Shailendra Kumar_Final Thesis_LJMU_Oct2023.pdf

by Shailendra Kumar

Submission date: 22-Oct-2023 05:38PM (UTC+0100)
Submission ID: 213849586
File name: Shailendra_Kumar_Final_Thesis_LJMU_Oct2023.pdf (6.42M)
Word count: 29115
Character count: 172767

A Comparative study of Optimizers for training Deep Neural Networks in Text Classification Tasks.

Shailendra Kumar

Final Thesis Report

OCTOBER 2023

DEDICATION

I would like to take the opportunity here to dedicate this thesis work to my very caring family. My family created an atmosphere of love and trust throughout my journey through this tough thesis work. They always constantly motivated me to work hard and reminded me of my capabilities. Which finally led to this work of mine.

My parents Shri Chandra Bhushan Ram and Shrimati Brij Bala Devi, their constant thought provoking and enthusiastic words helped me maintain motivation throughout this journey. They believed in me also shared quite a lot of responsibility of running a home. This gave me additional time to dedicate to my work. I would always be indebted to them for their sacrifice.

My life partner Diksha Prajapati has been a constant source of love and affection. During hard times she motivated me to keep faith in my abilities. She took out a lot of my responsibilities which gave me time to work hard on my project. My Son Shreshth, who is very young but his smile encouraged me to excel in my work.

This complete thesis is possible only due to the love, care, affection support and motivation of my family. I am very glad to dedicate this thesis work to my family members.

ACKNOWLEDGEMENTS

I take an opportunity here to express my sincere gratitude towards all those who are involved in this successful journey of mine. First, my very sincere gratitude to my University Liverpool John Moores University and upGrad for making it possible to study this prestigious course from my city. The well-designed platform and course work helped me a lot in this intellectual journey. This journey not only developed my understanding of machine learning and artificial intelligence but also developed skills to tackle individual research work in my field of interest.

I feel deep gratitude towards my supervisor Shri Arindam Chaudhuri. He has always supported me despite his bad health and work commitments. His guidance paved a strong path to success in this intellectual research journey.

My family has been a very strong support pillar throughout this research journey. Their love, affection and motivation make all this possible. They constantly supported me without any demand. I am highly indebted to them for their selfless support.

I can't forget my mentors, friends and colleagues who also helped me with their understanding of the subject. This helped me create a broader perspective of the subject.

Lastly It's time to appreciate the work of all imminent scholars whose work has paved the way for my research. Their research work is a must for developing the understanding on subject matter and creating this work.

In short, I am thankful to each individual and institution for their support, guidance, and love. Which made it possible for me to complete this thesis work. Thanks a lot for being my guiding light through this journey.

TABLE OF CONTENTS

ABSTRACT

In today's data-driven world, the generation of textual data has witnessed an unprecedented surge, primarily due to the widespread use of handheld devices among the global population. As such, the urgent need for efficient systems to analyse and classify text data with high accuracy and ease has become increasingly paramount. This requirement stems from the diverse array of critical applications that rely on accurate text classification, including sentiment analysis, hoax detection, and spam detection, all of which significantly impact human life daily. During this study, it was revealed that the conventional SGD classifier inadequately addressed the complexities inherent in text classification tasks, serving as the catalyst for an in-depth exploration of optimizer functionalities, thereby leading to the development of this thesis work.

The research process involved extensive data preprocessing, including text normalization, stop words removal, and lemmatization, followed by the conversion of the dataset into embeddings to facilitate effective neural network training. Notably, three distinct neural network architectures were developed, encompassing a basic neural network, an RNN with LSTM, and an RNN with GRU units, Models were also trained of Basic NN Model after tuning hyperparameter of each optimizer. The performance evaluation of the optimizers was based on various training and validation metrics, such as training loss, validation loss, training accuracy and validation accuracy, alongside an in-depth analysis of the training graphs over successive epochs. The findings prominently showed the efficacy of the Adam, AdamW, and Nadam optimizing algorithms, exhibiting rapid learning, stable training dynamics, and minimal time requirements, positioning them as the optimal choices for training neural networks in the realm of text classification tasks.

