

Tribhuwan University
Institute of Engineering
Pulchowk Campus

A Final Report on:
Nepali Language Processing

Submitted By:

Nabin Da Shrestha(076bct037)
Nirajan Bekoju(076bct039)
Nishant Luitel (076bct041)

Submitted To:

Department of Electronics and Computer Engineering

Submission Date:

22nd March, 2023

Acknowledgement

We would like to express our deepest gratitude to our supervisor, Dr. Basanta Joshi, for his invaluable guidance, and support throughout this project. His expertise in natural language processing and dedication to our success have been instrumental in our ability to develop a comprehensive understanding of Nepali language processing.

We would like to express our profound appreciation to Prasid Karki, the Project Manager at Rara Labs, for his invaluable assistance and support in helping us develop this project. His extensive knowledge and expertise in the field of natural language processing have played a crucial role in our project's success. We are deeply grateful for his guidance and mentorship throughout this endeavor.

Finally, we would like to thank our friends and family for their encouragement and support throughout this project. Their unwavering belief in our abilities and dedication to our success have been a constant source of inspiration.

Thank you all for your support and guidance throughout this project.

Contents

1	Introduction to Nepali Language Model	9
1.1	Abstract	9
1.1.1	N-gram	9
1.1.2	Continuous Space	9
1.2	Problem Statement	10
1.3	Objectives	10
1.4	Expected Outcome	10
1.5	Data Source Link	11
2	Literature Review	12
2.1	A Neural Probabilistic Language Model	12
2.1.1	Introduction	12
2.1.2	NPLM Architecture	12
2.1.3	Conclusion and Results	13
2.2	Attention is all you need	14
2.2.1	Introduction	14
2.2.2	Transformer Model Architecture	14
2.2.3	Conclusion and Results	17
3	Requirement Analysis and Feasibility Study	18
3.1	Functional and Non Functional Requirements	18
3.1.1	Functional Requirements	18
3.1.2	Non Functional Requirements	19
3.2	Hardware Requirements	20
3.3	Software Requirements	20
3.4	Feasibility Study	20
3.4.1	Technical Feasibility	20
3.4.2	Economic Feasibility	21
3.4.3	Legal Feasibility	21
3.4.4	Scheduling Feasibility	21
4	System Design and Architecture	22
4.1	Use Case Diagram	22
4.2	Activity Diagram	23
5	Project Management	26
5.1	Agile Development Methodology	26

5.2	Trello	26
5.3	Discord	27
6	Probabilistic Language Model	28
6.1	Introduction	28
6.2	N-gram	28
6.3	Sequence Notation	29
6.4	N-gram Probability	29
6.4.1	Uni-gram Probability	29
6.4.2	Bi-gram Probability	29
6.4.3	Tri-gram Probability	30
6.5	Probability of a sequence	30
6.6	Approximation of Sequence Probability	31
6.7	Starting and Ending Sentences	31
6.7.1	Start of sentence symbol $\langle s \rangle$	31
6.7.2	End of sentence symbol $\langle /s \rangle$	31
6.8	Count Matrix	32
6.9	Probability Matrix	32
6.10	Generative Language Model	33
6.11	Train, Validation and Test Split	34
6.12	Language Model Evaluation	34
6.12.1	Extrinsic evaluation	34
6.12.2	Intrinsic evaluation	34
6.12.3	Perplexity	34
6.12.4	Log Perplexity	35
6.13	Vocabulary	35
6.14	Missing N-gram in training corpus	36
6.14.1	Laplacian Smoothing	36
6.14.2	K-smoothing	36
7	Sequence Modelling	37
7.1	Introduction	37
7.2	Recurrent Neural Networks	37
7.3	Vanishing Gradients with RNNs	39
7.4	LSTMs	40
7.4.1	LSTM Cell	41
7.5	Bidirectional RNNs	44
7.6	Sequence Modelling Applications	45
8	Methodology, Result and Evaluation	46
8.1	Word Embedding Visualization	46
8.2	Probabilistic Language Model	47
8.3	Transformer based language model	48
8.4	Context Based Spelling Correction	49
8.5	Backend	49
9	Sentimental Classification Model and Evaluation	50
9.1	Sentimental Data Exploration	50

9.2	Data Preprocessing	51
9.3	Train test split	51
9.4	Tokenization and One Hot Encoding	52
9.5	Sentimental Classification Model	52
9.5.1	Sentimental Classification Model Version 1	52
9.5.2	Sentimental Classification Model Version 2	54
9.5.3	Bert Model	57
10	Frontend	59
11	Conclusion	66

List of Figures

1.1	Nepali Word Cloud	10
1.2	Nepali Text Generation	11
1.3	Nepali Spelling Correction based on contextual meaning	11
2.1	Direct Architecture of Probabilistic Language Model	13
2.2	The Transformer - model architecture	14
2.3	Scaled Dot-Product Attention	15
2.4	Multi-Head Attention consists of attention layer running in parallel	16
4.1	System Use Case Diagram	22
4.2	Word Embeddings Activity Diagram	23
4.3	Sentiment Classification Activity Diagram	24
4.4	Language model Activity Diagram	24
4.5	Spelling Correction Activity Diagram	25
5.1	Agile Development Methodology	26
5.2	Trello	27
5.3	Discord	27
6.1	Language Model for Auto-complete	28
7.1	Recurrent Neural Networks	38
7.2	An unrolled recurrent neural networks	38
7.3	Short Sequence in RNNs	39
7.4	Long Sequence in RNNs	39
7.5	The repeating module in a standard RNN contains a single layer.	40
7.6	The repeating module in an LSTM contains four interacting layers.	41
7.7	LSTM Cell	41
7.8	Bidirectional RNN	44
8.1	2d plot of word embeddings vectors	46
8.2	3d plot of word embeddings vectors	47
8.3	Next word prediction using Probabilistic LM	48
8.4	Next word prediction using Transformer based LM	48
9.1	Sentimental Classification Dataset	50
9.2	Kaggle Dataset Label Count plot	51
9.3	Combined Dataset Label Count plot	51

9.4	Length Series Count Plot : Length 10 seems to be appropriate for padding length from the plot.	52
9.5	Sentimental Model Version 1 Summary	52
9.6	Sentimental Model Version 1 Accuracy Curve	53
9.7	Sentimental Model Version 1 Loss Curve	53
9.8	Sentimental Model Version 1 Confusion Matrix	54
9.9	Sentimental Model Version 1 Classification Report	54
9.10	Sentimental Model Version 2 Summary	55
9.11	Sentimental Model Version 2 Accuracy Curve	55
9.12	Sentimental Model Version 2 Loss Curve	56
9.13	Sentimental Model Version 2 Confusion Matrix	56
9.14	Sentimental Model Version 2 Classification Report	56
9.15	Kaggle and HuggingFace Sentimental Data Count plot	57
9.16	Sentimental Bert Model Confusion Matrix	58
9.17	Sentimental Bert Model Classification Report	58
10.1	Homepage Light Mode	60
10.2	Transformer based LM Frontend	60
10.3	Probabilistic Language Model Frontend	61
10.4	Positive Sentiment Prediction	62
10.5	Negative Sentiment Prediction	62
10.6	Neutral Sentiment Prediction	63
10.7	Word Embedding 2d plot Frontend	63
10.8	Word Embedding 3d plot Frontend	64
10.9	Auto Spelling Correction	64
10.10	Manual Spelling Correction	65

List of Tables

6.1	Count Matrix for bi-gram model	32
6.2	Count Matrix for bi-gram model with row-sum	32
6.3	Probability Matrix for bi-gram model	33
6.4	Train, Validation and Test Split	34

Chapter 1

Introduction to Nepali Language Model

1.1 Abstract

Language modeling (LM) is the use of various statistical and probabilistic techniques to determine the probability of a given sequence of words occurring in a sentence. Language models analyze bodies of text data to provide a basis for their word predictions. They are used in natural language processing (NLP) applications, particularly ones that generate text as an output. Some of these applications include , machine translation and question answering.

Some common statistical language model are:

1.1.1 N-gram

N-grams are a relatively simple approach to language models. They create a probability distribution for a sequence of n. The n can be any number, and defines the size of the "gram", or sequence of words being assigned a probability. For example, if n = 5, a gram might look like this: "can you please call me." The model then assigns probabilities using sequences of n size. Basically, n can be thought of as the amount of context the model is told to consider. Some types of n-grams are unigrams, bigrams, trigrams and so on.

1.1.2 Continuous Space

This type of model represents words as a non-linear combination of weights in a neural network. The process of assigning a weight to a word is also known as word embedding. This type becomes especially useful as data sets get increasingly large, because larger datasets often include more unique words. The presence of a lot of unique or rarely used words can cause problems for linear model like an n-gram. This is because the amount of possible word sequences increases, and the patterns that inform results become weaker. By weighting words in a non-linear, distributed way, this model can "learn" to approximate words and therefore not be misled by any unknown values. Its "understanding" of a given word is not as tightly tethered to the immediate surrounding words as it is in n-gram models.

Keywords : Transformers, LSTMs, Word2Vec, Attention, Tokenization



Figure 1.1: Nepali Word Cloud

1.2 Problem Statement

1. Nepali Language is rich in vocabulary and it is difficult to choose the best possible vocab.
2. Spelling correction for nepali language available today are based on dictionary rather than contextual meaning of the sentence.
3. There is no proper development of various NLP tasks like text generation, text summarization, image captioning, text to speech, etc. due to lack of reliable nepali language model

1.3 Objectives

1. To develop nepali language model for text generation.
2. Use the nepali language model to develop the spelling correction based on contextual meaning.

1.4 Expected Outcome

For language generation model, after user input some text, language generation model should suggest next words as shown below.



Figure 1.2: Nepali Text Generation

Similarly, for the spelling correction model, the model should be able to correct word based on contextual meaning as shown below.

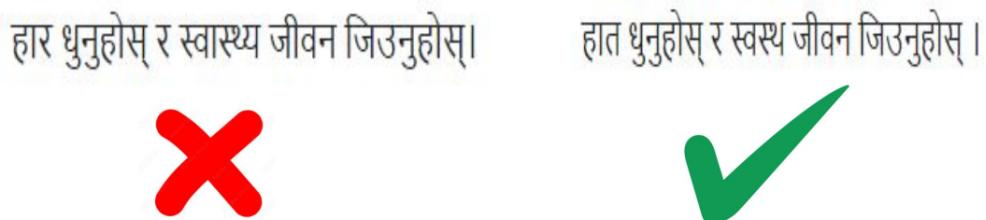


Figure 1.3: Nepali Spelling Correction based on contextual meaning

1.5 Data Source Link

1. Nepali Sentimental Data from Kaggle
2. Nepali Sentimental Data from HuggingFace
3. Nepali Corpus used in Probabilistic Language Model and Word Embedding
4. Nepali Corpus from Kaggle used in Transformer Based Language Model

Chapter 2

Literature Review

2.1 A Neural Probabilistic Language Model

2.1.1 Introduction

Goal of statistical language modeling is to learn the joint probability function of sequences of words in a language. This is intrinsically difficult because of the curse of dimensionality. NPLM proposes to fight the curse of dimensionality by learning a distributed representation of words which allows each training sentence to inform the model about an exponential number of semantically neighboring sentences. The model learns simultaneously a distributed representation for each word along with the probability function for word sequences, expressed in terms of these representations. Generalization is obtained because a sequence of words that has never been seen before gets high probability if it is made of words that are similar to the words forming an already seen sentence. The idea of the proposed model can be summarized as follows

- associate with each word in the vocabulary a distributed word feature vector (a real-valued vector in \mathbb{R}^m , where m is the size of embedding vector.)
- express the joint probability function of word sequences in terms of the feature vectors of these words in the sequence
- learn simultaneously the word feature vectors and the parameters of that probability function

2.1.2 NPLM Architecture

The basic form of the model is shown in the figure 2.1. The objective is to learn the function $f(w_t, w_{t-1}, \dots, w_{t-n}) = P(w_t | w_1^{t-1})$, in the sense that it gives high out-of-sample likelihood. A mapping C from any element of V to a real vector $C(i) \in \mathbb{R}^m$. It represents the distributed feature vector associated with each word in the vocabulary. In practice, C is represented by a $|V| \times m$ matrix (of free parameters)

From the direct architecture figure 2.1, $f(i, w_{t-1}, w_{t-2}, \dots, w_{t-n}) = g(i, C(w_{t-1}), C(w_{t-2}), \dots, C(w_{t-n}))$. Softmax function is used in the output layer of the neural network to get the probability of the target word.

The function f is the composition of two mappings (C and g), with C being shared across all words in the context. The function g may be implemented by a feed-forward or recurrent neural network or another parameterized function, with parameters θ .

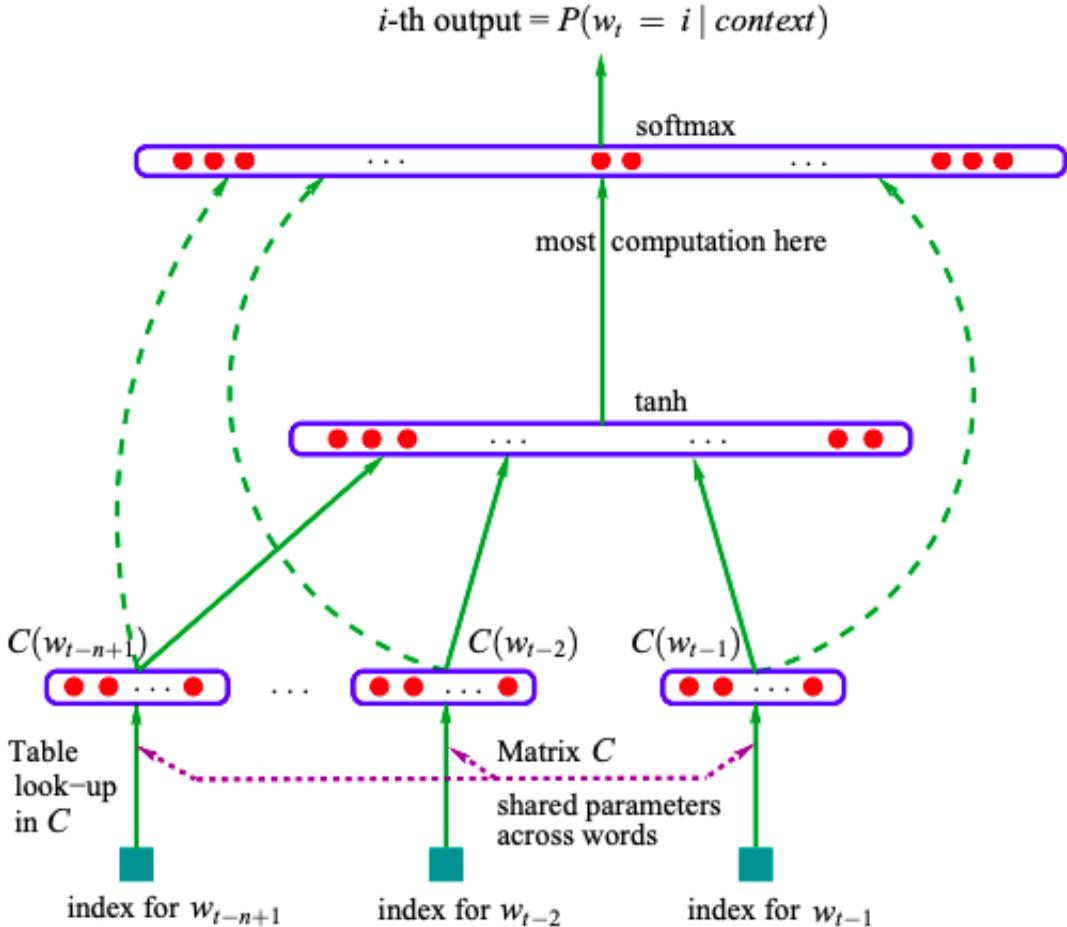


Figure 2.1: Direct Architecture of Probabilistic Language Model

2.1.3 Conclusion and Results

The main result of this experiment is that the neural network performs much better than the smoothed trigram. Greater the length of the context words and higher the number of hidden units, the model becomes more efficient. Moreover, direct architecture was found to be better by about 2% than the cycling architecture.

It can be deduced that the neural probabilistic model performs better due to the advantage of the learned distributed representation to fight the curse of dimensionality.

2.2 Attention is all you need

2.2.1 Introduction

Before the introduction of the transformer, the dominant sequence transduction models were based on the complex recurrent or convolutional neural networks that include an encoder and a decoder. But this paper introduced a new network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train.

Attention mechanisms have become an integral part of compelling sequence modeling and transduction models in various tasks, allowing the modeling of dependencies without regard to their distance in the input or output sequences. In all but a few cases, however, such attention mechanisms are used in conjunction with a recurrent network.

