Next Word Prediction Model

A MINOR PROJECT

Submitted in Partial Fulfillment of the Requirement for the Award of the Degree of

BACHELOR OF TECHNOLOGY (HONS) CSE

In

Artificial Intelligence

SUBMITTED TO

SUBMITTED BY

Aneesh Jain (22BTA3DSC10001)

UNDER THE SUPERVISION OF

Dr. Gourav Shrivastava
SCHOOL OF COMPUTER TECHNOLOGY

SANJEEV AGRAWAL GLOBAL EDUCATIONAL UNIVERSITY,

BHOPAL

AUTUMN 2025-26

SANJEEV AGRAWAL GLOBAL EDUCATIONAL UNIVERSITY, BHOPAL

SCHOOL OF COMPUTER TECHNOLOGY

CANDIDATE'S DECLARATION

I, Aneesh Jain**,** Student of **Bachelor of Technology (Hons) CSE (Artificial Intelligence), Sanjeev Agrawal Global Educational University, Bhopal** Academic Year 2025-26 hereby declare that the work presented in the Minor Project report entitled "**Next Word Prediction Model"** is outcome of my own bonafide work, which is correct to the best of my knowledge and this work has been carried out taking care of Engineering Ethics. The work presented does not infringe any previous work and has not been submitted to any University for the award of any degree.

Aneesh Jain

22BTA3DSC10001

SANJEEV AGRAWAL GLOBAL EDUCATIONAL UNIVERSITY, BHOPAL

DEPARTMENT OF ADVANCED COMPUTING

CERTIFICATE

This is to certify that the work embodies in this Minor Project entitled **Next Word Prediction Model** being submitted by **Aneesh Jain (22BTA3DSC10001)** for partial fulfillment of the requirement for the award of degree of **Bachelor of Technology (Hons) CSE (Artificial Intelligence)** during the academic year 2025-26 is a record of real piece of work, carried out by him under my supervision and guidance in the **School of Computer Technology, Sanjeev Agrawal Global Educational University, Bhopal (M.P.)**

SANJEEV AGRAWAL GLOBAL EDUCATIONAL UNIVERSITY, BHOPAL

DEPARTMENT OF ADVANCED COMPUTING

ACKNOWLEDGEMENT

At the outset, I would like to thank the Almighty, who made everything possible. Writing this project synopsis report would not have been possible without the support of several people whom we need to wholeheartedly thank. I express a deep sense of gratitude to my Supervisor **Dr. Gourav Shrivastava, School of Computer Technology,** for the valuable and inspirational guidance from the initial to the final level that enabled me to develop an understanding of this Project work.

I would like to give my sincere thanks to **Dr. Gourav Shrivastava, School of Computer Technology,** for their kind help, encouragement, and co-operation throughout my Project Synopsis period, and I owe my special thanks to **Prof.(Dr.) Gourav Shrivastava** for their guidance and suggestions during the Project work.

Lastly, I want to thank my parents, friends, and all those people who have contributed to my project synopsis directly or indirectly for their moral and psychological support.

Aneesh Jain (22BTA3DSC10001)

TABLE OF CONTENTS

ABSTRACT

This report details the design, implementation, and evaluation of a Next Word Prediction model. The project leverages concepts from Natural Language Processing (NLP) and Deep Learning to create a system capable of predicting the most likely word to follow a given sequence of text. The model is built in Python, primarily utilizing the Keras deep learning library, which runs on top of TensorFlow. The core of the model is a Recurrent Neural Network (RNN) architecture, specifically a Long Short-Term Memory (LSTM) network, which is well-suited for learning from sequential data like text. This report covers the complete project lifecycle, including data acquisition, text preprocessing, model architecture design, training, and evaluation. The final model is trained on a text corpus (1661-0.txt) and, given a sequence of five words, can successfully predict a probable subsequent word.