The implications of this research extend to both academic and industrial domains, providing valuable insights for the efficient utilization of resources and laying the groundwork for future advanced research. Future work may include evaluating the optimizer on more advanced models like BERT & GPT. A meticulous work of optimizing hyperparameters of these optimizers can be taken to enhance the efficiency of neural network training in text classification tasks. Lastly, research work can also be taken over multiple dataset of different language, genre and classes to understand efficacy over broader range.

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

SGD…….. Stochastic Gradient Descent

TC…..... Text Classification

NLP……. Natural Language Processing

CHAPTER 1 : INTRODUCTION

1.1 Background

The wide-ranging applications of Natural Language Processing (NLP) in artificial intelligence have spurred significant research efforts aimed at enabling machines to comprehend and interpret human language. Notably, within the domain of text classification, numerous studies have contributed to the development of sophisticated algorithms and methodologies. However, the low performance of the widely employed Stochastic Gradient Descent (SGD) optimization algorithm in this context has underscored the need for a more comprehensive investigation into the intricacies of optimization techniques in NLP.

Motivated by the observed limitations of the SGD algorithm, a deep-seated curiosity to unravel the underlying factors governing optimization efficacy in text classification tasks prompted an in-depth exploration of alternative approaches. This exploration ultimately led to the decision to undertake a comprehensive evaluation of available optimizers, with a specific focus on their effectiveness in training deep neural networks for text classification purposes. By delving into the nuanced dynamics and intricacies of various optimization algorithms, the study aims to discern the unique strengths and weaknesses of each approach, thereby providing valuable insights into their applicability and performance in the context of text-based classification challenges.

With the overarching goal of contributing to the refinement of optimization methodologies for NLP applications, the research endeavours to fill the existing gap in understanding regarding the impact of different optimization strategies on the training efficiency and generalization capability of deep learning models. By addressing the critical need for more efficient and effective optimization frameworks tailored to the specific demands of text classification tasks, the study seeks to lay the foundation for the development of robust and adaptive optimization methodologies that can be readily applied to diverse NLP applications, thereby advancing the field and facilitating the development of more sophisticated and reliable NLP systems.

1.2 Problem Statement

Within the dynamic landscape of Natural Language Processing (NLP), the continuous evolution of deep learning methodologies has brought forth a pressing challenge regarding the comprehensive understanding of the efficacy of available optimization algorithms and their implications for training deep neural networks in text classification tasks. While significant strides have been made in the field, the persistent performance discrepancies of the widely adopted Stochastic Gradient Descent (SGD) algorithm have brought into sharp focus the need for an extensive exploration of alternative optimization techniques tailored to the nuances of NLP applications.

This existing gap in knowledge poses a significant obstacle, hindering the development of robust and adaptive optimization methodologies necessary for the efficient handling of text-based classification challenges. The lack of comprehensive insights into the comparative performance and behaviour of diverse optimization algorithms in the context of training deep neural networks for text classification tasks remains a critical concern. Understanding the intricate interplay between optimization strategies and the complex dynamics of textual data presents a pivotal challenge that demands urgent attention and investigation.
Therefore, the central problem addressed by this research is the critical need to bridge the existing gap in understanding the efficacy of various optimization algorithms and their specific implications for training deep neural networks in text classification tasks.

By conducting an in-depth analysis and comprehensive evaluation of the behaviours and performance characteristics of different optimizers, the study aims to contribute vital insights necessary for the development of more effective and efficient optimization frameworks precisely tailored to the unique demands of NLP applications. This exploration seeks to not only address the current challenges in text classification but also pave the way for the advancement of the broader field of Natural Language Processing through the development of more robust and adaptable deep learning methodologies.