2.2.2 Transformer Model Architecture

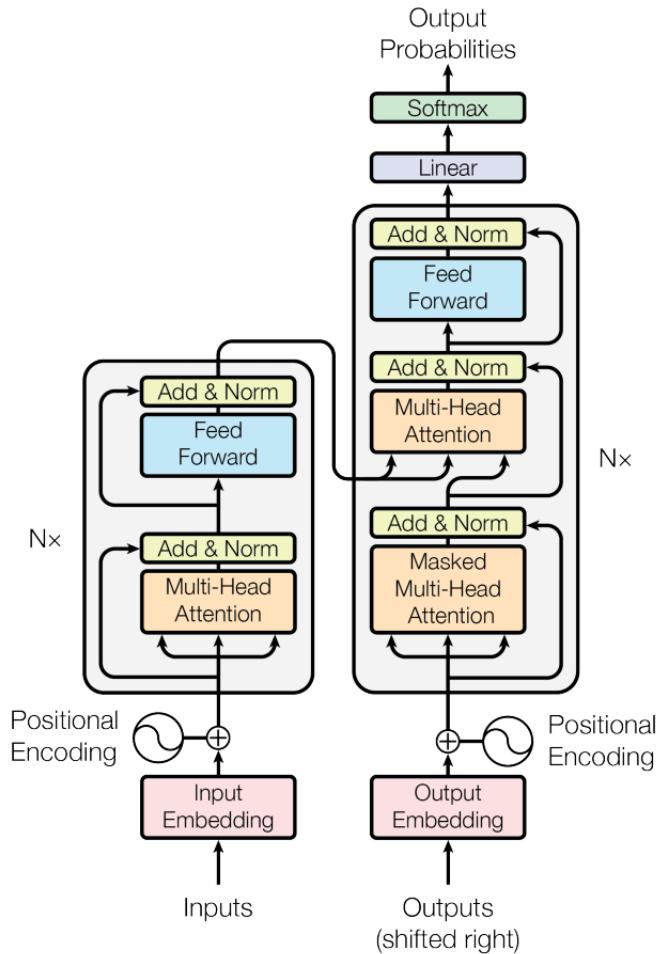


Figure 2.2: The Transformer - model architecture

Encoder and Decoder Stacks

The encoder is composed of a stack of $N = 6$ identical layers. Each layer has two sublayers: multi-head self-attention mechanism, and the simple position wise fully connected feed-forward network. Residual connection around each of two sub layers, followed by a layer normalization is employed in the encoder.

The decoder is also composed of a stack of $N = 6$ identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, residual connections around each of the sub layers, followed by layer normalization is employed. Self attention sub-layer in the decoder stack is modified to prevent the positions from attending to subsequent positions.

Attention

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

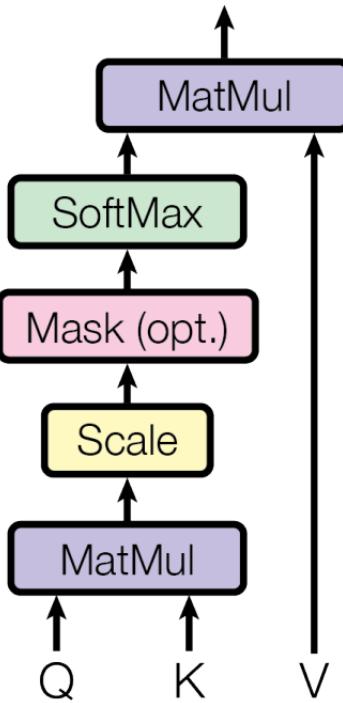


Figure 2.3: Scaled Dot-Product Attention

In scaled dot-product attention, the input consists of queries and keys of dimension d_k and values of dimension d_v . We compute the dot products of the query with all keys, divide each by d_k and apply a softmax function to obtain the weights on the values.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (2.1)$$

In multi-head attention, instead of performing a single attention function with d_{model} -dimensional keys, values and queries, we found it beneficial to linearly project the queries, keys and values h times with different, learned linear projections to d_k , d_k and d_v dimensions, respectively. On each of these projected versions of queries, keys and values we then perform the attention function in parallel, yielding d_v -dimensional output values. These are concatenated and once again projected, resulting in the final values, as depicted in figure 2.4.

Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_2)W^o \quad (2.2)$$

$$\text{where } \text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

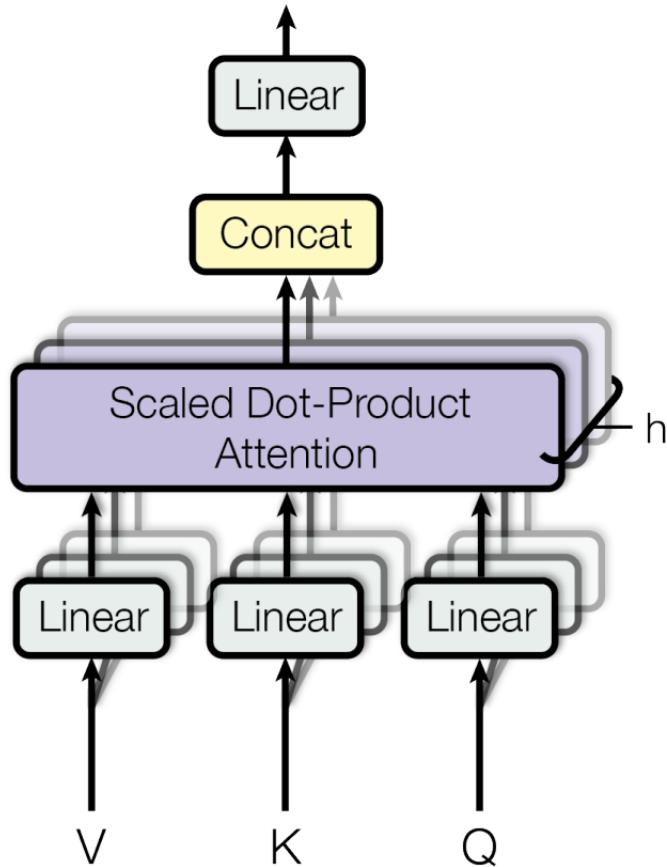


Figure 2.4: Multi-Head Attention consists of attention layer running in parallel

Position wise Feed Forward Neural Network

In addition to attention sub-layers, each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically.

This consists of two linear transformations with a ReLU activation in between.

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2 \quad (2.3)$$

Embedding and Softmax

Similarly to other sequence transduction models, Learned embeddings are used to convert the input tokens and output tokens of dimension d_{model} . Usual learned linear transformation and softmax function are used to convert the decoder output to predict next-token probabilities.

Positional Encoding

Since the model contains no recurrence and no convolution, in order for the model to make use of the order of the sequence, some information about the relative or absolute position of the tokens in the sequence must be injected. To this end, “positional encodings” are added to the input embeddings at the bottoms of the encoder and decoder stacks. The positional encodings have the same dimension model as the embeddings, so that the two can be summed. Sine and cosine functions of different frequencies are used for positional encoding as follow

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \quad (2.4)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right) \quad (2.5)$$

where pos is the position and i is the dimension. That is, each dimension of the positinal encoding corresponds to a sinusoid.

2.2.3 Conclusion and Results

In the machine translation task, the transformer model outperformed all previous state of the art models which have used the RNNs, GRUs and LSTMs. To evaluate if the Transformer can generalize to other tasks, experiments on English constituency parsing were performed and despite the lack of task-specific tuning, the transformer model performed surprisingly well, yielding better results than all previously reported models with the exception of the Recurrent Neural Network Grammar.

In this work, the Transformer, the first sequence transduction model based entirely on attention, replacing the recurrent layers most commonly used in encoder-decoder architectures with multi-headed self-attention is presented.

Till today, various transformer models have been developed which have performed better on all kinds of natural language processing tasks like language modeling, text classifications, questions answering, machine translation, sentence similarity, summarization, etc.

Chapter 3

Requirement Analysis and Feasibility Study

3.1 Functional and Non Functional Requirements

3.1.1 Functional Requirements

1. R1: Word Embeddings Visualization

Description : Word embeddings of nepali words created using various methods like word2vec, embedding layers should be visualized in 2d and 3d graphs and show the required relationship between the words.

(a) Visualize word embeddings

Input : Select word embeddings visualization tab

Output : 2d and 3d graphs of the word embeddings vector of few selected words or default 100 or 200 words(Number of words may vary as per requirements).

Processing : All the default word embeddings vectors will be returned and plotted in 2d and 3d graphs.

(b) Search Words

Input : Keyword to be searched in the graph

Output: Vector of keyword and position in the graphs highlighted

Processing: Keyword searched in the vocabulary and if found its vector is returned and plotted with highlight color in the graph. If not found, display NOT FOUND message. If the keyword is not found in the vocabulary, archive it to the vocabulary database for future references.

2. R2: Sentiment Classification

Description : When the user enters a sentence and performs sentimental classification, the sentimental classification should be able to distinguish the sentence into positive, negative and neutral with some cutoff probability (about 70%).

(a) Classify and result

Input : A sentence

Ouput : Probabilities of the sentence being positive, negative and neutral.

Processing :

(b) Store the result

Input : A sentence to be classified

Ouput : Response message for successful storage

Processing : Store the sentence and its predicted label in the sentimental classification table. If the predicted label is incorrect , get the correct label from the user and store it in the database.

3. R3: Language Modelling

Description : Nepali Language modeling based on both probabilistic and neural approach. Based on the trained language model, it should be able to auto generate the next word given context words.

(a) Predict next word

Input : An incomplete sentence

Ouput : List of words along the probabilities of being the next word.

Processing : The language model should be able to predict the next word from vocabulary based on the previous words provided by the user.

4. R4: Spelling Correction

Description : Based on the trained language model, the system should suggest correct spelling for a given sentence.

Input : A sentence

Ouput : Provide suggestions to correct the spelling of the words in the input sentence.

Processing : Firstly, candidate sentences are found using 1 or 2 minimum edit distance. Probability of each candidate sentence and likelihood of error of given sentence is calculated. And suggestions should be made based on maximum posterior.

5. R5: Interactive Web Application

Description : All the requirements R1, R2, R3 and R4 should be easily accessible through a website and APIs.

3.1.2 Non Functional Requirements

1. R1: Performance and scalability

(a) The load time for the user interface screens shall take no longer than 3 seconds.

(b) Queries shall return results within 3 seconds.

2. R2: Design Constraints

- (a) The system shall be developed using python and postgresql databases.

3. R3: Standard Compliance

- (a) The graphical user interface shall have a consistent look and feel.

4. R4: Availability

- (a) The system shall be available all time.

5. R5: Portability and Compatibility

- (a) The system shall be able to run in all browsers.

6. R6: Reliability and Maintainability

- (a) The mean time to recover from a system failure must not be greater than 2 hours.

- (b) Rate of system failure should not be greater than twice a month.

3.2 Hardware Requirements

1. GeForce RTX 3060 for prototype development
2. Minimum 32 GB RAM
3. Minimum 15 GB Storage

3.3 Software Requirements

1. Programming Language : Python, Javascript
2. Libraries : Tensorflow, keras, scikit-learn, spacy, nltk, and other python libraries
3. Frameworks : Django, Django Rest Framework, React
4. Application software : Postman
5. Target Platform : Web

3.4 Feasibility Study

Feasibility is defined as the practical extent to which a project can be performed successfully. To evaluate feasibility, a feasibility study is performed, which determines whether the solution considered to accomplish the requirements is practical and workable in the software.

3.4.1 Technical Feasibility

Technical Feasibility is the formal process of assessing whether it is technically possible to manufacture a product or service. For our system, the required software and hardware were readily available to us. So we can state with confidence that the project is technically feasible.

3.4.2 Economic Feasibility

The purpose of an Economic Feasibility Study (EFS) is to demonstrate the net benefit of a proposed project, taking into consideration the benefits and costs to the agency, other state agencies, and the general public as a whole. It is the process of determining whether the project is worth the cost and time investment. Since most of the hardware was already in our possession and the softwares used was mostly free, we can declare that the system is economically feasible.

3.4.3 Legal Feasibility

Since we are using open source data for building language model, we can state that the project is legally feasible.

3.4.4 Scheduling Feasibility

This assessment is the most important for project success; after all, a project will fail if not completed on time. In scheduling feasibility, an organization estimates how much time the project will take to complete. Since we have properly planned our approach to completing the project in components using agile model, we can say that it is feasible timewise.

