch Size: 64
   - Learning Rate: 0.01
   - Number of Epochs: 10
   - Hidden Size (The number of features in the hidden state h) : 768
   - Embedding Dimension: 768
   - LSTM Layers: 2

   Results after 10 epochs:

   - Training Loss: 1.03
   - Validation Loss: 1.01
   - Perplexity: 2.74

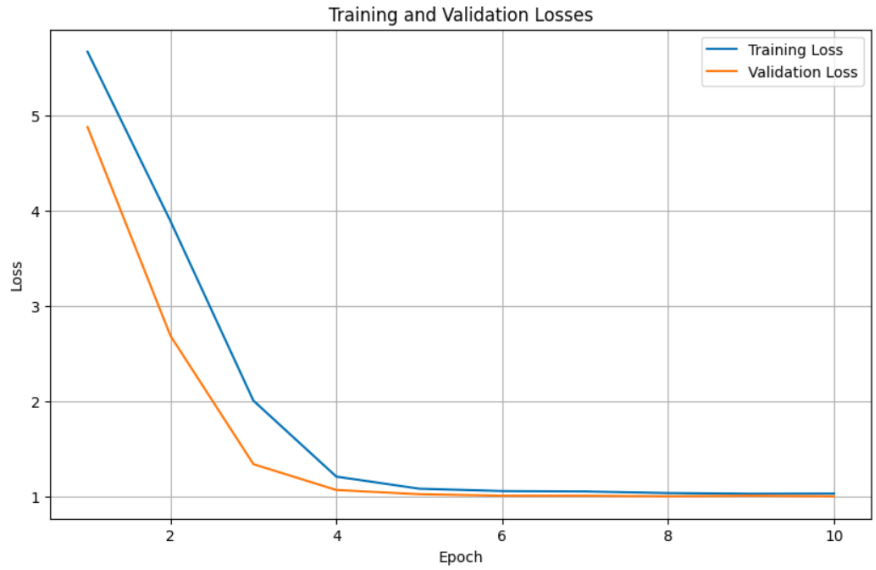

Figure 6.4: Graph Illustrating Loss from Epoch 0 to 10 for Setup 5

6. **Model 6:** In our sixth model iteration, we experimented with **reducing the batch size,** decreasing the number of LSTM layers to 3, and reducing the dimensions of both the embedding and hidden states. By reducing the batch size, our aim was to assess whether the model could receive more frequent updates to its internal weights during each iteration, enabling it to promptly adapt to evolving patterns in the data. Below is the rundown of the hyperparameters and the architecture of the LSTM model:

   - Sequence Length: 10
   - Batch Size: 32
   - Learning Rate: 0.001
   - Number of Epochs: 82
   - Hidden Size (The number of features in the hidden state h) : 256
   - Embedding Dimension: 256
   - LSTM Layers: 3

   Results after 82 epochs:

   - Training Loss: 6.98
   - Validation Loss: 6.65

By reducing the batch size, the model received more frequent updates to its internal weights during each iteration, allowing it to swiftly respond to changing patterns in the data. However, we observed a validation loss of 6.65 after 82 epochs, signaling a potential downside to this adjustment. The noticeable difference between the training and validation losses suggests that the model might have become too fixated on the specific details of the training data

7. **Model 7:** In our seventh model iteration,we increased the batchsize and reduced the Embedding and hidden size dimensions. Also we trained a model a little longer that is for 50 epochs. Below is the rundown of the hyperparameters and the architecture of the LSTM model:

   - Sequence Length: 15
   - Batch Size: 264
   - Learning Rate: 0.001
   - Number of Epochs: 50
   - Hidden Size (The number of features in the hidden state h) : 128
   - Embedding Dimension: 128
   - LSTM Layers: 4

   Results after 50 epochs:

   - Training Loss: 4.7076
   - Validation Loss: 4.4758

   The results, along with graphical representations, are presented below.