CHAPTER 1

INTRODUCTION

1.1 Motivation

The primary motivation for this project stems from the ubiquitous nature of predictive text in modern digital life. From the autocomplete features on smartphone keyboards and search engines (like Google Search) to intelligent code completion in software development environments (IDEs), next word prediction technology is fundamental to improving user experience, increasing typing efficiency, and reducing errors. This project seeks to explore the foundational principles of this technology by building a functional, word-level prediction model from the ground up, providing practical insights into the capabilities and challenges of applied deep learning in NLP.

1.2 Scope

The scope of this project is to develop a proof-of-concept word prediction model. This includes:

- Utilizing a single, static text corpus (1661-0.txt) for training and validation.
- Implementing word-level tokenization and vocabulary creation.
- Constructing sequences of a fixed length (5 words) as input to predict the next word.
- Building, training, and evaluating a single-layer LSTM model.
- Analyzing the model's performance based on its accuracy and loss metrics.

This project's scope does not extend to building a real-time application, creating a user interface, or training a large-scale model on massive, web-scale datasets.

1.3 Objective

The primary objective of this project is to design, build, and train an LSTM neural network that can, given a sequence of five words, predict the next most probable word from its learned vocabulary.

Secondary objectives include:

- To understand and implement the data preprocessing pipeline required for text-based deep learning.
- To gain practical experience with the Keras library for building and training neural networks.
- To analyze and interpret the model's training and validation performance.
- To create a functional prediction script that demonstrates the model's capabilities on new seed text.

1.4 Application

The principles and architecture used in this project are a simplified version of systems with wide-ranging real-world applications. These include:

- **Smart Keyboards:** Powering the predictive text and autocomplete suggestions on mobile devices.
- **Search Engines:** Suggesting query completions to users as they type.
- **Email Clients:** Features like "Smart Compose" that suggest entire phrases or sentences.
- **Assistive Technology:** Helping individuals with communication difficulties to compose messages more quickly.
- **Creative Tools:** Aiding writers or musicians by suggesting the next word, phrase, or note.

CHAPTER 2

LITERATURE SURVEY

2.1 Literature Survey

The task of language modeling, or predicting the next word, has evolved significantly. Early approaches relied on statistical models, most notably **n-grams**. An n-gram model predicts the next word by looking at the previous 'n-1' words and calculating the probability of each word in the vocabulary appearing next based on its frequency in the training corpus. While effective for small 'n' (like bigrams or trigrams), these models suffer from the "curse of dimensionality" and data sparsity. They fail to capture long-range dependencies in language and cannot generalize to n-grams not seen during training.

The advent of **Recurrent Neural Networks (RNNs)** provided a more powerful solution. RNNs process sequences by maintaining an internal "hidden state," allowing them to, in theory, capture context from arbitrarily far back in the sequence. However, simple RNNs often struggle with the **vanishing gradient problem**, making it difficult to learn long-range dependencies.

This limitation was famously addressed by **Long Short-Term Memory (LSTM)** networks, a special type of RNN introduced by Hochreiter & Schmidhuber (1997). LSTMs introduce a "cell state" and a series of "gates" (input, forget, and output) that meticulously control the flow of information. This architecture allows the network to learn which information to store, which to discard, and what to output, enabling it to effectively capture and model long-term dependencies in text. This project's model is directly based on this architecture.

Furthermore, this model uses an **Embedding Layer**. This is a technique popularized by models like Word2Vec (Mikolov et al., 2013), where words are not treated as discrete, arbitrary indices but are mapped to a dense, low-dimensional vector space. The network learns this mapping during training, placing words with similar meanings close to each other in this vector space, which greatly improves the model's ability to generalize.

2.2 Conclusion

The literature review confirms that for a sequential, context-dependent task like next word prediction, an LSTM network is a highly effective and well-established choice. It directly addresses the shortcomings of both traditional statistical models (n-grams) and simple RNNs. The chosen methodology of using an Embedding layer followed by an LSTM layer is a standard, robust, and powerful approach for this problem.

CHAPTER 3

PROBLEM STATEMENT

3.1 Problem Statement

The formal problem is to develop a computational model that can effectively learn the intricate statistical patterns, syntactic rules, and linguistic structures embedded within a given text corpus. This project constrains the problem to a specific, well-defined task: given a sequence of N antecedent words (where N=5, as defined by WORD_LENGTH in the code), the model must process this input sequence and predict the single word that is most likely to follow.