Chapter 4

System Design and Architecture

4.1 Use Case Diagram

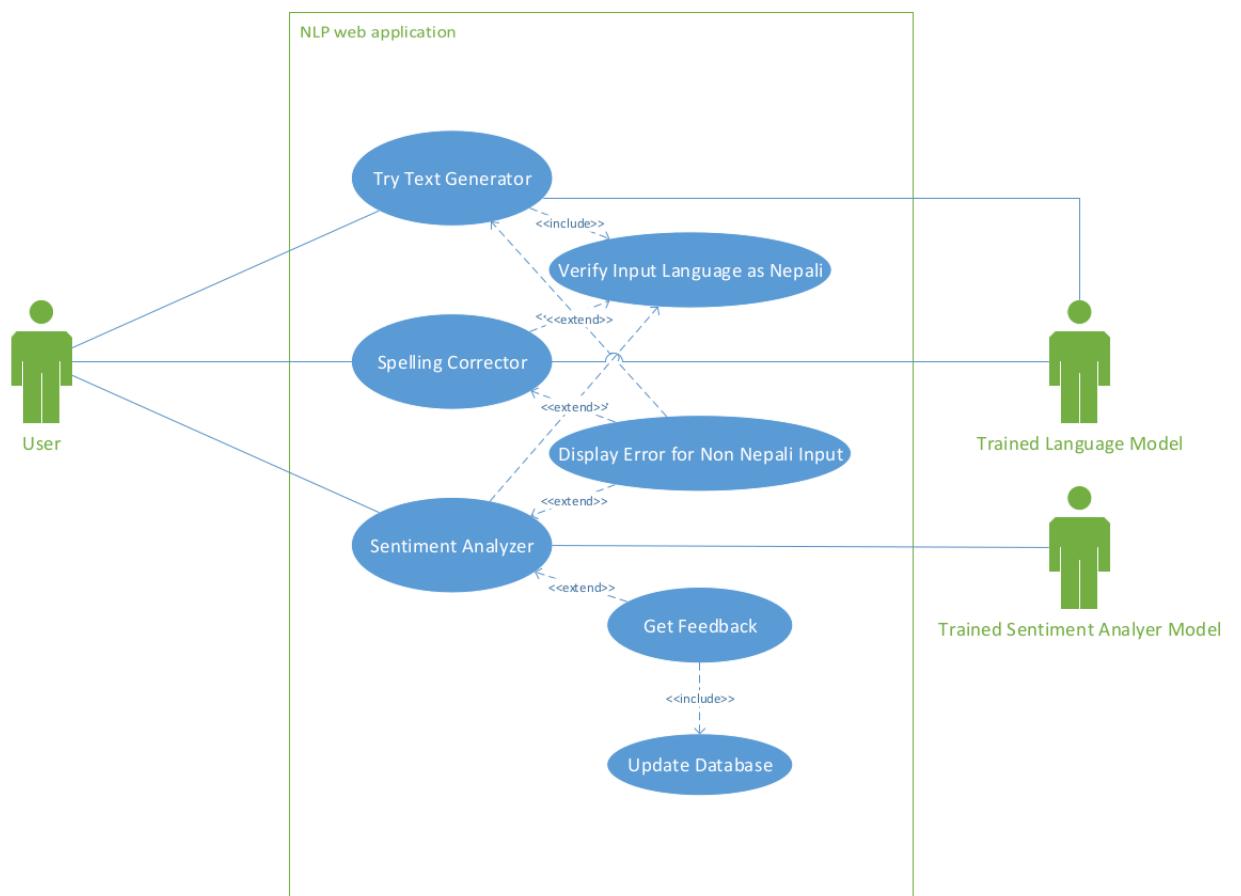


Figure 4.1: System Use Case Diagram

4.2 Activity Diagram

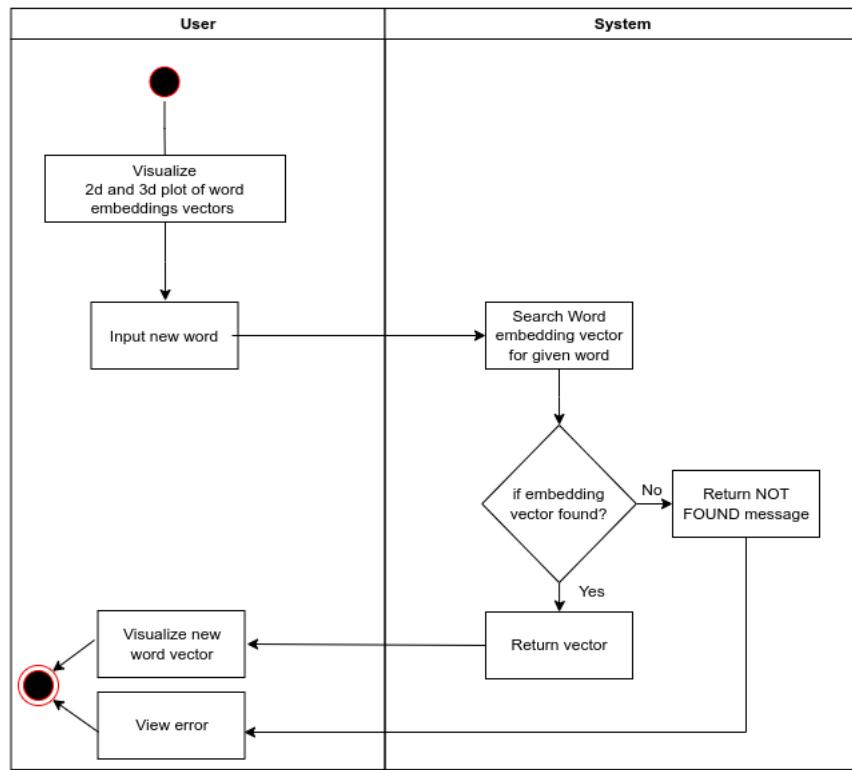


Figure 4.2: Word Embeddings Activity Diagram

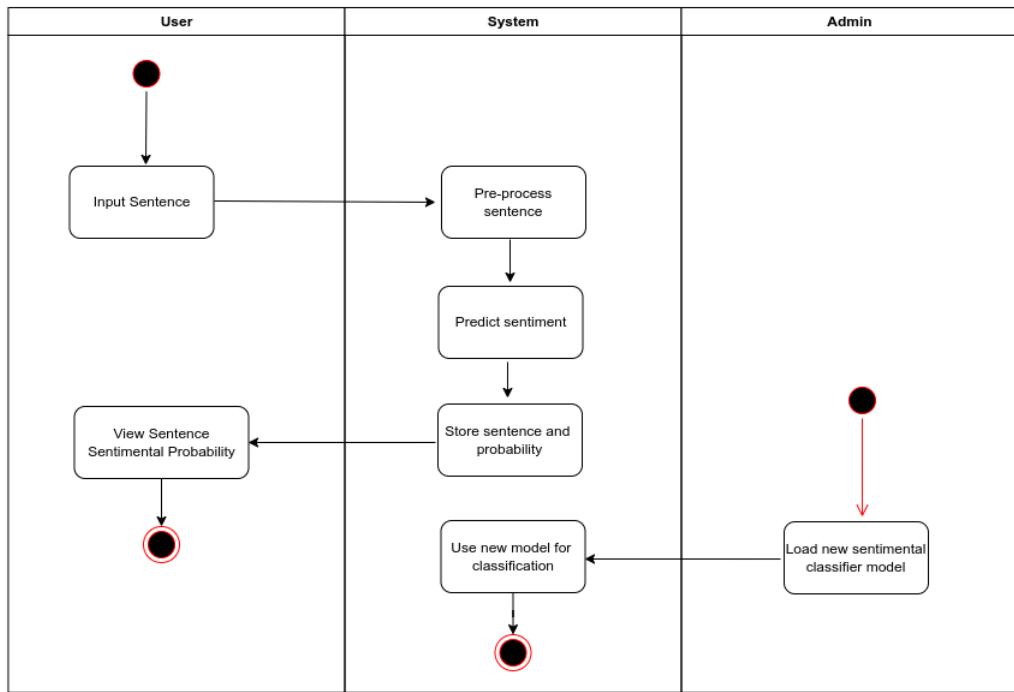


Figure 4.3: Sentiment Classification Activity Diagram

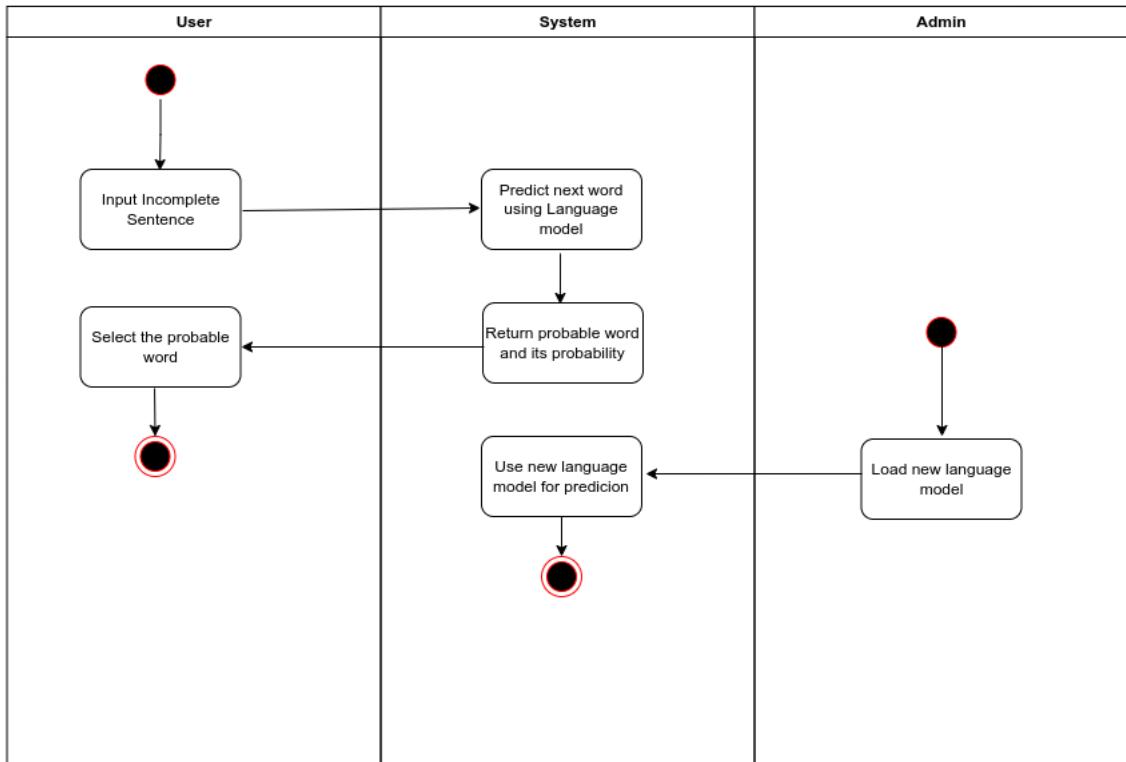


Figure 4.4: Language model Activity Diagram

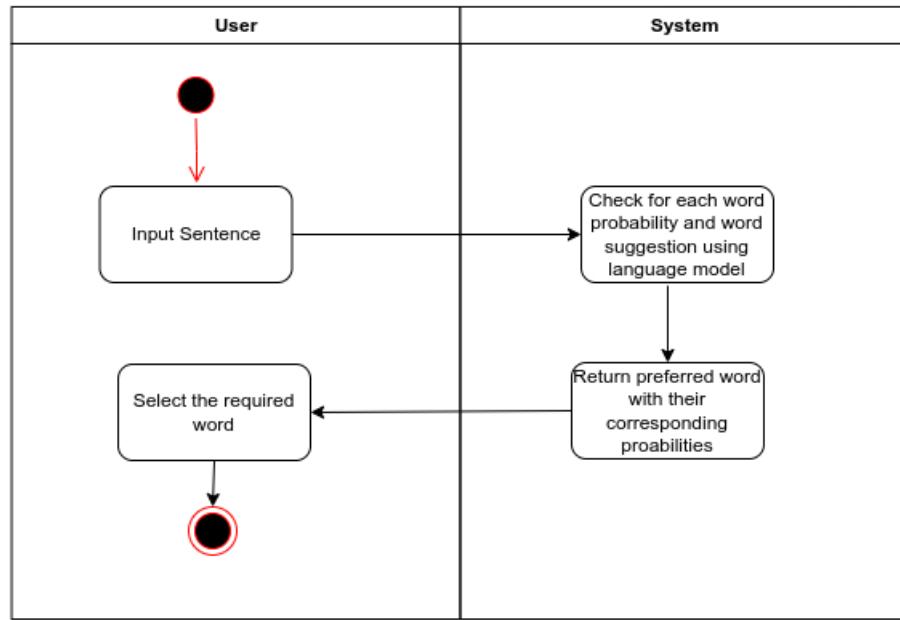


Figure 4.5: Spelling Correction Activity Diagram

Chapter 5

Project Management

5.1 Agile Development Methodology

Teams use the agile development methodology to minimize risk (such as bugs, cost overruns, and changing requirements) when adding new functionality. In all agile methods, teams develop the software in iterations that contain mini-increments of the new functionality. There are many different forms of the agile development method, including scrum, crystal, extreme programming (XP), and feature-driven development (FDD).

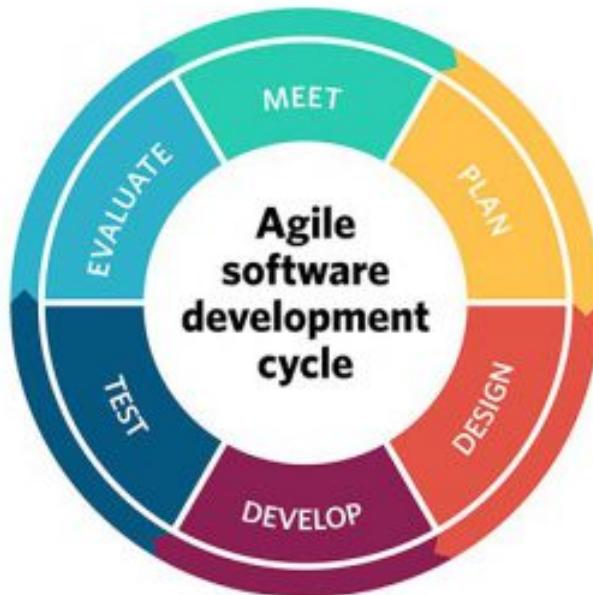


Figure 5.1: Agile Development Methodology

5.2 Trello

Trello is a web-based project management and collaboration tool that allows users to organize tasks and projects using boards, lists, and cards.

We have used trello's agile board template for project management and collaboration.

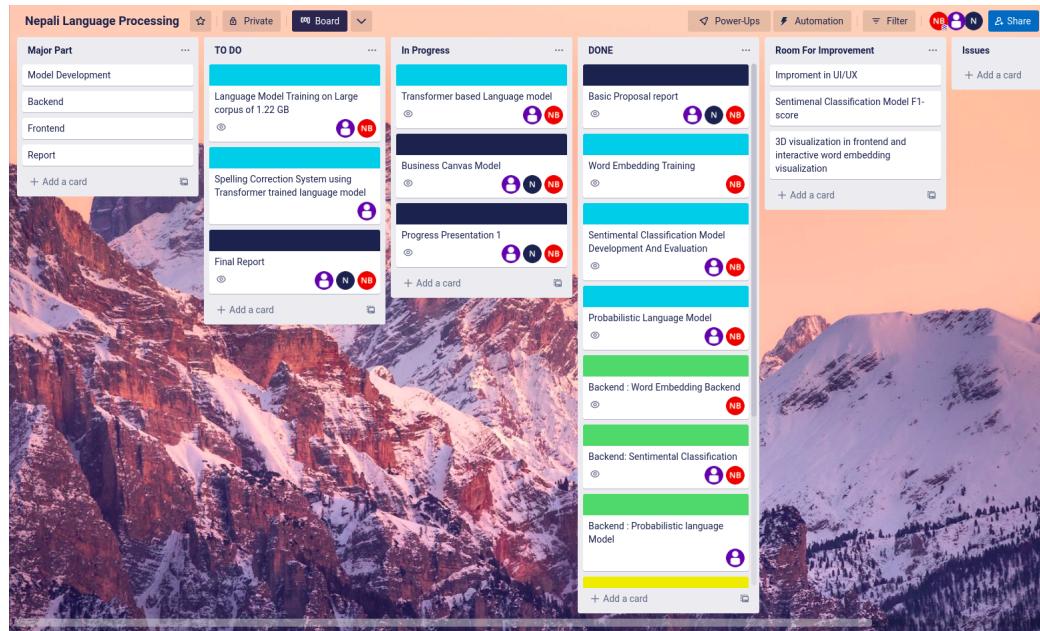


Figure 5.2: Trello

5.3 Discord

We have used discord for effective communication and team meetings.

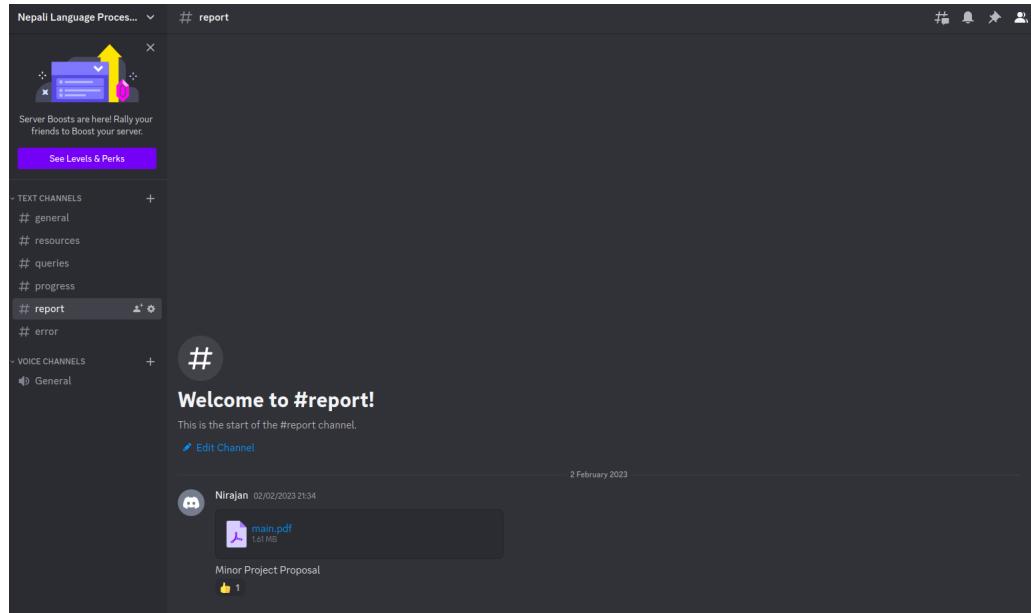


Figure 5.3: Discord

Chapter 6

Probabilistic Language Model

6.1 Introduction

Language model is used to estimate the probability of the word sequences and to estimate the probability of a word following the sequence of words. The concept can be used to auto-complete a sentence with most likely suggestions as shown below:

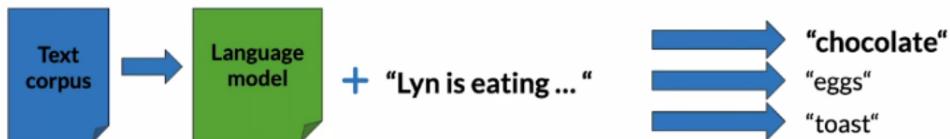


Figure 6.1: Language Model for Auto-complete

Some other applications of the language model are:

1. Speech Recognition
2. Spelling Correction
3. Augmentative Communication

6.2 N-gram

A N-gram is a sequence of N-words. For example:

Corpus : I am happy because I am learning.

Uni-grams : {I, am, happy, because, I, am, learning }

Bi-grams : {I am, am happy, happy because, ... }

Tri-grams : {I am happy, am happy because, ... }

6.3 Sequence Notation

Let's define a standard sequence notation to represent the sequence of words in our corpus. Let our corpus be as shown below where n^{th} is represented as W_n . Let the number of words in the corpus be $m = 500$

Corpus: This is great. ... teacher drinks tea.

Hence, $W_1 = \text{This}$, $W_2 = \text{is}$, $W_3 = \text{great}$ and so on...

Let us represent the sequence of words as W_s^e where s represent the starting index and e represent the ending index of the words in the corpus. So,

$$W_1^m = W_1 \ W_2 \ W_3 \dots \ W_{m-1} \ W_m$$

Similarly, $W_1^3 = W_1 \ W_2 \ W_3 = \text{This is great}$

6.4 N-gram Probability

N-gram probability is the probability of the occurrence of the sequence of N-words in a corpus. From uni-gram, bi-gram and tri-gram probability, we can deduce for N-gram probability denoted by $P(W_N|W_1^{N-1})$ as

$$P(W_N|W_1^{N-1}) = \frac{C(W_1^{N-1} W_N)}{C(W_1^{N-1})} = \frac{C(W_1^N)}{C(W_1^{N-1})} \quad (6.1)$$

6.4.1 Uni-gram Probability

Uni-gram Probability is the probability of occurrence of a word in a given corpus.