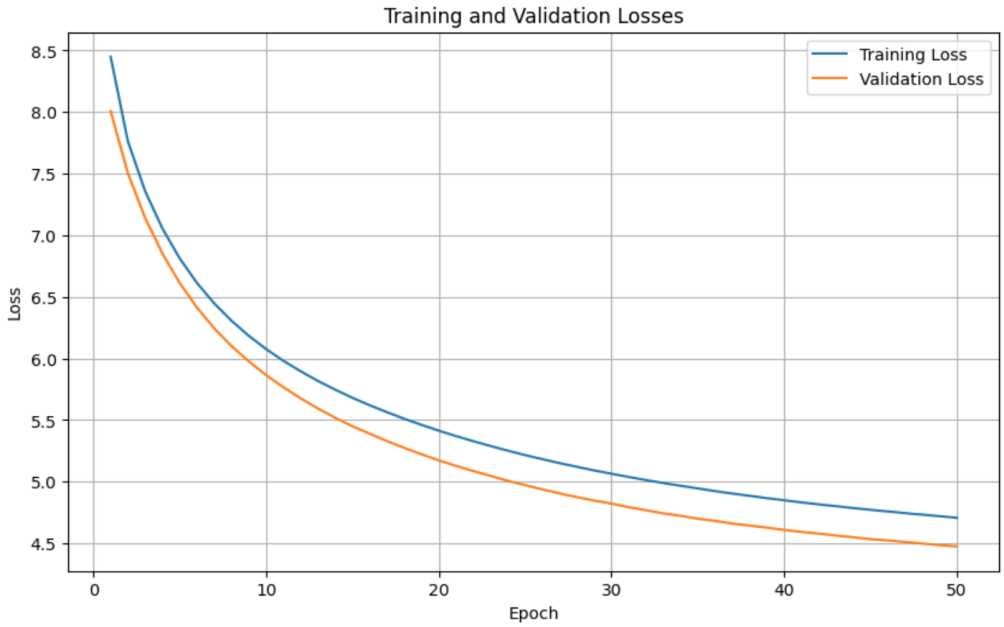

Figure 6.5: Graph Illustrating Loss for Setup 7

8. **Model 8:** In the eight model, recognizing the potential for enhanced model performance through increased capacity, we decided to make adjustments. Consequently, we increased the embedding size to 256, providing a higher-dimensional representation of the input data. Simultaneously, the LSTM size was augmented to 256, facilitating a more complex and expressive hidden state within the recurrent neural network. Furthermore, we **extended the sequence length** from its original value to 20, enabling the model to capture longer-term dependencies and contextual information in the input sequences. Below is the rundown of the hyperparameters and the architecture of the LSTM model:

   - Sequence Length: 20
   - Batch Size: 64
   - Learning Rate: 0.001
   - Number of Epochs: 100
   - Hidden Size (The number of features in the hidden state h) : 256
   - Embedding Dimension: 256
   - LSTM Layers: 3

   Results after 20 epochs:

- Training Loss: 4.3130
- Validation Loss: 4.1743
- Perplexity: 64.9971

While the initial epochs showed promising results as the validation loss was gradually increase. The model seemed to struggle capturing long term dependencies.

9. **Model 9:** In our ninth model, we narrowed down our dataset to just 5322 words from the 'Sampadakiya' category. In our previous models we had experimented with the Blog category dataset, but the sentences generated were quite intangible. Thus, we opted for the 'Sampadakiya' category only. Given the relatively modest size of our dataset, we reduced the number of training epochs to mitigate the risk of overfitting. Moreover, to enhance the model's capacity to capture nuanced features, we augmented the Embedding dimensions and hidden size to 768 each. Below is the rundown of the hyperparameters and the architecture of the LSTM model:

   - Sequence Length: 20
   - Batch Size: 64
   - Learning Rate: 0.01
   - Number of Epochs: 20
   - Hidden Size (The number of features in the hidden state h) : 768
   - Embedding Dimension: 768
   - LSTM Layers: 2

   Results after 20 epochs:

   - Training Loss: 1.5461
   - Validation Loss: 1.5265
   - Perplexity: 4.6022

   Remarkably, our model achieved an impressive accuracy of 50.008% and was able to capture long term dependencies comparatively better than the previous models. However, it's worth noting the inherent trade-off between accuracy and dataset size.

   The results, along with graphical representations, are presented below.