This is fundamentally a large-scale, multi-class classification problem. The "input features" are the sequences of 5 words, and the "classes" are all the possible words that could come next. The challenge is that the number of "classes" is equal to the entire size of the vocabulary, which in this project is 8,201 unique words.

Therefore, the model's task is not to generate *one* word, but to output a probability distribution across all 8,201 words. This distribution represents the model's confidence for each word being the correct next word. The final prediction is then determined by selecting the word that is assigned the highest probability. The core challenge is to build a model that can, with better-than-chance accuracy, assign the highest probability to the word that is contextually and grammatically correct, based only on the five words it has just "seen".

CHAPTER 4

MINIMUM REQUIREMENTS

4.1 Minimum Software Requirement

- **Python:** Version 3.x
- **TensorFlow:** The backend engine for Keras.
- **Keras:** The high-level deep learning library used to build the model.
- **NLTK (Natural Language Toolkit):** Used for text tokenization.
- **NumPy:** Used for all numerical array operations.
- **Matplotlib:** Used for plotting the training/validation accuracy and loss.
- **Pickle:** Used for saving the training history object.
- **Jupyter Notebook (Recommended):** As the provided code suggests, an interactive environment like Jupyter is ideal for developing and testing the model.

4.2 Minimum Hardware Requirement

- **CPU:** A standard multi-core processor (e.g., Intel Core i5 or AMD Ryzen 5). The model can be trained on a CPU, but it will be relatively slow.
- **RAM:** 8 GB+ of system RAM is recommended to handle the dataset, vocabulary, and model in memory.
- **GPU (Highly Recommended):** For a task like this, a CUDA-enabled NVIDIA GPU (e.g., GeForce GTX 1650 or higher) is highly recommended. It will accelerate the model training process by orders of magnitude compared to a CPU.

CHAPTER 5

METHODOLOGY USED

5.1 Methodology

The project follows a systematic methodology, broken down into the following stages:

1. **Data Acquisition:** The project begins by loading a text corpus (1661-0.txt) from disk.
2. **Text Preprocessing:** The entire text is converted to lowercase to ensure uniformity. A RegexpTokenizer from the NLTK library is used to extract all word tokens, effectively removing punctuation and symbols.
3. **Vocabulary Creation:** A comprehensive vocabulary is built by identifying all unique words in the tokenized corpus. This results in a vocabulary size of 8,201 words. Two mapping dictionaries, word_to_index and index_to_word, are created to convert words to numerical indices and back.
4. **Sequence Generation:** The tokenized corpus is transformed into a dataset of input-output pairs. A sliding window with a length of 5 words (WORD_LENGTH = 5) moves across the text. At each step, the 5-word sequence is stored as the input (prev_words), and the single word immediately following it is stored as the target (next_words).
5. **Vectorization:** The dataset is prepared for the neural network.
    - The input sequences (X) are converted into a 2D NumPy array of integers, where each word is replaced by its corresponding index from the word_to_index dictionary.
    - The target words (Y) are converted into a 2D NumPy array and one-hot encoded. This means each target is represented by a vector of length 8,201 (the vocab_size), which is all zeros except for a '1' at the index corresponding to the correct word.
6. **Model Training:** The compiled Keras model is trained on the vectorized data using the .fit() method. 5% of the data is held back for validation

(validation_split=0.05) to monitor the model for overfitting. The EarlyStopping callback is used to stop training if the validation loss does not improve for 5 consecutive epochs.

7. **Evaluation:** The model's performance (accuracy and loss) on both the training and validation sets is tracked during training and saved to a history object. These results are then plotted using Matplotlib.
8. **Prediction:** A prediction function is defined to take a 5-word seed text, preprocess it, feed it to the trained model, and return the predicted next word.