Corpus : I am happy because I am learning.

Size of corpus : $m = 7$

$$P(I) = \frac{2}{7} \quad (6.2)$$

$$P(\text{happy}) = \frac{1}{7} \quad (6.3)$$

Hence, we can see that Probability of uni-gram is given by:

$$P(w) = \frac{C(w)}{m} \quad (6.4)$$

where $C(w)$ = frequency of the word in the corpus

6.4.2 Bi-gram Probability

Bi-gram Probability is the probability of occurrence of sequence of two words in a corpus.

Corpus : I am happy because I am learning.

Size of corpus : $m = 7$

$$P(am|I) = \frac{C(I \ am)}{C(I)} = \frac{2}{2} = 1 \quad (6.5)$$

$$P(happy|I) = \frac{C(I\ happy)}{C(I)} = \frac{0}{7} = 0 \quad (6.6)$$

$$P(learning|am) = \frac{C(am\ learning)}{C(am)} = 1/2 \quad (6.7)$$

Hence, we can see that Probability of bi-gram is given by:

$$P(y|x) = \frac{C(x\ y)}{C(x)} \quad (6.8)$$

where $C(w)$ = frequency of the word in the corpus

6.4.3 Tri-gram Probability

Tri-gram Probability is the probability of the occurrence of the sequence of three words in a corpus.

Corpus : I am happy because I am learning.

Size of corpus : $m = 7$

$$P(happy|I\ am) = \frac{C(I\ am\ happy)}{C(I\ am)} = \frac{1}{2} \quad (6.9)$$

Hence, we can see that Probability of bi-gram is given by:

$$P(W_3|W_1^2) = \frac{C(W_1^2\ W_3)}{C(W_1^2)} = \frac{C(W_1^3)}{C(W_1^2)} \quad (6.10)$$

where $C(w)$ = frequency of the word in the corpus

6.5 Probability of a sequence

From the conditional probability and chain rule, we have

$$P(B|A) = \frac{P(A, B)}{P(A)} \quad (6.11)$$

From (6.11), we get

$$P(A, B) = P(A).P(B|A) \quad (6.12)$$

Using (6.12), we can deduce

$$P(A, B, C, D) = P(A).P(B|A).P(C|A, B).P(D|A, B, C) \quad (6.13)$$

Now, the probability of the sequence of words: "the teacher drinks tea" is given by:

$$\begin{aligned} P(\text{the teacher drinks tea}) &= P(\text{the}).P(\text{teacher}|\text{the}) \\ &\quad P(\text{drinks}|\text{the teacher}).P(\text{tea}|\text{the teacher drinks}) \end{aligned} \quad (6.14)$$

6.6 Approximation of Sequence Probability

When using n-gram language model, the exact sentence for the required n-gram might not be present exactly in corpus due to which the frequency of the words sequence becomes zero and hence the probability. For example:

Input : the teacher drinks tea

$$P(\text{the teacher drinks tea}) = P(\text{the}) \cdot P(\text{teacher} | \text{the}) \cdot P(\text{drinks} | \text{the teacher}) \cdot P(\text{tea} | \text{the teacher drinks})$$

Here,

$$P(\text{tea} | \text{the teacher drinks}) = \frac{C(\text{the teacher drinks tea})}{C(\text{the teacher drinks})} \quad (6.15)$$

In the above equation (6.15), the frequency of "the teacher drinks tea" and "the teacher drinks" are both likely 0.

Hence, in order to calculate the required probability of the sequence of the words, as per Markov assumption, only last N words matter. Suppose, only last one words matter then, we have

$$\begin{aligned} & P(\text{the teacher drinks tea}) \\ &= P(\text{the}) \cdot P(\text{teacher} | \text{the}) \cdot P(\text{drinks} | \text{the teacher}) \cdot P(\text{tea} | \text{the teacher drinks}) \\ &\approx P(\text{the}) \cdot P(\text{teacher} | \text{the}) \cdot P(\text{drinks} | \text{teacher}) \cdot P(\text{tea} | \text{drinks}) \end{aligned}$$

Hence, we can model entire sentence with n-gram following Markov Assumption as follow

$$P(W_n | W_1^{n-1}) \approx P(W_n | W_{n-N+1}^{n-1}) \quad (6.16)$$

6.7 Starting and Ending Sentences

6.7.1 Start of sentence symbol $\langle s \rangle$

For an input sentence, "the teacher drinks tea", we have

$$P(\text{the teacher drinks tea}) \approx P(\text{the}) \cdot P(\text{teacher} | \text{the}) \cdot P(\text{drinks} | \text{teacher}) \cdot P(\text{tea} | \text{drinks})$$

After, we add a start token $\langle s \rangle$, we have

Input : $\langle s \rangle$ the teacher drinks tea

$$P(\text{the teacher drinks tea}) \approx P(\text{the} | \langle s \rangle) \cdot P(\text{teacher} | \text{the}) \cdot P(\text{drinks} | \text{teacher}) \cdot P(\text{tea} | \text{drinks})$$

In this way, we can add start token to determine the possible starting words and calculate the probability of sequence considering first word in N-gram language model. For Tri-gram and N-gram, we add (N - 1) start token $\langle s \rangle$ in the beginning of the sequence.

6.7.2 End of sentence symbol $\langle /s \rangle$

For an input sentence, "the teacher drinks tea", we have

$$P(\text{the teacher drinks tea}) \approx P(\text{the}) \cdot P(\text{teacher} | \text{the}) \cdot P(\text{drinks} | \text{teacher}) \cdot P(\text{tea} | \text{drinks})$$

After, we add a start token $\langle s \rangle$ and an end token $\langle /s \rangle$, we have

Input : $\langle s \rangle$ the teacher drinks tea $\langle /s \rangle$

$$P(\text{the teacher drinks tea}) \approx P(\text{the} | \langle s \rangle) \cdot P(\text{teacher} | \text{the}) \cdot P(\text{drinks} | \text{teacher}) \cdot P(\text{tea} | \text{drinks}) \cdot P(\langle /s \rangle | \text{tea})$$

For bi-gram, tri-gram or any N-gram model, we add end token just once at the end of sentence unlike start token

6.8 Count Matrix

Count matrix is the matrix containing the frequency of occurrence of N-gram. Rows in count matrix represent the unique corpus of (N - 1) gram and columns represent the unique corpus word.

Corpus : $\langle s \rangle$ I study I learn $\langle /s \rangle$

Bi-gram Count Matrix is as follow:

	$\langle s \rangle$	$\langle /s \rangle$	I	study	learn
$\langle s \rangle$	0	0	1	0	0
$\langle /s \rangle$	0	0	0	0	0
I	0	0	0	1	1
study	0	0	1	0	0
learn	0	1	0	0	0

Table 6.1: Count Matrix for bi-gram model

6.9 Probability Matrix

Probability matrix is the matrix containing the probability of the occurrence of N-gram words sequence. It can be created by dividing each elements of the count matrix by total sum of all row elements. For example:

Corpus : $\langle s \rangle$ I study I learn $\langle /s \rangle$

	$\langle s \rangle$	$\langle /s \rangle$	I	study	learn	sum
$\langle s \rangle$	0	0	1	0	0	1
$\langle /s \rangle$	0	0	0	0	0	0
I	0	0	0	1	1	2
study	0	0	1	0	0	1
learn	0	1	0	0	0	1

Table 6.2: Count Matrix for bi-gram model with row-sum

	$\langle s \rangle$	$\langle /s \rangle$	I	study	learn
$\langle s \rangle$	0	0	1	0	0
$\langle /s \rangle$	0	0	0	0	0
I	0	0	0	0.5	0.5
study	0	0	1	0	0
learn	0	1	0	0	0

Table 6.3: Probability Matrix for bi-gram model

From the above probability matrix, we can calculate the probability of sentence and also predict the next word based on higher probability.

For example :

Input : $\langle s \rangle$ I learn $\langle /s \rangle$

$$\begin{aligned}
 P(\text{Input}) &= P(\langle s \rangle \text{ I learn } \langle /s \rangle) \\
 &= P(\text{I} \mid \langle s \rangle) \cdot P(\text{learn} \mid \text{I}) \cdot P(\langle /s \rangle \mid \text{learn}) \\
 &= 1 \times 0.5 \times 1 \\
 &= 0.5
 \end{aligned}$$

6.10 Generative Language Model

Generative language model is used to generate a most likely sentences.

Algorithm for the generative language model is:

Step 1 : Choose sentence start

Step 2 : Choose next bi-gram starting with the previous word

Step 3 : Continue until $\langle /s \rangle$ is picked

For example:

Assume our corpus be:

$\langle s \rangle$ Lyn drinks chocolate $\langle /s \rangle$

$\langle s \rangle$ John drinks tea $\langle /s \rangle$

$\langle s \rangle$ Lyn eats chocolate $\langle /s \rangle$

Then, the generative language model can generate a sentence as follow:

Step 1: $(\langle s \rangle, \text{Lyn})$ or $(\langle s \rangle, \text{John})$

Step 2: $(\text{Lyn}, \text{eats})$ or $(\text{Lyn}, \text{drinks})$

Step 3: $(\text{drinks}, \text{tea})$ or $(\text{drinks}, \text{chocolate})$

Step 4: $(\text{tea}, \langle /s \rangle)$

6.11 Train, Validation and Test Split

The recommended split of corpus into train, validation and test data is as follow

Type of corpus	Train	Validation	Test
Small	80%	10%	10%
Large	98%	1%	1%

Table 6.4: Train, Validation and Test Split

6.12 Language Model Evaluation

We can use two different approaches to evaluate and compare language models:

6.12.1 Extrinsic evaluation

This involves evaluating the models by employing them in an actual task (such as machine translation) and looking at their final loss/accuracy. This is the best option as it's the only way to tangibly see how different models affect the task we're interested in. However, it can be computationally expensive and slow as it requires training a full system.

6.12.2 Intrinsic evaluation

This involves finding some metric to evaluate the language model itself, not taking into account the specific tasks it's going to be used for. While intrinsic evaluation is not as "good" as extrinsic evaluation as a final metric, it's a useful way of quickly comparing models. Perplexity is an intrinsic evaluation method.

6.12.3 Perplexity

Perplexity is an evaluation metric for language models. For better language model, perplexity value should be smaller. Perplexity can be calculated as follow:

$$PP(W) = P(s_1, s_2, s_3, s_4, \dots, s_m)^{\frac{-1}{m}} \quad (6.17)$$

where,

W = test set containing m sentences s

s_i = i^{th} sentence in the test set, each ending with $\langle /s \rangle$

m = number of all words in entire test set **W** including $\langle /s \rangle$ but not including $\langle s \rangle$

Similarly, Perplexity for the bi-gram model can be calculated as:

$$PP(W) = \sqrt[m]{\prod_{i=1}^m \prod_{j=1}^{|S_i|} \frac{1}{P(w_j^{(i)} | w_{j-1}^{(i)})}} \quad (6.18)$$

where, $w_j^{(i)}$ = j^{th} word in i^{th} sentence

If we concatenate all the sentences in W, we can get perplexity as

$$PP(W) = \sqrt[m]{\prod_{i=1}^m \frac{1}{P(w_i|w_{i-1})}} \quad (6.19)$$

where, w_i = i^{th} word in test set

6.12.4 Log Perplexity

Log perplexity is calculated by taking logarithm on both sides of equation (6.19) and we get

$$\log PP(W) = \frac{-1}{m} \sum_{i=1}^m \log_2(P(w_i|w_{i-1})) \quad (6.20)$$

6.13 Vocabulary

Vocabulary is a set of words from the training corpus. We can create vocabulary from training corpus using following criteria.

1. Min word frequency f
2. Max $|V|$, include words by frequency
3. Using $\langle UNK \rangle$ for unknown token

For example: Suppose our corpus is as follow

$\langle s \rangle$ Lyn drinks chocolate $\langle /s \rangle$

$\langle s \rangle$ John drinks tea $\langle /s \rangle$

$\langle s \rangle$ Lyn eats chocolate $\langle /s \rangle$

let min frequency to be added in vocabulary be $f = 2$. So, now corpus will be as follow:

$\langle s \rangle$ Lyn drinks chocolate $\langle /s \rangle$

$\langle s \rangle$ $\langle UNK \rangle$ drinks $\langle UNK \rangle$ $\langle /s \rangle$

$\langle s \rangle$ Lyn $\langle UNK \rangle$ chocolate $\langle /s \rangle$

Hence, vocabulary = {Lyn, drinks, chocolate}

Suppose,

Input Query : $\langle s \rangle$ Adam drinks chocolate $\langle /s \rangle$

So, the input query after processing becomes

Ouput : $\langle s \rangle$ $\langle UNK \rangle$ drinks chocolate $\langle /s \rangle$

In this way, vocabulary can be created and unknown words can be handled.

6.14 Missing N-gram in training corpus

N-grams made of known words still might be missing in the training corpus. N-gram probability is given as:

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}, w_n)}{C(w_{n-N+1}^{n-1})} \quad (6.21)$$

In equation (6.21), both $C(w_{n-N+1}^{n-1}, w_n)$ and $C(w_{n-N+1}^{n-1})$ can be 0 in the probability matrix which causes the probability to be indefinite. It can be solved by using smoothing.

6.14.1 Laplacian Smoothing

In Laplacian smoothing, probability is calculated as follow:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}, w_n) + 1}{C(w_{n-1}) + V} \quad (6.22)$$

6.14.2 K-smoothing

In K-smoothing, probability is calculated as follow

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}, w_n) + K}{C(w_{n-1}) + k * V} \quad (6.23)$$

Chapter 7

Sequence Modelling

7.1 Introduction

Sequence Modelling is the ability of a computer program to model, interpret, make predictions about or generate any type of sequential data, such as audio, text etc. For example, a computer program that can take a piece of text in English and translate it to French is an example of a Sequence Modelling program (because the type of data being dealt with is text, which is sequential in nature). An AI algorithm called the Recurrent Neural Network, is a specialised form of the classic Artificial Neural Network (Multi-Layer Perceptron) that is used to solve Sequence Modelling problems. Recurrent Neural Networks are like Artificial Neural Networks which has loops in them. This means that the activation of each neuron or cell depends not only on the current input to it but also its previous activation values.

7.2 Recurrent Neural Networks

Humans don't start their thinking from scratch every second. As we read a text, we understand each word based on your understanding of previous words. We don't throw everything away and start thinking from scratch again. Our thoughts have persistence.

Traditional neural networks can't do this, and it seems like a major shortcoming. For example, imagine you want to classify what kind of event is happening at every point in a movie. It's unclear how a traditional neural network could use its reasoning about previous events in the film to inform later ones.

Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist.

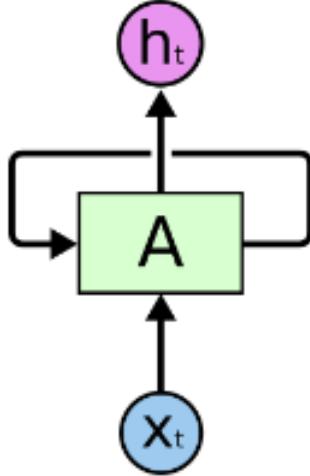


Figure 7.1: Recurrent Neural Networks

In the above diagram, a chunk of neural network, A , looks at some input x_t and outputs a value h_t . A loop allows information to be passed from one step of the network to the next.

These loops make recurrent neural networks seem kind of mysterious. However, if you think a bit more, it turns out that they aren't all that different than a normal neural network. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. Consider what happens if we unroll the loop:

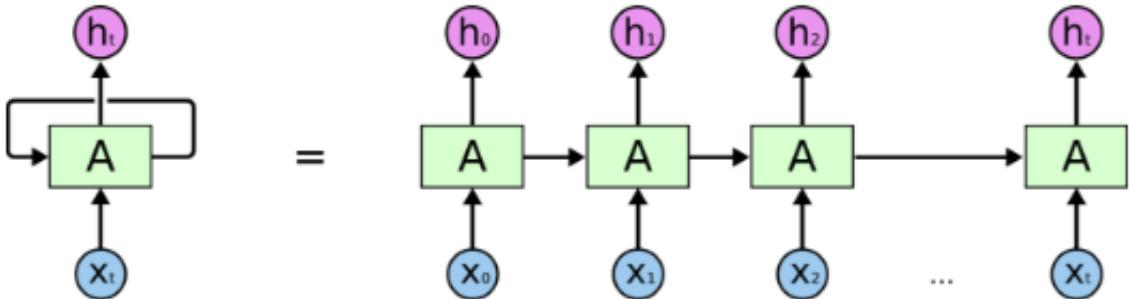


Figure 7.2: An unrolled recurrent neural networks

This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They're the natural architecture of neural network to use for such data.