1.3 Aims and objectives

The primary aim of this study is to comprehensively evaluate the efficacy of various optimization algorithms available in Keras Library in the training of deep neural networks for text classification tasks. By systematically examining the behaviors and performance characteristics of these optimizers, the research aims to identify the most suitable optimization strategies capable of addressing the complexities and challenges inherent in text-based classification problems. Furthermore, the study aims to contribute to the advancement of optimization methodologies in the domain of Natural Language Processing, facilitating the development of more efficient and effective deep learning frameworks for text classification tasks.

Objectives:

- To design and implement a comprehensive experimental framework for evaluating the performance of optimizers in training various deep learning models for text classification.
- To assess and compare the convergence dynamics, generalization performance, and computational efficiency of the identified optimizers through rigorous experimentation and analysis.
- To provide actionable insights and recommendations for practitioners and researchers regarding the selection and application of optimization algorithms for text classification tasks, thereby contributing to the broader understanding of optimization methodologies in the field of Natural Language Processing.

1.4 Research Questions

What is the comparative efficacy of different optimization algorithms in the training of deep neural networks for text classification tasks, and how do these optimizers impact the convergence dynamics, generalization performance, and computational efficiency of the models?

1.5 Scope of Study

1.5.1 In-Scope:

1. Comprehensive evaluation optimization algorithms in the context of training deep neural networks for text classification tasks.
2. Analysis of the convergence dynamics, generalization performance.
3. Assessment of the behavior of the optimizers with different types of deep learning architectures specifically tailored for text classification.
4. Providing actionable insights and recommendations for practitioners and researchers regarding the selection and application of optimization algorithms in the context of Natural Language Processing (NLP) tasks.

1.5.2 Out-Scope:

1. Detailed analysis of optimization algorithms in domains other than text classification, such as image recognition.
2. Investigation into the intricacies of hardware-specific optimization for deep learning models.
3. Implementation of complex ensemble methods or hybrid approaches involving optimization algorithms.
4. In-depth exploration of the theoretical foundations and mathematical properties of optimization algorithms beyond their application in training deep neural networks for text classification tasks.

1.6 Significance of the study

The significance of this study lies in its potential to revolutionize the landscape of Natural Language Processing (NLP) by offering comprehensive insights into the efficacy of different available optimization algorithms in the training of deep neural networks for text classification tasks. Given the burgeoning volume of textual data generated daily, the need for robust and efficient optimization methodologies has become increasingly paramount. By meticulously evaluating the performance and convergence dynamics of these optimizers, this study aims to provide valuable guidelines and recommendations for researchers and practitioners in the field, facilitating the development of more effective and accurate text classification models. Furthermore, the findings of this research are anticipated to contribute significantly to the

refinement of optimization strategies in NLP, fostering the development of advanced deep learning frameworks capable of addressing the intricate challenges posed by real-world text-based applications.

Moreover, the implications of this study extend beyond the realm of NLP, offering valuable insights into the broader domain of deep learning optimization. By elucidating the distinctive characteristics and performance metrics of available optimizers, the research seeks to establish a framework for the systematic evaluation and comparison of optimization algorithms in diverse machine learning applications. The findings of this study have the potential to inform future research endeavours and inspire the development of novel optimization techniques tailored to specific application domains, thereby fostering advancements in the wider field of artificial intelligence.

1.7 Structure of Study

The study consists of several structured sections contributing to a comprehensive analysis of optimization algorithms in training deep neural networks for text classification tasks. The abstract serves as a succinct overview of the research, while the list of tables and figures provides a comprehensive overview of the visual aids integrated into the study.

In the introduction, the focal point revolves around establishing the background of the study, emphasizing the significance of evaluating optimizers in text classification. Moreover, the problem statement is articulated with precision, accompanied by a clear outline of the study's aims, objectives, research questions, and scope.