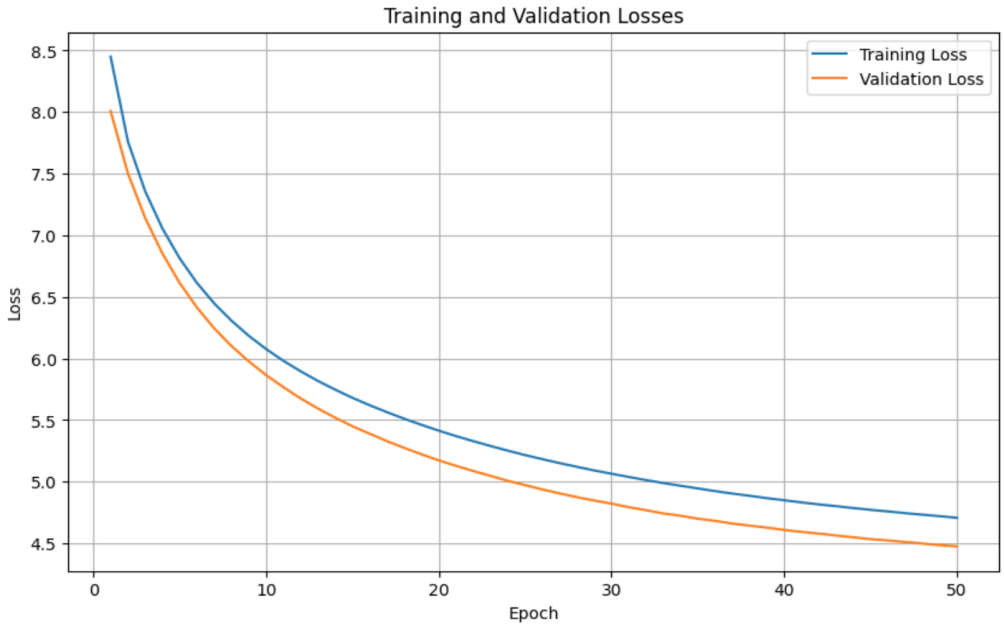

Figure 6.6: Graph Illustrating Loss for Setup 9

6.2 Final Result:

```
torch.Size([1, 1, 2172])
Input Sentence: कोभिसिल्ड खोपको पहिलो मात्रा लगाएर दोस्रो मात्रा लगाउन पर्खेका ज्येष्ठ नागरिकले अर्को मात्रा तत्काल नपाउने खुलासा स्वास्थ्य तथा जनसङ्ख्या
Predicted Next Word: मन्त्रालयको
```

Figure 6.7: Model Prediction

As our final endeavor to translate our research into a tangible product, we initially explored the development of an Add-On for Google Docs. However, due to the necessity of programming within Google Apps Script in the Google Workspace environment, this approach mandated the deployment of the backend on the web, which could potentially be accessed via the Google Apps Script source code. Consequently, we redirected our efforts towards the development of a Web Application.

This Web Application was constructed using Fast API as the Backend and React as the Frontend tool. Serving as a testament to the platform we've built, this application demonstrates our commitment to practical application. Importantly, it provides a flexible framework for integrating any future advancements or improvements in model performance seamlessly. This adaptability ensures that our solution remains relevant and capable of delivering even better real-world results in the future.

6.2.1 Homepage

Figure 6.8: Home page

Our homepage exudes simplicity and elegance, featuring a clean and intuitive interface designed to provide users with a seamless browsing experience. At the natigation bar, users are greeted by a bold and inviting 'Explore' button, beckoning them to delve deeper into our platform's offerings.

Crafted with React, our homepage boasts dynamic functionality, allowing users to effortlessly navigate to the next page with a single click. The minimalist design ensures that the focus remains squarely on the user's journey, facilitating smooth transitions and intuitive interactions.