And they certainly are used! In the last few years, there have been incredible success applying RNNs to a variety of problems: speech recognition, language modeling, translation, image captioning, etc.

7.3 Vanishing Gradients with RNNs

One of the appeals of RNNs is the idea that they might be able to connect previous information to the present task, such as using previous video frames might inform the understanding of the present frame. If RNNs could do this, they'd be extremely useful. But can they? It depends.

Sometimes, we only need to look at recent information to perform the present task. For example, consider a language model trying to predict the next word based on the previous ones. If we are trying to predict the last word in “the clouds are in the sky,” we don't need any further context - it's pretty obvious the next word is going to be sky. In such cases, where the gap between the relevant information and the place that it's needed is small, RNNs can learn to use the past information.

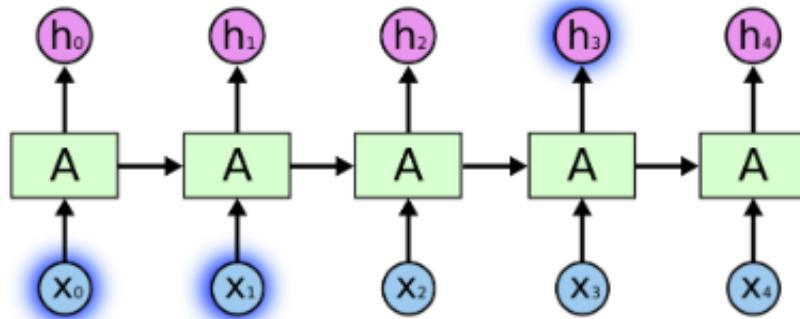


Figure 7.3: Short Sequence in RNNs

But there are also cases where we need more context. Consider trying to predict the last word in the text “I grew up in France... I speak fluent French.” Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back. It's entirely possible for the gap between the relevant information and the point where it is needed to become very large.

Unfortunately, as that gap grows, RNNs become unable to learn to connect the information.

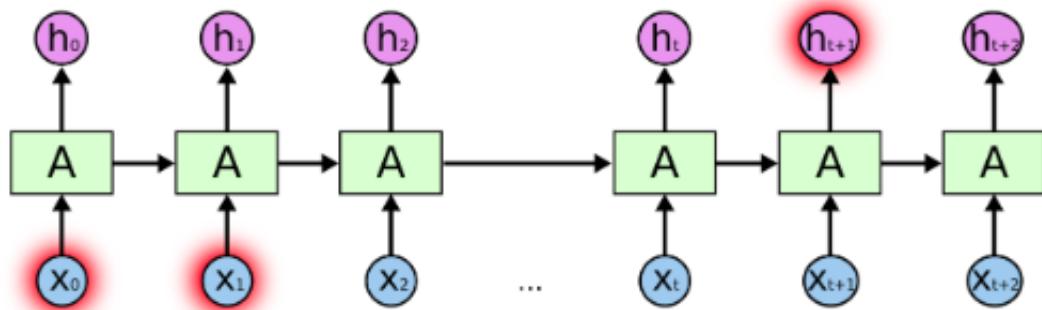


Figure 7.4: Long Sequence in RNNs

In theory, RNNs are absolutely capable of handling such “long-term dependencies.” A human could carefully pick parameters for them to solve toy problems of this form. Sadly, in practice,

RNNs don't seem to be able to learn them. The problem was explored in depth by Hochreiter (1991) [German] and Bengio, et al. (1994), who found some pretty fundamental reasons why it might be difficult.

7.4 LSTMs

Long Short Term Memory networks - usually just called “LSTMs” - are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter and Schmidhuber (1997), and were refined and popularized by many people in following work. They work tremendously well on a large variety of problems, and are now widely used.

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer as shown below.

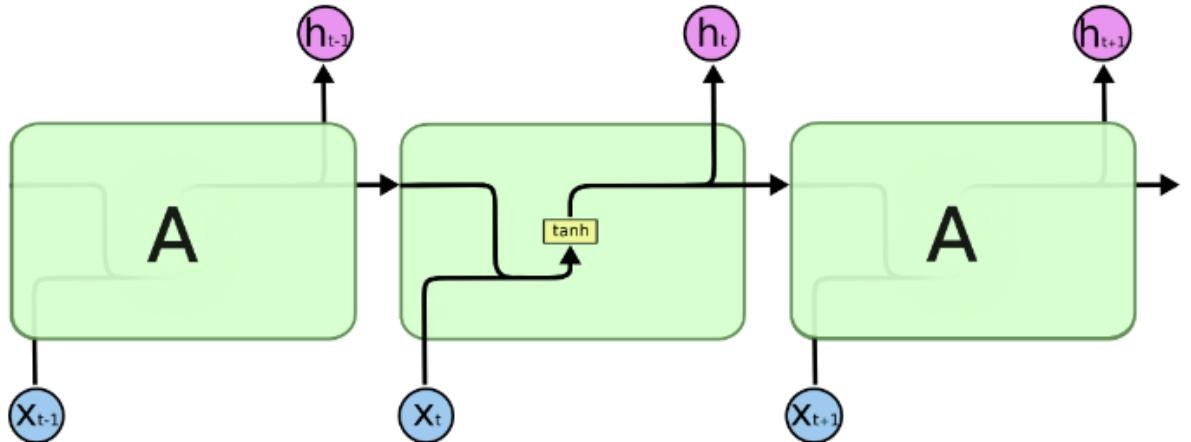


Figure 7.5: The repeating module in a standard RNN contains a single layer.

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.

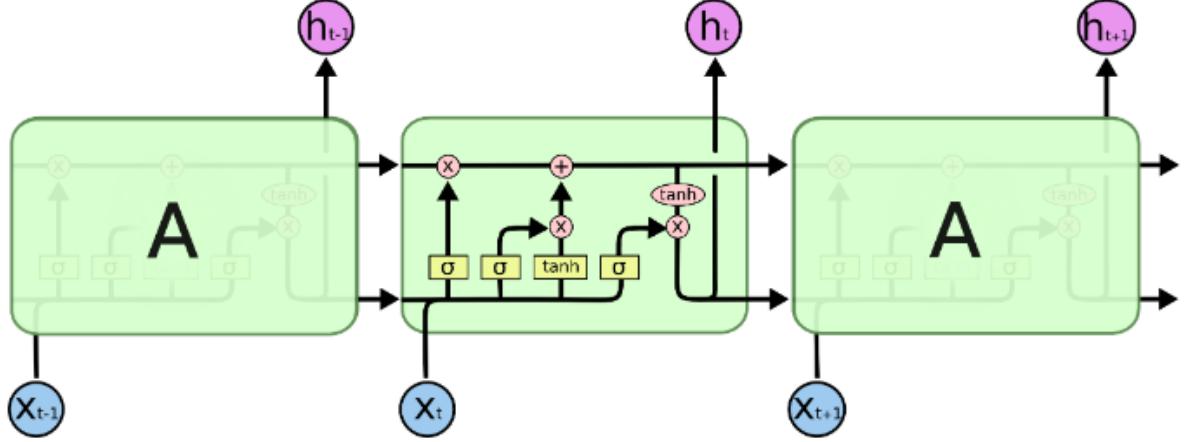


Figure 7.6: The repeating module in an LSTM contains four interacting layers.

7.4.1 LSTM Cell

The following figure shows the operations of an LSTM cell:

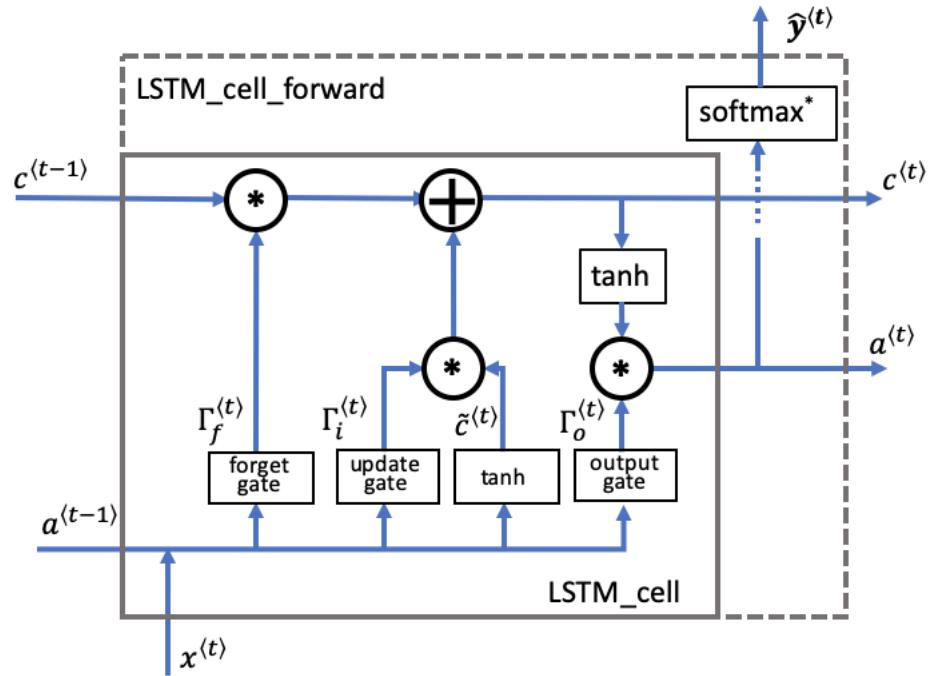


Figure 7.7: LSTM Cell

Forget Gate Γ_f

Assume, we are reading words in a piece of text and plan to use an LSTM to keep track of grammatical structures, such as whether the subject is singular("dog") or plural("dogs"). If the subject changes its state (from a singular word to a plural word), the memory of the previous state becomes outdated, so it is better to forget that outdated state.

The forget gate is a tensor containing values between 0 and 1. If a unit in the forget gate has a value close to 0, the LSTM will forget the stored state in the corresponding unit of the previous cell state. If a unit in the forget gate has a value close to 1, the LSTM will mostly remember the corresponding value in the stored state.

$$\Gamma_f^{<t>} = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f) \quad (7.1)$$

where,

- W_f = forget gate weight W_f
- b_f = forget gate bias b_f
- $\Gamma_f^{<t>} = \text{Forget Gate}$

Explanation of the equation:

- W_f contains the weights that govern the forget gate's behavior.
- The previous time step's hidden state $a^{<t-1>}$ and current time step's input $x^{<t>}$ are concatenated together and multiplied by W_f .
- A sigmoid function is used to make each of the gate tensor's values $\Gamma_f^{<t>}$ range from 0 to 1.
- The forget gate $\Gamma_f^{<t>}$ has the same dimensions as the previous cell state $c^{<t-1>}$.
- Multiplying the tensors $\Gamma_f^{<t>} * c^{<t-1>}$ is like applying a mask over the previous cell state.
- If a single value in $\Gamma_f^{<t>}$ is 0 or close to 0, then the product is close to 0. This keeps the information stored in the corresponding unit in $c^{<t-1>}$ from being remembered for the next time step.
- Similarly, if one value is close to 1, the product is close to the original value in the previous cell state. The LSTM will keep the information from the corresponding unit of $c^{<t-1>}$, to be used in the next time step.

Candidate Value $\tilde{c}^{<t>}$

The candidate value is a tensor containing information from the current time step that may be stored in the current cell state $c^{<t>}$. The parts of the candidate value that get passed on depend on the update gate. The candidate value is a tensor containing values that range from -1 to 1. The tilde " " is used to differentiate the candidate $\tilde{c}^{<t>}$ from the cell state $c^{<t>}$.

$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c) \quad (7.2)$$

Update Gate Γ_i

Update gate can be used to decide what aspects of the candidate $\tilde{c}^{<t>}$ to add to the cell state $c^{<t>}$. The update gate is a tensor containing values between 0 and 1. When a unit in the update gate is close to 1, it allows the value of the candidate $\tilde{c}^{<t>}$ to be passed onto the hidden state

$c^{<t>}$. When a unit in the update gate is close to 0, it prevents the corresponding value in the candidate from being passed onto the hidden state.

$$\Gamma_i^{<t>} = \sigma(W_i[a^{<t-1>}, x^{<t>}] + b_i) \quad (7.3)$$

Explanation of the equation:

- Similar to the forget gate, here $\Gamma_i^{<t>}$, the sigmoid produces values between 0 and 1.
- The update gate is multiplied element-wise with the candidate, and this product $\Gamma_i^{<t>} * \tilde{c}^{<t>}$ is used in determining the cell state $c^{<t>}$.

Cell State $c^{<t>}$

The cell state is the "memory" that gets passed onto future time steps. The new cell state $c^{<t>}$ is a combination of the previous cell state and the candidate value.

$$c^{<t>} = \Gamma_f^{<t>} * c^{<t-1>} + \Gamma_i^{<t>} * \tilde{c}^{<t>} \quad (7.4)$$

Explanation of the equation

- The previous cell state $c^{<t-1>}$ is adjusted (weighted) by the forget gate $\Gamma_f^{<t>}$.
- the candidate value $\tilde{c}^{<t>}$ is adjusted (weighted) by the update gate $\Gamma_i^{<t>}$.

Output Gate Γ_o

The output gate decides what gets sent as the prediction (output) of the time step. The output gate is like the other gates, in that it contains values that range from 0 to 1.

$$\Gamma_o^{<t>} = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o) \quad (7.5)$$

Explanation of the equation

- The output gate is determined by the previous hidden state $a^{<t-1>}$ and the current input x^t
- The sigmoid makes the gate range from 0 to 1.

Hidden State $a^{<t>}$

The hidden state gets passed to the LSTM cell's next time step. It is used to determine the three gates ($\Gamma_f, \Gamma_u, \Gamma_o$) of the next time step. The hidden state is also used for the prediction $y^{<t>}$.

$$a^{<t>} = \Gamma_o^{<t>} * \tanh(c^{<t>}) \quad (7.6)$$

Explanation of the equation

- The hidden state $a^{<t>}$ is determined by the cell state $c^{<t>}$ in combination with the output gate Γ_o .
- The cell state state is passed through the tanh function to rescale values between -1 and 1.
- The output gate acts like a "mask" that either preserves the values of $\tanh(c^{<t>})$ or keep those values from being included in the hidden state $a^{<t>}$.

Prediction $y_{pred}^{<t>}$

The prediction in this use case is a classification, so you'll use a softmax.

$$y_{pred}^{<t>} = \text{softmax}(W_y a^{<t>} + b_y) \quad (7.7)$$

7.5 Bidirectional RNNs

A typical state in an RNN (simple RNN, GRU, or LSTM) relies on the past and the present events. A state at time t depends on the states x_1, x_2, \dots, x_{t-1} and x_t . However, there can be situations where a prediction depends on the past, present, and future events.

For example, predicting a word to be included in a sentence might require us to look into the future, i.e., a word in a sentence could depend on a future event. Such linguistic dependencies are customary in several text prediction tasks.

Thus, capturing and analyzing both past and future events is helpful in the above-mentioned scenarios.

To enable straight (past) and reverse traversal of input (future), Bidirectional RNNs, or BRNNs, are used. A BRNN is a combination of two RNNs - one RNN moves forward, beginning from the start of the data sequence, and the other, moves backward, beginning from the end of the data sequence. The network blocks in a BRNN can either be simple RNNs, GRUs, or LSTMs.