The subsequent sections include a thorough exploration of the relevant literature, covering the evolution of optimization algorithms, their limitations, and previous studies on optimizers in the context of natural language processing (NLP). The methodology section entails a comprehensive overview of the research design, encompassing data collection, preprocessing techniques, and the implementation of various deep learning model architectures. This section provides detailed insights into the systematic approach adopted to assess the performance of available optimizers in the context of training deep neural networks for text classification.

The implementation details section delves into the intricate data cleansing, feature selection, and model building, accompanied by detailed study. The results and discussion segment entails an exhaustive analysis of the research findings, focusing on the evaluation of different optimizers in the context of training deep neural networks for text classification. This

comprehensive examination includes an in-depth comparison of the efficacy of available optimizers, shedding light on their impacts on convergence dynamics and generalization performance. The implications of these findings are discussed in relation to the broader landscape of natural language processing, highlighting the practical implications.

The study concludes by emphasizing the significance of the research outcomes and offering valuable recommendations for future studies and practical applications within the field of natural language processing and deep learning optimization. This final segment underscores the critical contributions of the study to the advancement of optimization methodologies, elucidating its potential impact on the development of more robust and efficient deep learning frameworks designed to effectively address the evolving demands of text-based classification tasks.

CHAPTER 2 : LITERATURE REVIEW

The Literature Review chapter provides a comprehensive exploration of the dynamic landscape of the natural language processing (NLP) domain, emphasizing the fundamental concepts, methodologies, and applications that have shaped the evolution of text analysis and classification techniques. This chapter serves as an essential overview of the diverse dimensions of NLP, highlighting the pivotal role of language processing algorithms in deciphering and extracting insights from textual data. By delving into a rich array of scholarly works and seminal studies, this review aims to offer a comprehensive understanding of the foundational principles and contemporary advancements in the NLP landscape, emphasizing the multifaceted challenges and intricacies inherent in the analysis and comprehension of human language by machines, alongside the historical progression and key contributions that have paved the way for the development of text classification techniques.

2.1 Introduction

Natural language Processing has emerged as one of the most important and challenging sub-fields of Artificial Intelligence. NLP (Natural Language Processing) revolves around the challenge of interpreting natural language using computers and making human computer interaction easy. NLP involves multiple sub-tasks like text classification, sentiment analysis, machine translation, topic modelling, text summarization, Question-answering systems like Chabot, text generation etc. Each of these tasks is very important as it handles some specific set of challenges in making machine understand natural language.

The field has witnessed a very rapid growth in the past decades. The growth is fueled by an increase in availability of data and generation of humungous data every second over internet. Social Media Platform like Facebook, Twitter, Blogs, e-learning platforms, and similar sites are adding variety of humungous data with every second. (Hassan et al., 2020; Pimpalkar and Raj, 2020; Shah et al., 2020; Goswami and Sabata, 2021; Al-Anzi, 2022b). Computational efficiency of machines has also increased manifold due to the development of new technologies like Graphical Processing Unit (GPU), Tensor Processing Unit (TPU). All these factors have boosted the growth and demand of NLP techniques.