6.2.2 Explorepage

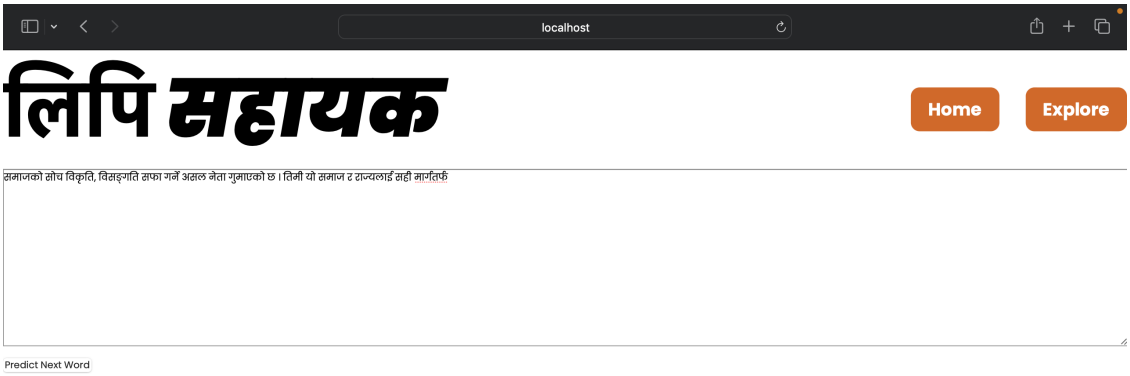

Figure 6.9: Main page

Our system simplifies the process of typing sentences by offering users a convenient solution. With just a simple click of the 'Predict Next Word' button, users can seamlessly receive predictions for the next word in their sentence. In the backend, the model operates by receiving the sentence as input, leveraging the present input context to generate predictions for the next word. The system then smoothly integrates the guessed word into the sentence's end, ensuring a fluid and uninterrupted typing experience.
This innovative feature significantly enhances typing efficiency, providing users with accurate predictions that align closely with their intended message. By anticipating the next word in real-time, our system not only streamlines the typing process but also fosters a more intuitive and productive user experience.
The users can write their thoughts as form of Blog or Editorial using our web application.

7. Conclusions

The development of the Nepali Language Auto-completion tool utilizing Long Short-Term Memory (LSTM) was initiated with the inception of data collection. Guided by advisement to refrain from working directly with data corpus, an alternative approach was adopted wherein pre-collected data from the paper titled "NepBERTa"**Timilsina et al.**[5] was utilized. This resource boasted a substantial Nepali Corpus, comprising 12.5 GB and encompassing approximately 0.8 Billion words across 2.75 million documents. The dataset exhibited a diverse array of news categories including 'Politics', 'Blog', 'Editorial', 'Sahityarkala', and 'Bichar'. Initial experimentation within the Politics category yielded results laden with overtly political connotations, diverging from the intended use case. Consequently, exploration shifted towards the Blog and Editorial categories. However, the absence of data in the anticipated format within the Blog category hindered effective model training, prompting a pivot towards the Editorial category.

Subsequent training iterations involved various subsets of the dataset, each experimented upon with distinct sets of hyperparameters. Notably, a discernible tradeoff emerged between data quantity and validation accuracy. The augmentation of data volume appeared to impede the model's capacity to discern intricate long-term patterns. This phenomenon was attributed to potential deficiencies in data collection methodologies or the nuanced inclusion of sarcastic elements within the Blog category, which inadvertently impacted the model's performance on unseen testing data. Consequently, the decision was made to focus exclusively on the "Sampadakiya/Editorial" category for training purposes. Following exhaustive iterations, detailed in the preceding "Results and Discussion" section, an optimal model with superior validation accuracy was identified. Subsequently, the integration of this model into a web application was accomplished, with Fast API serving as the backend framework complemented by React for frontend development.

The culmination of our efforts in developing the Nepali Language Auto-Completion Tool utilizing LSTM represents a pivotal milestone in translating our research endeavors into a practical, accessible solution. Tailored to assist Nepali writers and editors, this tool stands as a testament to our commitment towards enhancing language processing capabilities within the Nepali domain. Furthermore, by embracing an ethos of open-sourcing our project and accompanying documentation, we extend an invitation to prospective collaborators to join us in advancing the frontiers of knowledge within this domain.

Through collaborative efforts and shared expertise, we endeavor to continually refine and optimize the outcomes of our collective endeavors, thereby fostering tangible improvements in this vital sphere of linguistic innovation, especially for low resource language like Nepali Language.

8. Limitations and Future enhancement

Initially, our model performed impressively with a smaller dataset, boasting high accuracy. However, we soon realized that its proficiency was confined to a specific domain due to the limited scope of training data. To enhance its adaptability and broaden its capabilities, we endeavored to train the model on a larger dataset. Unfortunately, this expansion came at the cost of reduced accuracy. We encountered a tradeoff between dataset size and accuracy, prompting a reassessment of our approach. After careful consideration, we've opted to revert to training on the smaller dataset for now. This decision ensures a balance between accuracy and the model's ability to generalize across different domains. The another observed limitation is that our current model, based on LSTM (Long Short-Term Memory), has not fully mastered learning long-term dependencies in the data. The model struggled to capture relationships and context over extended sequences of words,leading to less accurate predictions in certain scenarios.There are a couple of potential reasons behind the less-than-desired performance of our model. Firstly, it's possible that the computational complexity of our LSTM architecture isn't optimal for the task at hand. Alternatively, the dataset we're currently utilizing may not be sufficiently diverse to train the model effectively. Despite all the limitation mentioned we have developed a tool that serves as a critical component in ensuring the performance of the model aligns with the objectives and facilitates prompt adjustments or interventions when necessary.
As we look to enhance the performance of our model, we're exploring the potential of adopting newer neural network architectures, such as the Transformer. This cutting-edge approach could offer significant improvements over our current LSTM-based model by better capturing long-term dependencies and contextual relationships in the data.