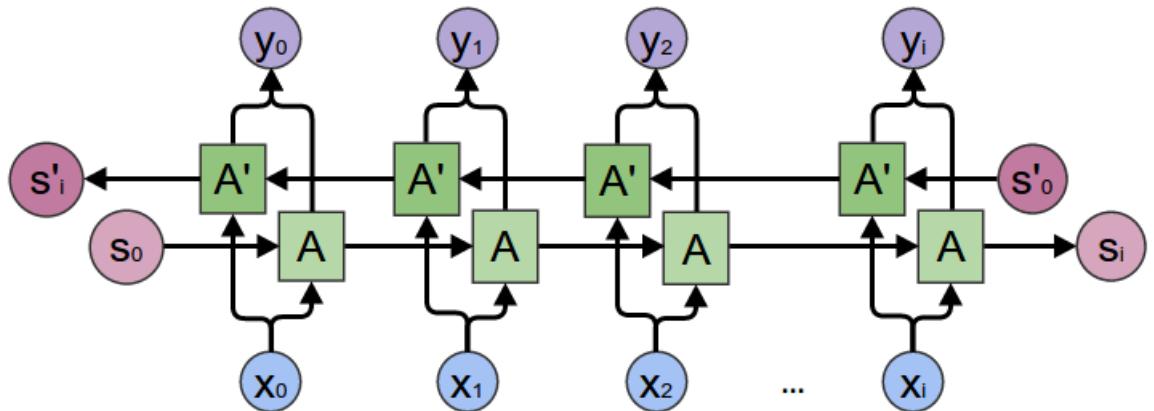


Figure 7.8: Bidirectional RNN

7.6 Sequence Modelling Applications

Some applications of sequence modelling are:

- **Language Modelling** : Auto-generation of next probable word.(Previous sequence of words are sequential data)
- **Image Captioning** (with the help of computer vision): Generating captions for images. (captions are sequential data)
- **Video Frame Prediction** (with the help of computer vision): Predict the subsequent frames of video given the previous ones. (the frames in a video are sequential in nature)
- **Classifying songs** (audio) as Jazz, Rock , Pop etc (genre). Here, audio is of sequential nature.
- **Composing Music** (music is sequential in nature)

Chapter 8

Methodology, Result and Evaluation

8.1 Word Embedding Visualization

Word embeddings vectors were created using gensim word2vec model. Then the obtained 300 dimensional word embeddings vectors were converted into 2-dimensional and 3-dimensional vectors using Principal Component Analysis (PCA). The result obtained using the first 50 vectors were as follow:



Figure 8.1: 2d plot of word embeddings vectors

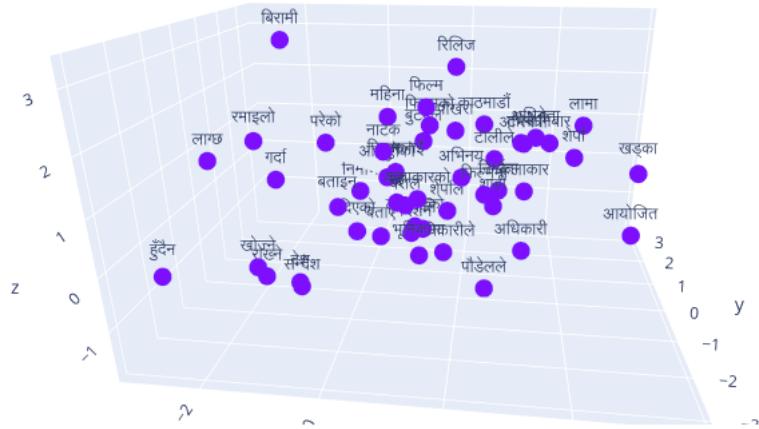


Figure 8.2: 3d plot of word embeddings vectors

8.2 Probabilistic Language Model

For the development of a probabilistic language model, at first vocabulary was created from the training dataset. At first, the corpus was split into a list of sentences. Text preprocessing involves:

1. remove all non-devanagari letters
 2. remove numbers from the corpus

Then word tokenization was created which was used to create vocabulary from the tokenized word using the constraints of minimum frequency. Then, the n-gram count list was generated in which the count of various n-grams were stored. For the creation of our model, unigram, bigram, trigram, 4-gram and 5-gram were created. Among these, the unigram, bigram and trigram performed better.

Hence, we used only unigram, bigram and trigram for the probability estimation of the next word.

```

POST http://127.0.0.1:8000/api/n-gram/
Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies
Body
  body: જરીએ
  n_from_ngram: 0
Body Cookies Headers (10) Test Results
Pretty Raw Preview Visualize JSON
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
  "Predicted Tokens": [
    {
      "token": "જરીએ",
      "prob": 0.000371089642888067
    },
    {
      "token": "એ",
      "prob": 0.00016000551191816238
    },
    {
      "token": "થિયા",
      "prob": 0.00016000551191816238
    },
    {
      "token": "એકડા",
      "prob": 0.00013606620239229122
    },
    {
      "token": "મની",
      "prob": 8.658758334054896e-05
    }
  ]

```

Figure 8.3: Next word prediction using Probabilistic LM

8.3 Transformer based language model

For this language model too, the vocabulary was created using the same method as in the Probabilistic language model. Then the transformer architecture with 4 multi-head attention and 4 encoder layers were developed for language model training. Cross Entropy loss was used as a loss function and SGD was used as an optimizing function. The perplexity obtained is 568.5.

Input to the LM is current words and num_words to be predicted. Then the language model generate probable num_words as shown in fig: 8.4

```

POST http://127.0.0.1:8000/api/transformer-lm/
Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies
Body
  body: જરીએ
  num_words: 10
Body Cookies Headers (10) Test Results
Pretty Raw Preview Visualize JSON
1
2
3
4
  "Input String": "જરીએ",
  "Generated String": "તથા અસ્તુરી મનાલ્ય ર મહાલકા નિવાયદુર્ઘાતાન શ્વાસન શિરખાતીન છું।"

```

Figure 8.4: Next word prediction using Transformer based LM

8.4 Context Based Spelling Correction

Noisy channel model is used for the development of spelling correction model. The channel model is trained based on Brill and Moore model using unsupervised data from corpus. The transformer based language model is used to determine the prior function. The candidate sentences are found using minimum edit distance based on following equation. Then the word that maximizes the channel model and prior is chosen.

$$\hat{w} = \arg \max_{w \in C} P(x|w)P(w) \quad (8.1)$$

here, $P(x | w)$ = channel model and $P(w)$ = prior

8.5 Backend

We have developed backend using Django and Django Rest Framework. For the API testing, Postman was used.

Chapter 9

Sentimental Classification Model and Evaluation

9.1 Sentimental Data Exploration

We obtained the sentimental classification datasets from Kaggle and HuggingFace. Some sample of the dataset were as follow:

	Data	Label
0	यो समान राम्रो रहेछ	1
1	समान राम्रो रहेछ	1
2	राम्रो रहेछ	1
3	यो घडी मलाई साँचिकै सुहाउछ म यसलाई खरीद गर...	1
4	साँचिकै सुहाउछ	1

Figure 9.1: Sentimental Classification Dataset

Dataset obtained from kaggle and huggingface has 2186 and 7985 unique entries respectively. The dataset distribution of kaggle is as follow:

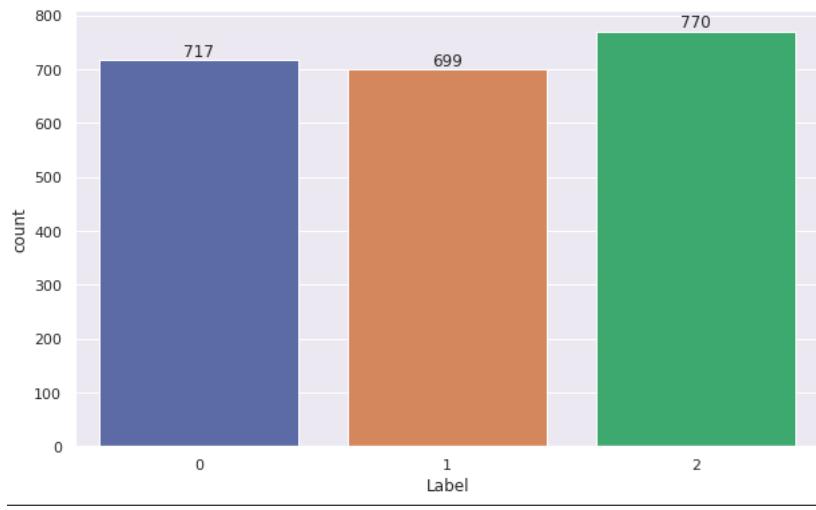


Figure 9.2: Kaggle Dataset Label Count plot

Both datasets were combined and the resulting data count plot is as follow:

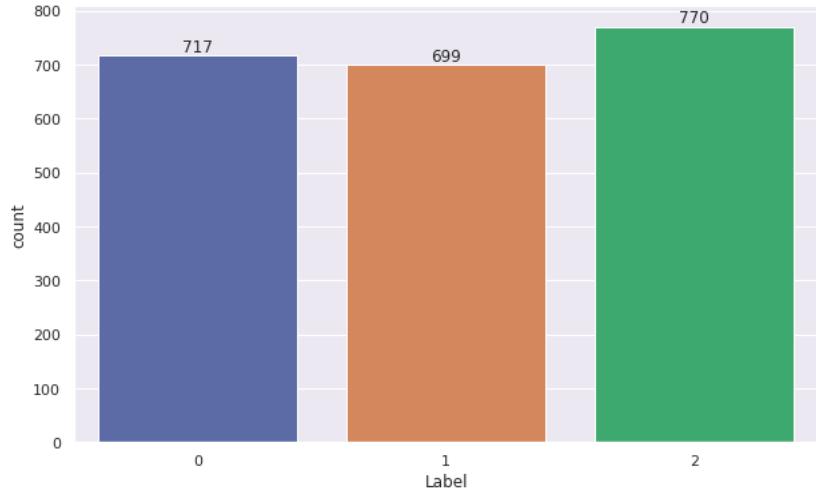


Figure 9.3: Combined Dataset Label Count plot

9.2 Data Preprocessing

The text data of our dataset was preprocessed as follow:

1. Tokenized the sentence using **nepalitokenizer**
2. Removed the nepali stopwords. Stop words were obtained from **nltk.corpus.stopwords**
3. Nepali words were stemmed using **nepali_stemmer.stemmer.NepStemmer**

9.3 Train test split

80%, 10% and 10% of the datasets were used for training, validation and testing of the model.

9.4 Tokenization and One Hot Encoding

Tokenization was performed using `keras.preprocessing.text.Tokenizer` and padded using `keras.preprocessing.sequence.pad_sequences`. The UNIQUE_WORD_COUNT was selected to be 3179 and MAX_PAD_LENGTH = 10 with the help of following graph.

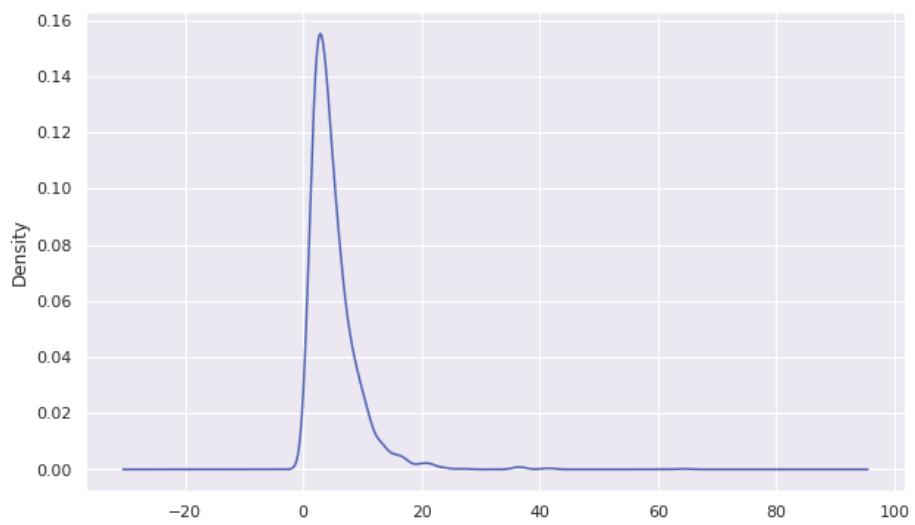


Figure 9.4: Length Series Count Plot : Length 10 seems to be appropriate for padding length from the plot.

9.5 Sentimental Classification Model

Only kaggle dataset was used for the training and evaluation of the classification model version 1 and version 2.

9.5.1 Sentimental Classification Model Version 1

The summary of the sentimental classification version 1 is as follow:

Layer (type)	Output Shape	Param #
<hr/>		
embedding (Embedding)	(None, 10, 64)	203456
lstm (LSTM)	(None, 10)	3000
dense (Dense)	(None, 20)	220
dropout (Dropout)	(None, 20)	0
dense_1 (Dense)	(None, 3)	63
<hr/>		
Total params: 206,739		
Trainable params: 206,739		
Non-trainable params: 0		

Figure 9.5: Sentimental Model Version 1 Summary

The training and test loss and accuracy variation with epochs are shown below:

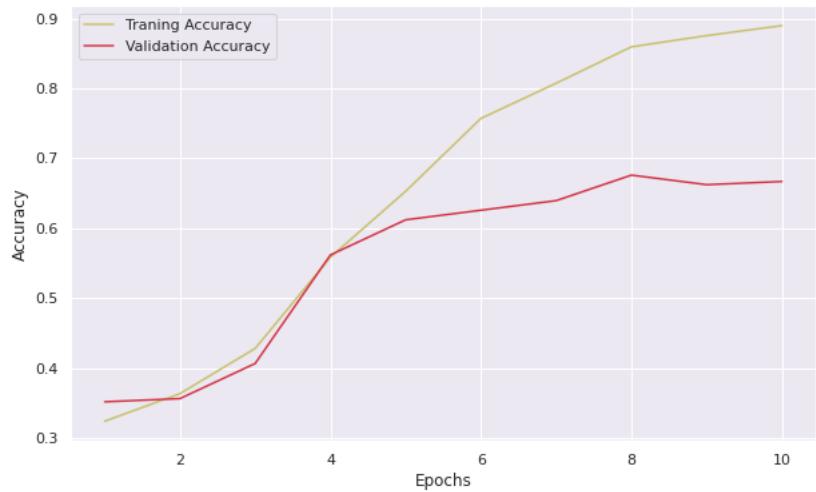


Figure 9.6: Sentimental Model Version 1 Accuracy Curve

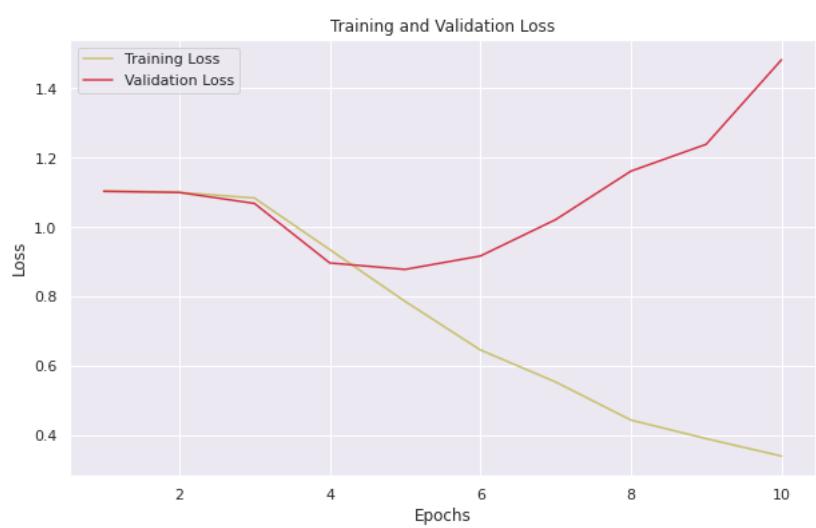


Figure 9.7: Sentimental Model Version 1 Loss Curve

The confusion matrix heatmap plot and classification report are shown below:

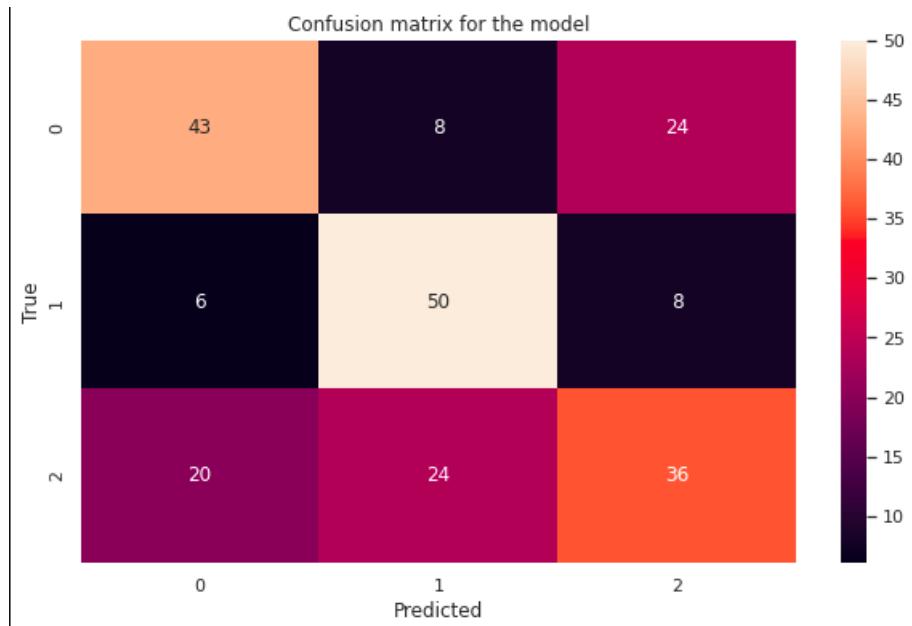


Figure 9.8: Sentimental Model Version 1 Confusion Matrix

	precision	recall	f1-score	support
Negative	0.62	0.57	0.60	75
Positive	0.61	0.78	0.68	64
Neutral	0.53	0.45	0.49	80
accuracy			0.59	219
macro avg	0.59	0.60	0.59	219
weighted avg	0.59	0.59	0.58	219