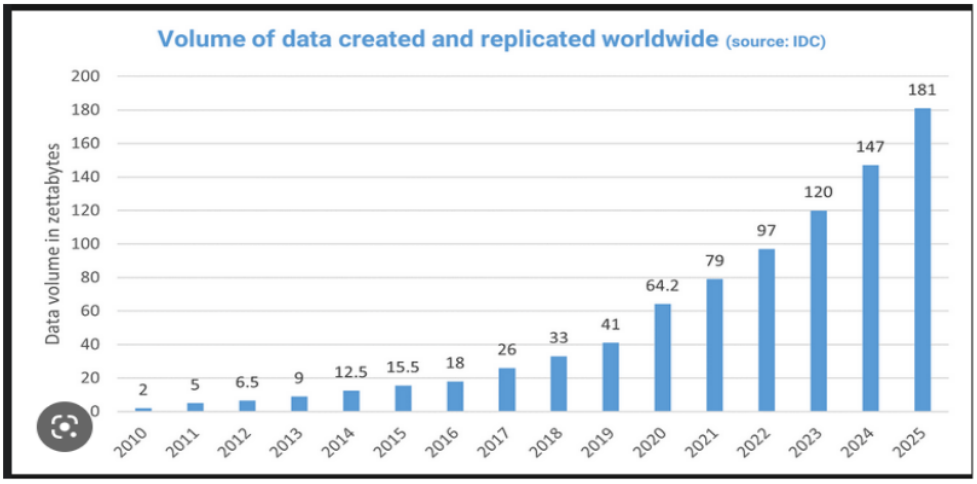


Figure 2.1 Figure for growth in data generation

Text classification is one such very important task. It involves classification of given text into predefined categories. Sentiment Analysis is also a similar task, which categorizes given piece of text in positive or negative category. Spam mail detection that is classification of an email into spam or ham is also a similar classification task. Text classification is a challenging task because it requires the algorithm to understand the meaning and semantics of the textual data.

2.2 Importance and application of the field

Text classification has developed as a very important field because it enables automatic analysis of readily available huge textual data. Which can be very tedious and error-prone for humans. Information Retrieval enables individuals to take informed decisions based on textual data. Sentiment Analysis helps understand how consumers think about their products and interact with them. This helps businesses improve their product to match consumer expectations. Spam Detection helps IT teams to fight cyber intrusion activities by directly filtering harmful and unwanted email from inbox. Text Classification has actively been researched to develop and refine efficiency and performance.

Numerous Tasks in Text Classification are:

- Sentiment Analysis
- Span Filtering
- Topic Classification
- Fraud detection
- New classification

2.3 General understanding about the whole process overview of TC

Text Classification involves a series of processes to accomplish the desired task. Text corpus is Pre-Processed to make data suitable for feature extraction. Main Pre-Processing steps are tokenization, stop word removal, stemming and lemmatization. After pre-processing, the text corpus is now required to be converted in a format that can be fed to machine learning algorithms. Feature extraction is using different techniques like Bag of Word (BOW), TF-IDF, n-gram etc.(Pimpalkar and Raj, 2020; Shah et al., 2020; Al-Anzi, 2022a) Dataset is split into train set, test set, and fed to the ML algorithms like Logistic regression, Support Vector Machines, etc.). After training is completed, evaluation is done based on suitable evaluation metrics like Precision, Recall, Accuracy and F1-score.

2.4 Comparison and Summary Tables:

Comparison and summary tables created to consolidate the important information in study. Table created for Pre-Processing techniques, Algorithms used for classification and evaluation matrix. Table also created for source of dataset.

Table 2.1 Summary Pre-Processing Techniques, Algorithms & Evaluation Metrics

Year	Reference	Pre-processing techniques used	Algorithms	Evaluation Metrics
2021	(Goswami and Sabata, 2021)	Term Frequency and Word Frequency used for text document Representation.	Logistic regression and SGD	Accuracy Precision Recall F1-Score
2022	(Al-Anzi, 2022b)	n-gram, TF-IDF, Partially Ordered Microword	SGD Classifier	Accuracy Precision Recall F1-Score
2020	(Shah et al., 2020)	n-gram, stemming, lemmatization, TF-IDF	Logistic regression, Random Forest, KNN	Accuracy Precision F1-Score Support
2020	(Pimpalkar and Raj, 2020)	BOW, TF-IDF	Multinomial Naïve Bayes, LR, Decision Tree, SVM, XGBoost	Accuracy Precision Recall F1-Score
2019	(Hassan et al., 2020)	n-grams, TF-IDF	Linear SVM, LR, RF, NB, KNN	Accuracy Precision Recall F1-Score
2021	(Patel and Meehan, 2021)	TF-IDF, count vectorizer	LR, MNB, SVM	Accuracy Precision Recall F1-Score