References

[1] Balaram Prasain. Computational analysis of nepali basic verbs (written forms). *Nepalese Linguistics*, 23:262–270, 2008.

[2] Tej Shahi and Chiranjibi Sitaula. Natural language processing for nepali text: a review. *Artificial Intelligence Review*, 55:1–29, 04 2022.

[3] Bal Krishna Bal. Structure of nepali grammar. 01 2004.

[4] Annie Rajan and Ambuja Salgaonkar. *Survey of NLP Resources in Low-Resource Languages Nepali, Sindhi and Konkani*, pages 121–132. 01 2022.

[5] Sulav Timilsina, Milan Gautam, and Binod Bhattarai. NepBERTa: Nepali language model trained in a large corpus. In Yulan He, Heng Ji, Sujian Li, Yang Liu, and Chua-Hui Chang, editors, *Proceedings of the 2nd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 12th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 273–284, Online only, November 2022. Association for Computational Linguistics.

[6] Rivindu Perera and Parma Nand. Recent advances in natural language generation: A survey and classification of the empirical literature. *Computing and Informatics*, 36(1):1–32, 2017.

[7] Ian H Witten. *The reactive keyboard*. Cambridge [England]: Cambridge University Press, 1992.

[8] Chetan Prajapati, Jwalanta Deep Shrestha, and Shishir Jha. Nepali unicode keyboard layout standarization based on genetic algorithm. 2008.

[9] Sourabh Ambulgekar, Sanket Malewadikar, Raju Garande, and Bharti Joshi. Next words prediction using recurrent neuralnetworks. In *ITM Web of Conferences*, volume 40, page 03034. EDP Sciences, 2021.

[10] Joel Stremmel and Arjun Singh. Pretraining federated text models for next word prediction. In *Advances in Information and Communication: Proceedings of the 2021 Future of Information and Communication Conference (FICC), Volume 2*, pages 477–488. Springer, 2021.

[11] Radhika Sharma, Nishtha Goel, Nishita Aggarwal, Prajyot Kaur, and Chandra Prakash. Next word prediction in hindi using deep learning techniques. In *2019 International Conference on Data Science and Engineering (ICDSE)*, pages 55–60, 2019.

[12] Pravesh Koirala and Nobal B. Niraula. NPVec1: Word embeddings for Nepali - construction and evaluation. In Anna Rogers, Iacer Calixto, Ivan Vulić, Naomi Saphra, Nora Kassner, Oana-Maria Camburu, Trapit Bansal, and Vered Shwartz, editors, *Proceedings of the 6th Workshop on Representation Learning for NLP (RepL4NLP-2021)*, pages 174–184, Online, August 2021. Association for Computational Linguistics.

[13] A Srinagesh, Ch Sudha Sree, B Prasanthi, P Rama Krishna, and P Siva Prasad. Next word prediction in telugu sentences using recurrent neural networks. *Turkish Online Journal of Qualitative Inquiry*, 12(5), 2021.

Appendices

```python
from torch import nn
class LSTMModel(nn.Module):
    def __init__(self, dataset):
        super(LSTMModel, self).__init__()
        #number of hidden units or neurons in each LSTM layer
        self.lstm_size = 768
        self.embedding_dim = 768
        # self.num_layers = 4
        self.num_layers = 2

        n_vocab = len(dataset.unique_words)
        #embedding layer is necessary to convert word into dense vector
        self.embedding = nn.Embedding(
            #how many unique words
            num_embeddings=n_vocab,
            #what dimension of vector will represent the unique words
            embedding_dim=self.embedding_dim,
        )
        self.lstm = nn.LSTM(
            input_size=self.embedding_dim,
            hidden_size=self.lstm_size,
            num_layers=self.num_layers,
            #regularization technique that helps to control overfitting
            dropout=0.2,
        )
```