Figure 9.9: Sentimental Model Version 1 Classification Report

9.5.2 Sentimental Classification Model Version 2

The summary of the sentimental classification version 2 is as follow:

Layer (type:depth-idx)	Output Shape	Param #
-Embedding: 1-1	[-1, 10, 128]	406,912
-LSTM: 1-2	[-1, 10, 64]	182,784
-Dropout: 1-3	[-1, 640]	--
-ReLU: 1-4	[-1, 640]	--
-Linear: 1-5	[-1, 64]	41,024
-ReLU: 1-6	[-1, 64]	--
-Dropout: 1-7	[-1, 64]	--
-Linear: 1-8	[-1, 16]	1,040
-ReLU: 1-9	[-1, 16]	--
-ReLU: 1-10	[-1, 16]	--
-Linear: 1-11	[-1, 3]	51
<hr/>		
Total params: 631,811		
Trainable params: 631,811		
Non-trainable params: 0		
Total mult-adds (M): 0.63		
<hr/>		
Input size (MB): 0.08		
Forward/backward pass size (MB): 0.02		
Params size (MB): 2.41		
Estimated Total Size (MB): 2.50		
<hr/>		

Figure 9.10: Sentimental Model Version 2 Summary

The training and test loss and accuracy variation with epochs are shown below:

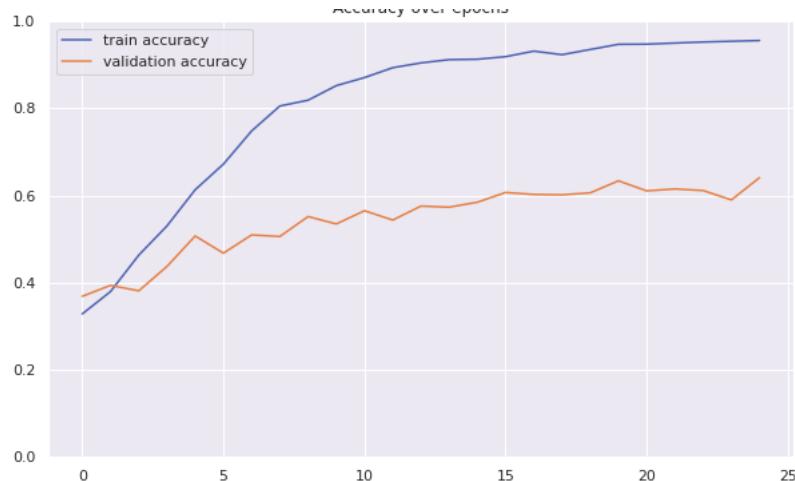


Figure 9.11: Sentimental Model Version 2 Accuracy Curve

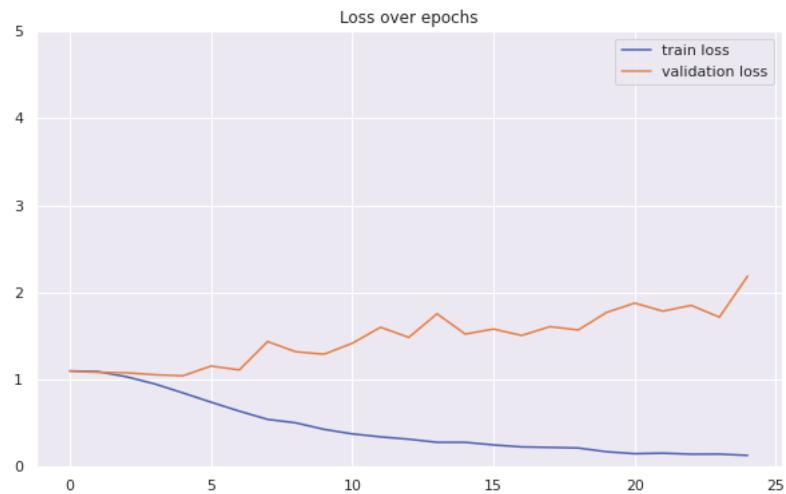


Figure 9.12: Sentimental Model Version 2 Loss Curve

The confusion matrix heatmap plot and classification report are shown below:

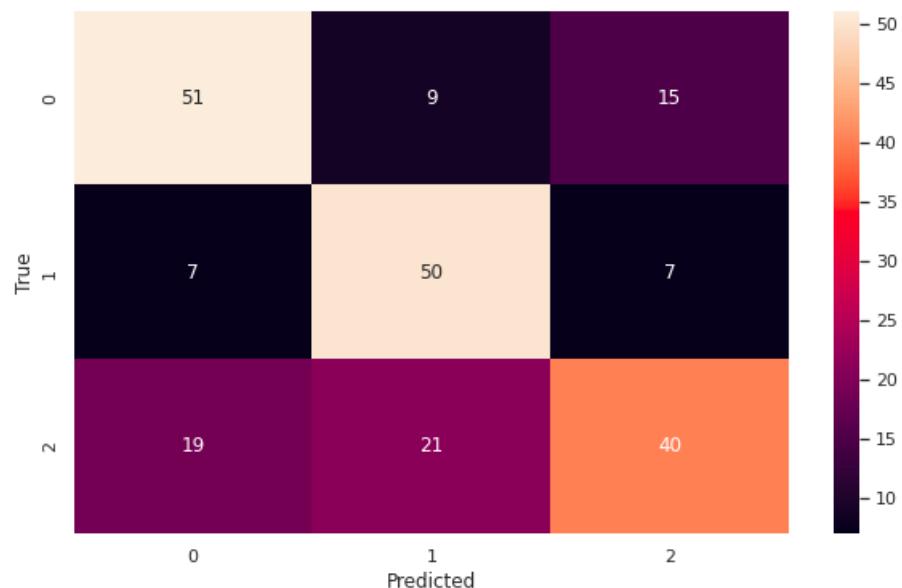


Figure 9.13: Sentimental Model Version 2 Confusion Matrix

	precision	recall	f1-score	support
Negative	0.66	0.68	0.67	75
Positive	0.62	0.78	0.69	64
Neutral	0.65	0.50	0.56	80
accuracy			0.64	219
macro avg	0.64	0.65	0.64	219
weighted avg	0.65	0.64	0.64	219

Figure 9.14: Sentimental Model Version 2 Classification Report

9.5.3 Bert Model

We found the dataset from HuggingFace and merged it with Kaggle for the training of this dataset. The basic preprocessing methods were same as in the previous model.

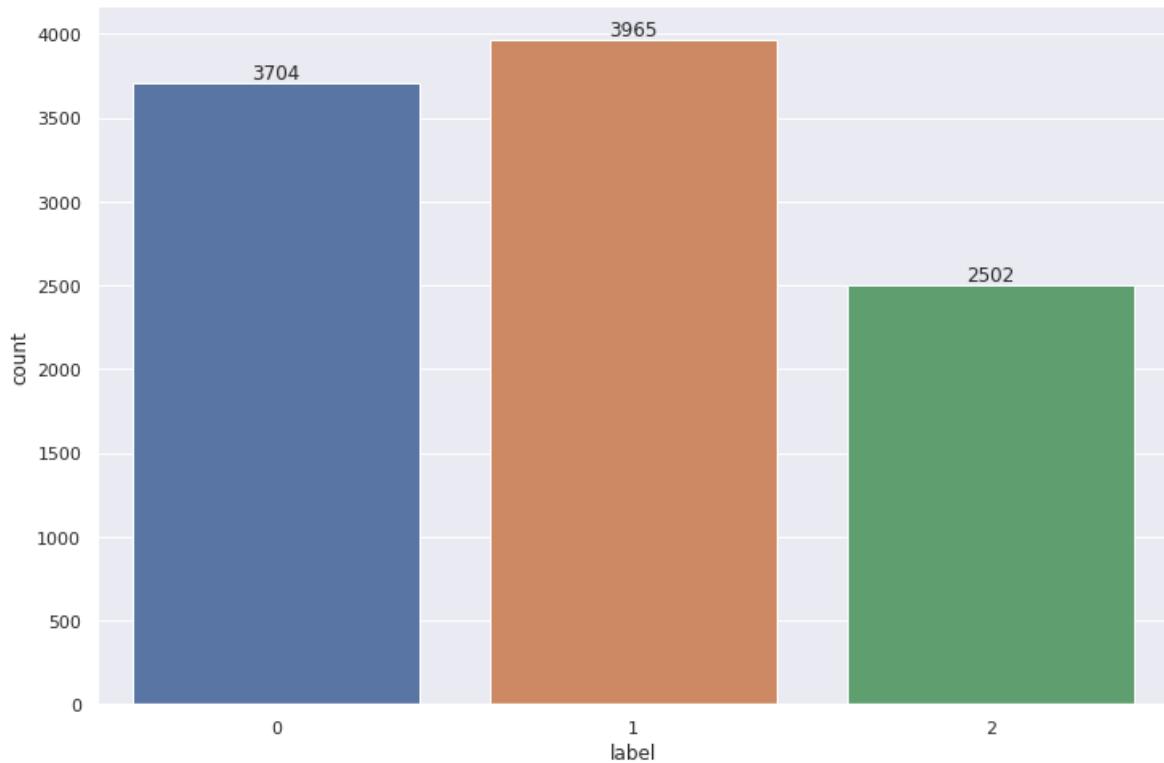


Figure 9.15: Kaggle and HuggingFace Sentimental Data Count plot

For the development of this model, pretrained tokenizer **bert-base-multilingual-cased** and pretrained model **bert-base-multilingual-cased** from the **transformers** module.

The classification report and the confusion matrix were as follow:

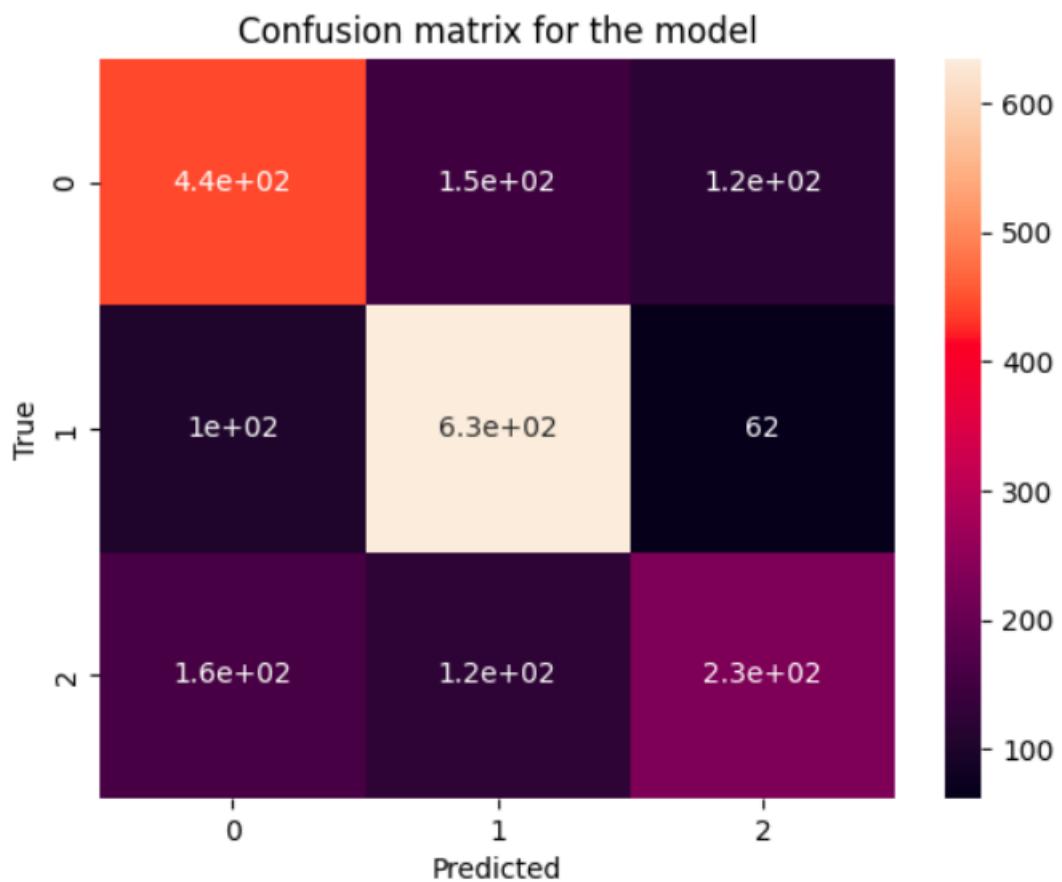


Figure 9.16: Sentimental Bert Model Confusion Matrix

	precision	recall	f1-score	support
Negative	0.63	0.62	0.63	709
Positive	0.70	0.80	0.75	797
Neutral	0.56	0.45	0.50	510
accuracy			0.65	2016
macro avg	0.63	0.62	0.62	2016
weighted avg	0.64	0.65	0.64	2016

Figure 9.17: Sentimental Bert Model Classification Report

Chapter 10

Frontend

The frontend was made using React JS. In our frontend, we have separate interfaces for each tasks and we can easily navigate each of the interfaces through the navbar. Also, we can also toggle between light mode and dark mode too. Keeping in mind about the user experience, we have also made the cursor appear inside the textbox without needing the user to click inside the textbox to type. The text typed inside the text box gets transliterated to nepali language for which we used **Google Input Tools Transliteration API**. This makes user able to type directly in nepali langauge and perform specific actions without need of any external tools for providing nepali language input. We have also made the UI as simple as possible and presented options in a visible way so users don't have to remember where they are and what they are able to do. We have also made use of proper mouse pointers as well to let the user know when they are able to click certain button and when they are not. We have kept the screens and dialogues focused and minimal to maximize visibility and clarity and used terminology and language familiar to the user.

Our default mode is light mode and default interface/page is home page.

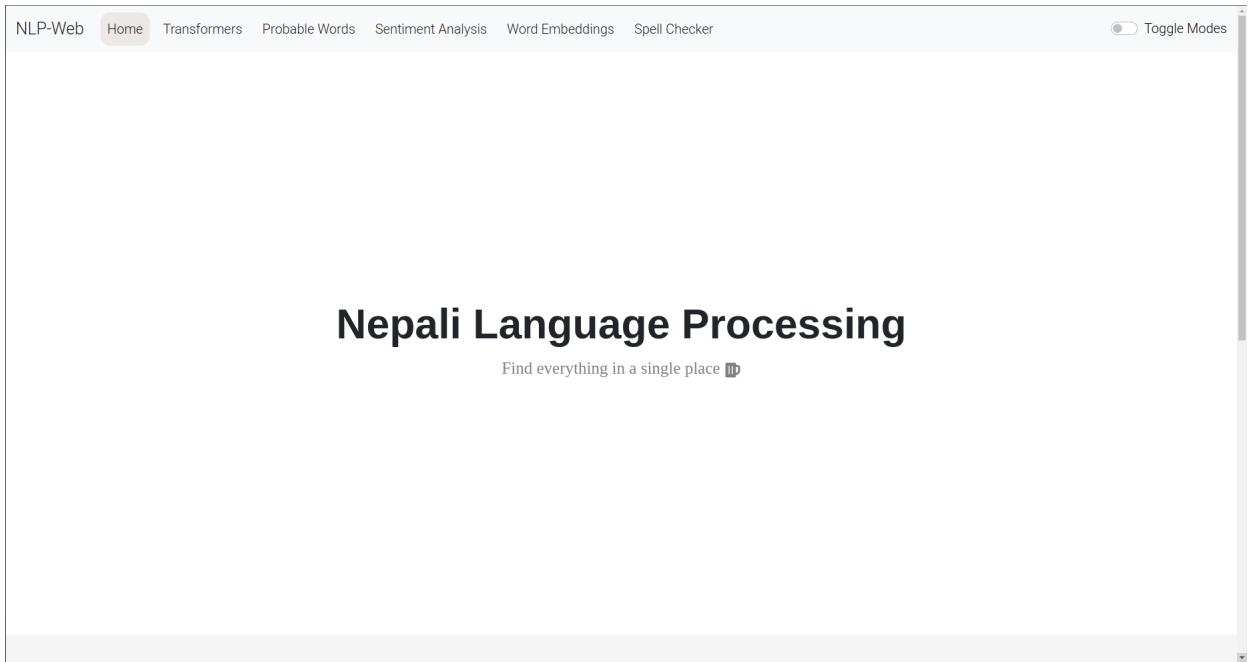


Figure 10.1: Homepage Light Mode

We can navigate to transformer through navbar which provides you with an interface that lets you to see the text generated based on your input text and number of words to be generated. By default the number of words is set to 3, with freedom for user to change it if they want as they like. Just after giving all the inputs as you like, you can now press the button. The request is then sent to the API of our backend and the response is displayed as shown in the figure below.