5.2 Proposed Algorithm

The core of the project is a Sequential Keras model, which defines a linear stack of layers. The algorithm's architecture is as follows:

1. **Embedding Layer:** Embedding(input_dim=8201, output_dim=50, input_length=5)
    - This is the first layer. It takes the integer-encoded input sequences of length 5.
    - It maps each of the 8,201 unique words to a dense 50-dimensional vector.
    - This layer learns the word embeddings *during* training.
2. **LSTM Layer:** LSTM(64, dropout=0.2, recurrent_dropout=0.2)
    - This is the main processing layer. It receives the sequence of 50-dimensional word vectors.
    - It has 64 memory units, which are responsible for learning the patterns and context from the sequences.
    - dropout=0.2 and recurrent_dropout=0.2 are applied for regularization, randomly dropping connections during training to help prevent overfitting.
3. **Dense Layer:** Dense(vocab_size=8201)
    - A standard, fully-connected neural network layer.
    - It takes the 64-unit output from the LSTM layer and projects it onto a high-dimensional space matching the vocabulary size (8,201). The outputs of this layer are known as "logits."

4. **Activation Layer:** Activation('softmax')
    - This final layer applies the softmax function to the logits from the Dense layer.
    - This converts the logits into a probability distribution, where each value represents the model's predicted probability for a specific word, and all 8,201 probabilities sum to 1.0.

This model is compiled with the RMSprop optimizer and the categorical_crossentropy loss function, which is the standard loss function for multi-class classification problems with one-hot encoded labels.

CHAPTER 6

EXPECTED OUTCOMES

6.1 Expected Results

The model was successfully trained for 10 epochs on the prepared dataset. The training process and its results are as follows:

- Training and Validation: The model trained without errors. The training logs show the loss and accuracy for the training data, and the val_loss and val_accuracy for the held-out validation data.
- Final Accuracy: The model achieved a final training accuracy of approximately 10.87% and a final validation accuracy of approximately 9.28% after 10 epochs.
- Performance Analysis:
    - Although ~9–10% accuracy may appear low, it must be considered relative to the task. With 8,201 possible next-word choices (random ≈0.012%), achieving 9.28% on the validation set shows the model is learning meaningful language patterns rather than guessing.
    - The loss curves support this: training loss decreases steadily, and validation loss also declines at a slower but consistent rate. This indicates the model is learning effectively and generalizing to unseen data.
    - Training and validation accuracy both rise over time, which is the desired trend. The moderate gap between the curves suggests some overfitting, but it remains acceptable for this model and task complexity.
- Functional Prediction: The model is functionally capable of prediction. The provided test case (seed_text = "the adventures of sherlock") resulted in the predicted next word 'and', demonstrating the model's ability to generate a plausible (though not the most famous) next word.

CHAPTER 7

CONCLUSION

This project successfully demonstrates the complete workflow for building a next word prediction model using an LSTM network in Keras. A functional prototype was built, which preprocesses text, trains on sequential data, and generates predictions.

The final validation accuracy of 9.28% is a promising result for a model of this limited scope, proving that it has learned meaningful linguistic patterns far better than chance.

The project highlights the challenges of language modeling, namely the high-dimensional output space (a large vocabulary) and the need for significant data and computational resources. Future work to improve this model could involve:

1. **Using a Larger Corpus:** Training on a much larger and more diverse dataset would improve the model's vocabulary and understanding of language.
2. **Increasing Model Complexity:** Stacking multiple LSTM layers or increasing the number of units (e.g., from 64 to 128 or 256) could allow the model to learn more complex patterns.
3. **Longer Training:** The model's validation loss was still decreasing, suggesting that training for more epochs could yield further improvements.
4. **Hyperparameter Tuning:** Experimenting with the learning rate, batch size, embedding dimensions, and dropout rates could further optimize performance.

Overall, the project serves as an excellent practical introduction to the field of sequential NLP and deep learning.

REFERENCES

- Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780. (The original LSTM paper)

- Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781. (The paper that popularized word embeddings/Word2Vec)

- Chollet, F. (2015). Keras. https://keras.io (Keras library documentation)

- Bird, S., Klein, E., & Loper, E. (2009). Natural Language Processing with Python. O'Reilly Media. (NLTK book)