Figure 8.1: Lstm model

```python
        #fully connected layer
        self.fc = nn.Linear(self.lstm_size, n_vocab)

    def forward(self, x, prev_state):
      #applies wrod embedding to input sequence x
        embed = self.embedding(x)
        output, state = self.lstm(embed, prev_state)
        # logits represent how likely each word is to come next in the sequence based on the learned patterns.
        logits = self.fc(output)

        return logits, state
    def accuracy(self, logits, targets):
      #logits return the word with the maximum probability and also its indices
      _, predicted = logits.max(2)
      correct = predicted.eq(targets)
      #.item() is used to convert it to a regular Python scalar
      accuracy = correct.sum().item() / correct.numel()
      return accuracy


    def init_state(self, sequence_length):
        return (
            torch.zeros(self.num_layers,
                        sequence_length, self.lstm_size),
            torch.zeros(self.num_layers,
                        sequence_length, self.lstm_size)
        )
```

Figure 8.2: Lstm model

```python
# Training loop
for epoch in range(num_epochs):
    model.train()
    total_loss = 0.0
    correct_train = 0
    total_train = 0

    for batch in train_loader:
        inputs, targets = batch
        # Move inputs and targets to CUDA
        inputs, targets = inputs.to(device), targets.to(device)

        optimizer.zero_grad()

        hidden = model.init_state(sequence_length)
        # Move each tensor in the tuple to CUDA
        hidden = tuple(h.to(device) for h in hidden)

        outputs, _ = model(inputs, hidden)

        loss = criterion(outputs.view(-1, len(dataset.unique_words)), targets.view(-1))
        loss.backward()

        optimizer.step()

        total_loss += loss.item()

        # Calculate accuracy for each batch
        accuracy_train = model.accuracy(outputs, targets)
        total_train += inputs.size(0) * inputs.size(1)
        correct_train += accuracy_train * inputs.size(0) * inputs.size(1)

    # Calculate average loss for the epoch
    average_loss = total_loss / len(train_loader)
    train_losses.append(average_loss)

    # Calculate average accuracy
    accuracy_train = correct_train / total_train
    train_accuracies.append(accuracy_train)

    # Print the epoch, average loss, and accuracy
    print(f"Epoch [{epoch+1}/{num_epochs}], Training Loss: {average_loss:.4f}, Training Accuracy: {accuracy_train:.4f}")
```

Figure 8.3: Trainig loop

```python
# Validation loop
model.eval()
val_loss = 0.0
correct_val = 0
total_val = 0

with torch.no_grad():
    for batch in val_loader:
        inputs, targets = batch
        # Move inputs and targets to CUDA
        inputs, targets = inputs.to(device), targets.to(device)

        hidden = model.init_state(sequence_length)
        # Move each tensor in the tuple to CUDA
        hidden = tuple(h.to(device) for h in hidden)
        outputs, _ = model(inputs, hidden)

        loss = criterion(outputs.view(-1, len(dataset.unique_words)), targets.view(-1))
        val_loss += loss.item()

        # Calculate accuracy for each batch
        accuracy_val = model.accuracy(outputs, targets)
        total_val += inputs.size(0) * inputs.size(1)
        correct_val += accuracy_val * inputs.size(0) * inputs.size(1)

# Calculate average validation loss for the epoch
average_val_loss = val_loss / len(val_loader)
val_losses.append(average_val_loss)

# Calculate perplexity
perplexity = math.exp(average_val_loss)

# Calculate average validation accuracy
accuracy_val = correct_val / total_val
val_accuracies.append(accuracy_val)

# Save the model after each epoch
if (epoch == 19 or epoch % 5 == 0):
  FULL_MODEL_PATH = f"{PATH}small-sampadak-768-epoch-{epoch}.pt"
  torch.save(model.state_dict(), FULL_MODEL_PATH)

# Print the epoch, average validation loss, and accuracy
print(f"Epoch[{epoch+1}/{num_epochs}], Validation Loss: {average_val_loss:.4f}, Validation Accuracy: {accuracy_val:.4f}, Perplexity : {perplexity:.4f}")
```

Figure 8.4: Validation loop

```
torch.Size([1, 1, 2172])
Input Sentence: कोभिसिल्ड खोपको पहिलो मात्रा लगाएर दोस्रो मात्रा लगाउन पर्खेका ज्येष्ठ नागरिकले अर्को मात्रा तत्काल नपाउने खुलासा स्वास्थ्य तथा जनसङ्ख्या
Predicted Next Word: मन्त्रालयको
```

Figure 8.5: Model Prediction