Text Generation Using Transformer Model

The screenshot shows a web form for generating text using a transformer model. At the top, there is a text input field containing the Nepali text 'मेरो भनाइ'. Below it is another text input field labeled 'Enter number of words (Default = 3)' with the value '4' entered. A 'Generate Text' button is located below these fields. Underneath the button, the 'Input String' is displayed as 'मेरो भनाइ' and the 'Generated String' is displayed as 'उद्धृत गर्दै अधिवक्ता ढकालले'.

Figure 10.2: Transformer based LM Frontend

Likewise, we can navigate to Probable words through navbar which provides you with an interface that lets you input text in text field where you can directly type just after navigating to it through navbar. The request is sent to the API of our backend and the response is displayed as shown in the figure below. Now just as before, enter the text you like and press the button. The request is

then sent to the API of our backend. Based on the response, the probability if the next 5 possible text with their probabiltiy are shown in the table along with their visualization with the help of progress bar.

Next Probable Words Using N-Gram Model

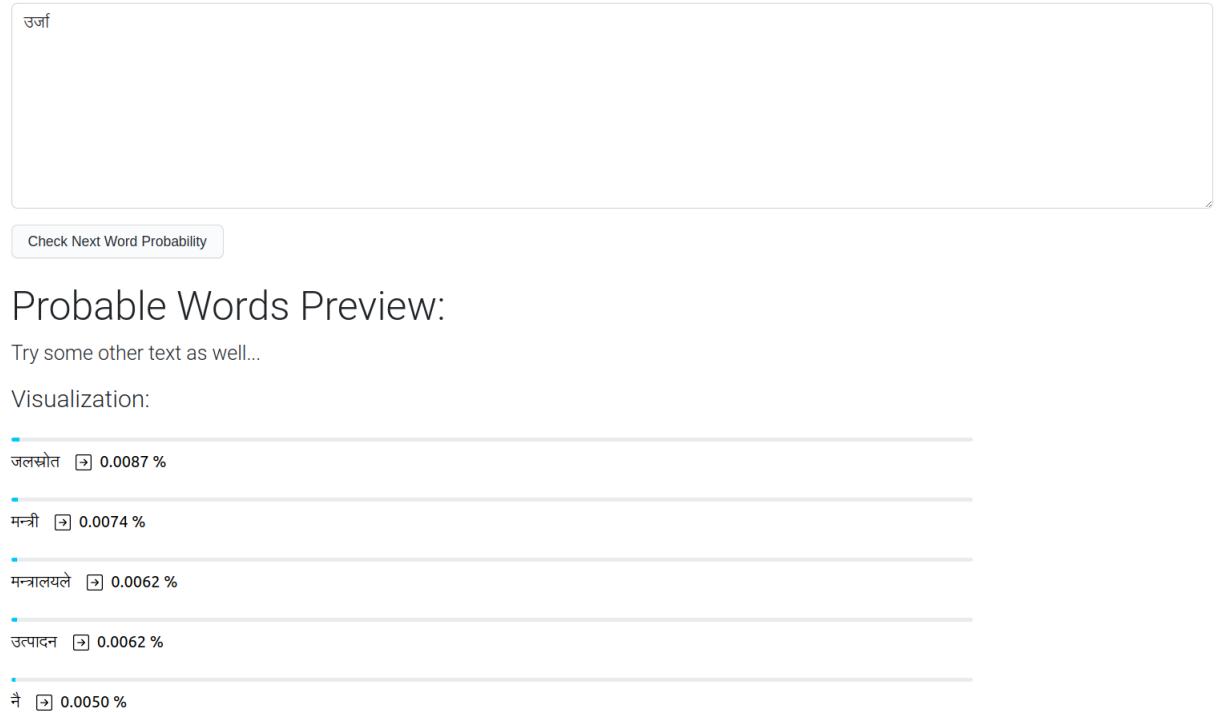
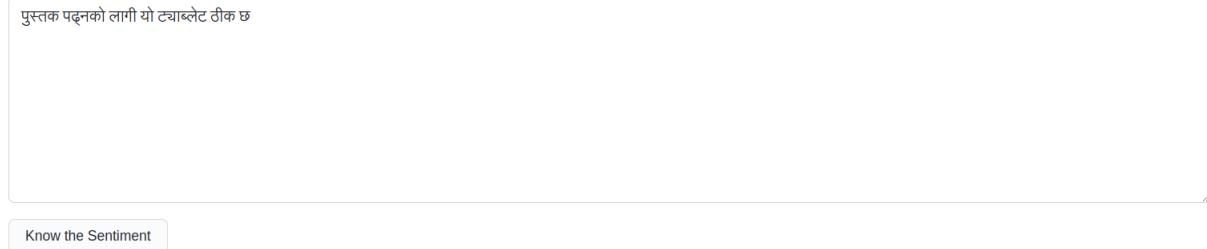


Figure 10.3: Proabilistic Language Model Frontend

Similarly, you will get another interface after navigating to sentiment analysis. The sentiment type of given text is displayed after you click the button along with the probability of its sentiments with progress bar.

Sentiment Analysis Using V2 Model



Sentiment:

Type: Positive

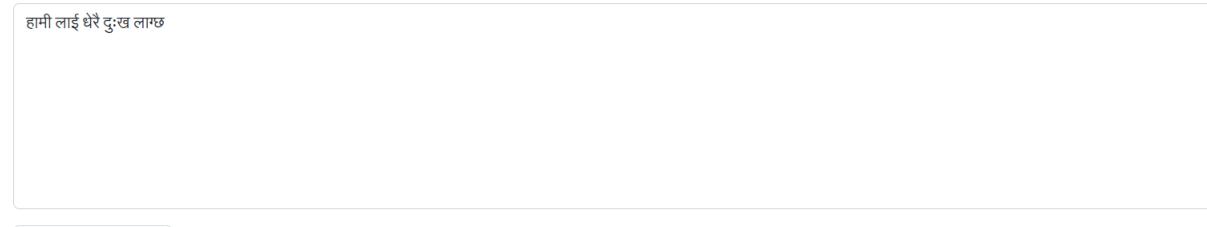
Negative 0.002

Positive 0.996

Neutral 0.002

Figure 10.4: Positive Sentiment Prediction

Sentiment Analysis Using V2 Model



Sentiment:

Type: Negative

Negative 0.941

Positive 0.019

Neutral 0.040

Figure 10.5: Negative Sentiment Prediction

Sentiment Analysis Using V2 Model

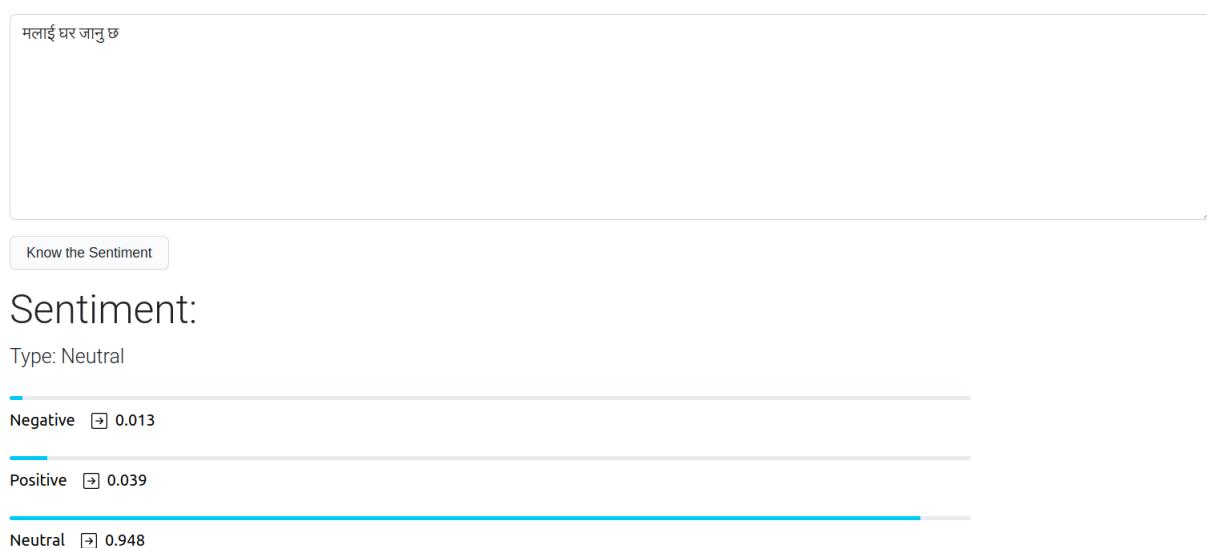


Figure 10.6: Neutral Sentiment Prediction

In word embeddings you can either choose to see 2d or 3d word embeddings visualization graph just by clicking buttons. The 2d graph is made using the canvjs.js graphing library and 3d graph is made using plotly.js library.

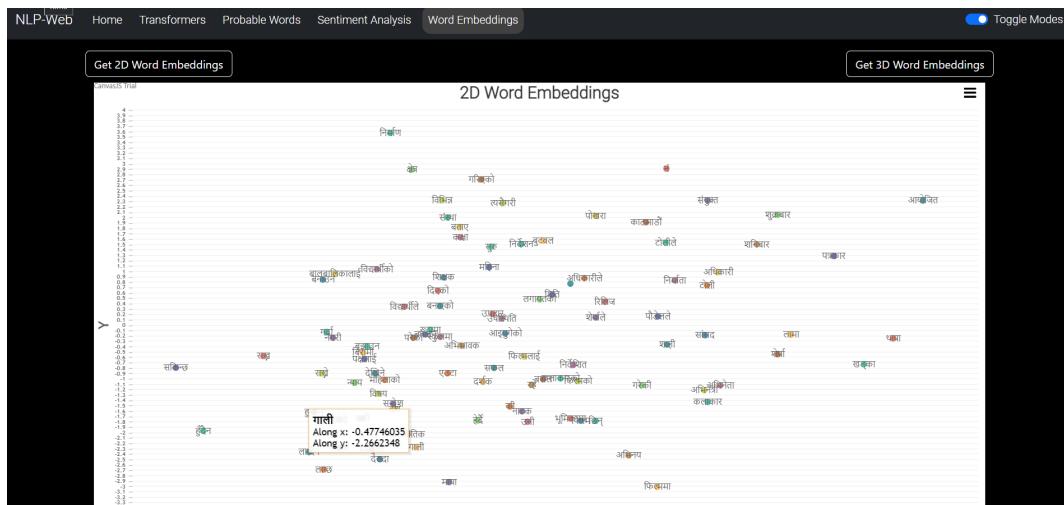


Figure 10.7: Word Embedding 2d plot Frontend

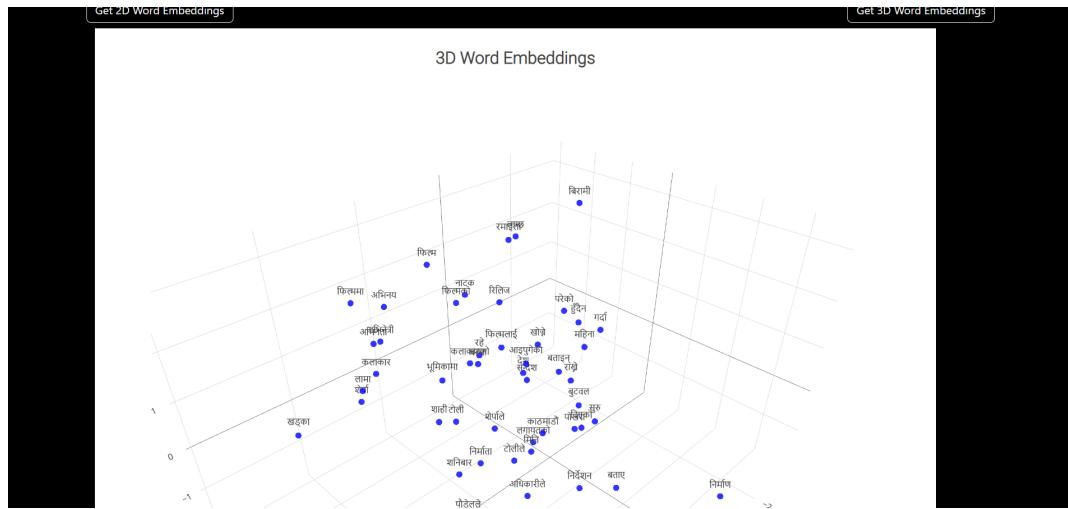


Figure 10.8: Word Embedding 3d plot Frontend

In spell checker, we can get auto corrected output of the given text. In addition to that, we can also manually correct the text based on the choices provided. By manually choosing the choices provided for each text, the spelling correction is more likely to be accurate. Below is the screenshot for spelling correction.

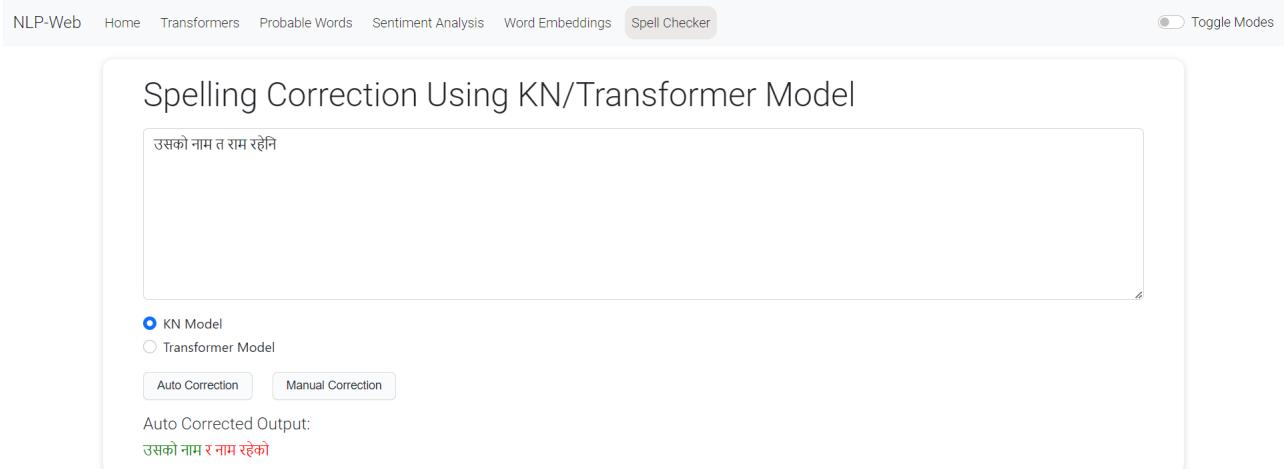


Figure 10.9: Auto Spelling Correction

Spelling Correction Using KN/Transformer Model

उसको नाम त राम रहेनि

KN Model
 Transformer Model

Click on the text to manually correct the sentences.

उसको नाम त राम रहेको

Choices:
नाम
राम
काम

Figure 10.10: Manual Spelling Correction

Chapter 11

Conclusion

In this project, we have explored various aspects of Nepali language processing, including word embedding visualization, sentimental classification, text generation, and spelling correction. Through our research and experimentation, we have gained a deeper understanding of the complexities of Nepali language and the challenges it poses for natural language processing.

We have leveraged cutting-edge technologies such as probabilistic and transformer language models to develop accurate and efficient language processing systems that can be used for text generation and spelling correction system. Additionally, our collaboration with our supervisor, Dr. Basanta Joshi, and our mentor, Prasid Karki, has enabled us to gain invaluable insights and guidance in this field.

We hope that this project will serve as a valuable resource for researchers and developers who are interested in Nepali language processing. We believe that the systems and models we have developed can be further refined and enhanced to improve their accuracy and effectiveness. Ultimately, our goal is to contribute to the advancement of natural language processing and to enable greater accessibility and understanding of the Nepali language.

Bibliography

- [1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.
- [2] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, “Language models are unsupervised multitask learners,” *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.
- [3] Y. Bengio, R. Ducharme, and P. Vincent, “A neural probabilistic language model,” *Advances in neural information processing systems*, vol. 13, 2000.
- [4] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” *Advances in neural information processing systems*, vol. 26, 2013.
- [5] S. Timilsina, M. Gautam, and B. Bhattacharai, “Nepberta: Nepali language model trained in a large corpus,” in *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*, pp. 273–284, 2022.