2019	(Kowsari et al., 2019)	all	all	all
2021	(Gowri et al., 2021)	count vectorizer, TF-IDF	Decision Tree, NB, RF, LR, SVM, Passive Aggressive Classifier, SGD	Accuracy Precision Recall F1-Score
2018	(Prasetijo et al., 2017)	tokenization, stop words, stemming, lowercasing, n-gram, bow, tf-idf	SGD Linear Regression, SVM Linear Kernel, SGD Modified Hurbe	Accuracy Precision Recall F1-Score
2018	(Shahreen et al., 2018)	count vectorizer, TF-IDF	SGD with lgbfs, sgd, Adam, log loss	precision, Recall, f1-score
2022	(Roshinta et al., 2022)	tf-idf, stemming, cleaning, stop word removal, tokenize	LR, SVM, SGD, NB	Accuracy Precision Recall F1-Score
2019	(Madhfar and Al-Hagery, 2019)	BOW	DT, NB, SVM, RF, SGD, LR	Precision, Recall, f1-score
2022	(Varshney et al., 2020)	TF-IDF	RF, SVM, LR, NB, SGD Classifier	accuracy, precision, recall, f1-score
2022	(Al-Anzi, 2022a)	Partially ordered micro-word representation, TF-IDF used for this microword-vector representation	SGD Classifier	Precision, Recall, Accuracy, F1-Score

Table 2.2 Summary of Different Dataset

Year of Publication	Reference	Dataset
2021	(Goswami and Sabata, 2021)	Amazon instant video datasets, bank dataset, Movie review.
2018	(Diab, 2019)	Global Terrorism Database (GTD) built by the Study of Terrorism and Responses to Terrorism (START) consortium.
2022	(Al-Anzi, 2022b)	created own dataset due to scarcity and limited availability of arabic corpora
2020	(Shah et al., 2020)	BBC News
2020	(Pimpalkar and Raj, 2020)	Twitter reviews of common, Internet film Database. Dataset downloaded from Kaggle; twitter was crawled.
2019	(Hassan et al., 2020)	PHEME Dataset, highly commented tweets regarding, Charlie Hebdo, Sydney Siege, Ottawa Shooting, Germanwings-Crash, and Ferguson Shooting “ CAT - Arabic Dataset
2021	(Patel and Meehan, 2021)	72,000 posts from Reddit.com
2021	(Gowri et al., 2021)	PHEME, CREDBANK
2018	(Prasetijo et al., 2017)	collected news article and manually labelled as hoax and non-hoax
2018	(Shahreen et al., 2018)	keywords identified with experts then using twitter streaming API real-time tweets were extracted.
2022	(Roshinta et al., 2022)	200 Indonesian dataset, 100 fake & 100 real

2019	(Madhfar and Al-Hagery, 2019)	111728 Documents that fall into 5 categories. 319,254,124 words.
2022	(Varshney et al., 2020)	16,00,000 tweets 8 lakh positive and 8lakh negative labelled.
2022	(Al-Anzi, 2022a)	developed one Arabic datasets and lexicons due to the scarce and limited availability of corpus needed for Arabic sentiment classification and manually annotated. ARABIC SENTIMENT ANALYSIS TWITTER DATA CORPUS

Evaluation metrics:

Comparison and study of research papers revealed that nearly all papers used standard evaluation metrics for classification i.e., Accuracy, Precision, Recall & F1-Score.

Resource requirement:

All papers compared and seemed to use normal hardware for the training. As the dataset is small and the training does not require very large resources for computing. (Cahyani and Nuzry, 2019; Varshney et al., 2020; Al-Anzi, 2022a; Daud et al., 2023)

2.5 Text Preprocessing Techniques

The textual data consists of unstructured text like contents from books, tweets, blogs, email etc. The raw text has a lot of unwanted text that doesn't add any significant information. These items are preposition, article, punctuation etc. These items do not add any valuable information instead, they can cause adverse effects on Algorithm performance. These items are removed from the textual data by doing different pre-processing operations like cleaning, stemming, lemmatization. (Kowsari et al., 2019; Gowri et al., 2021) Each task has its own significance.

2.5.1 Tokenization

Tokenization is the process of creating tokens from the continuous stream of text. Text is generally available in sentences in continuous form. Hence, it needed to be broken down into basic units of text data i.e., words. It can also have other elements, for example numbers, punctuation marks, or other items. Text is split into individual words based on spaces or

punctuation. Tokenization prepares the basis for further NLP operations and analysis.(Kowsari et al., 2019)

2.5.2 Stop Word removal.

Textual data is structured and meant for human understanding, so it contains a good number of articles, prepositions etc. This does not really add much value to Classification algorithms. Since this does not add much value, it is the most common technique to remove stop words from text data. Example: {“an”, “above”, “on”, “in”, “before”, “a”, “the”}.

2.5.3 Case Normalization or Capitalization

Text documents have inherently use capital letters and this can lead to issues in classifying the documents. This happens because the capitalized word is treated as a different word from the non-capitalized word. For Example, the following words are considered different from each other. {“new”, “New”, “NEW”}. In text documents Proper Nouns, beginning of sentences, Titles and headings, Acronyms, Pronoun first letter is capitalized. Hence, all tokens are lowercased to remove inconsistency and duplicity of words. Sometimes, it may cause interpretation problem for Example when “US” is lower cased. It becomes “us” pronoun. Some techniques have been developed to handle this issue for Example: Slang and Abbreviation.

2.5.4 Stemming

Stemming is a technique used in pre-processing of text. The main objective of this step is to reduce the different words to their root form. This reduces the no. of unique words available in a given corpus. This reduces the memory, compute resource and other related complexities due training of the algorithm. Some of the techniques are rule based and use statistical model for the same. In English language, Porter Stemmer is most widely used.

Stemming face issues when same stem has two different meaning. In this case, it cause ambiguity in meaning. Stemming removes prefix and suffix only to reach at the stem of word. Hence, It is always not correct and sometimes give wrong or incomplete results.

2.5.5 Lemmatization

To overcome deficiencies in Stemming processing lemmatization is used. Lemmatization uses context and meaning of word to decide the base form of word. It can also handle irregular words.

Overall Stemming is fast but less accurate. On the contrary, lemmatization is accurate but more time and resource consuming. Suitable method must be selected based on the requirement and considering this trade off. (Plisson et al., n.d.; Korenius et al., 2004; Kowsari et al., 2019)

2.6 Syntactic Word Representation

Pre-processing of the text data makes it suitable for further processing. Feature extraction is the next step. The pre-processed text still cannot be directly feed to machine learning algorithms because machine learning algorithms need numerical data for training. Hence, the next step is to convert text data in to numerical data using feature extraction techniques.

Following are well-established techniques for the feature extraction.

2.6.1 N-gram

n-gram is a sequence of n contiguous words from a given sentence, text or document. For example: in the sentence “Sun rises in the east”, some example are “Sun rises”, “rises in”, “in the”, “the east” etc. N-grams is a versatile technique and useful in different type of NLP tasks such as machine translation, speech recognition, text classification, information retrieval. Bag of Words is an example of 1-gram. Here in BOW the words lose their syntactic order. In most of NLP tasks 2-gram and 3-gram is used because it can extract more information compared to 1-gram.

For example:
Sample sentence:
“Sun rises in the east”
2-grams – {“Sun rises”, “rises in”, “in the”, “the east”}
3-grams – {“Sun rises in”, “rises in the”, “in the east”}

2.7 Weighted words

Weighted word is a technique of feature extraction. Term Frequency is the most basic form of weighted word feature extraction. In TF technique, each word is assigned a number equal to its occurrence in the text document. One main disadvantage of this technique is that the words, which occur more frequently in the given language, can dominate and create a biased view.

2.7.1 Bag of Words

BOW is a very simple and effective technique of feature extraction from text data. It treats a document or sentence as a bag of words. It divides text corpus into a simple list of words. Only content is important. Grammar, sentence structure and semantic relationship between words of given text is ignored. Multiple occurrences of words are sometime taken into consideration.(Zhang et al., n.d.; Kowsari et al., 2019; Peng et al., 2022)

Example - “LJMU is an celebrating its bicentenary.”
Bag-of-Words (BoW) - {“LJMU”, “is”, “an”, “celebrating”, “its”, “bicentenary”}

BoW has some limitations because one-hot-encoding is used for creating a matrix. Where 1 represents the presence of a word and 0 is at every other position. As the vocabulary increases, the size of matrix also increases, and it becomes a challenge of scalability of systems. Also, the order is not important here Hence, two different sentences with same vocabulary have same vector representation.

TF (term frequency) feature extraction is a technique where number of occurrences of a word is counted and then it is normalized by dividing the term frequency by total number of terms in the text corpus. And then the normalized term frequencies are arranged in a vector that represent the document. Each element of vector represents the normalized frequency of a term. This method also has limitation that it does not capture the semantics and context.

Count Vectorizer is one the most common techniques used for feature extraction. It converts text corpus into a sparse matrix of tokens. Here each row of matrix is represented as a document and each column is represented as term in the documents. Cell represents the number of times the term (column) has occurred in the document (row).

This method limitation in terms of limited semantics, High dimensionality. Interpretability is also challenging due to high dimensional feature space. Pre-processing steps can have a huge impact on the results of count vectorizer because it is very sensitive to Pre-processing steps.

2.7.2 TF-IDF

TF-IDF is a versatile and robust technique used for feature extraction. It is known as Term Frequency – Inverse Document frequency. It combines two things as the term suggest Term Frequency and Inverse document frequency. Term Frequency is a measure of how frequently a term occurs in a document and calculated as described earlier. IDF measures is a measure of the rarity of a term in a document. It is calculated as logarithm of the ratio of the total no. of documents in the collection to the number of documents that contain the term. (Salton and Buckley, 1988; Kowsari et al., 2019; Peng et al., 2022)

TF-IDF is finally calculated as product of TF and IDF. The final value shows the importance of the term in the document or collection of documents. It is a powerful technique because of its ability to capture importance of the term in a document and its rarity. It solves some of challenges and limitations of Count Vectorizer like semantics and sensitivity to preprocessing techniques.

$$W(d,t) = TF(d,t) * log(\frac{N}{df(t)})$$

2.8 Word embeddings

In spite of having syntactic representation of words. It is still a challenge to capture semantic meaning of words. To overcome this issue word embeddings are used which is representation of words in a high dimensional vector space. The vector space is designed in such a way so that it can capture semantics & contextual meaning between words. This enabled machine learning models to efficiency in understanding the meaning of a given word, its relationship with other words in a given context.(Kowsari et al., 2019; Peng et al., 2022)

Word embeddings capture semantics and relationships. Words having similar meanings, close relationship or that appear in similar contexts are represented by vector which are close to each other. Here similar words are having similar representations in form of similar vector. Some of the most popular word embeddings are Word2Vec, GloVe (Mikolov et al., n.d.; Pennington et al., n.d.; Goldberg et al., 2014; Kowsari et al., 2019) and FastText.

These are already trained on very large corpus of data to create word embeddings. These can further be used in different Natural Language Processing tasks like sentiment analysis, text classification, translation etc.

Word embeddings have proven their efficacy in all NLP tasks due to their ability to capture semantics and internal relationships in different contexts. This has resulted in better performance of different NLP trained models.

2.8.1 Word2vec

(Mikolov et al., n.d.; Goldberg et al., 2014; Kowsari et al., 2019) Word2Vec is one of the most used techniques in NLP for creating word embeddings. As discussed earlier, word embeddings are numerical representations of words. The focus is representation of words in vector space so that it captures semantics and relationships. Hence, the words which are similar will lie close to each other in this vector space.

Word embeddings are created here by training a neural network on corpus of data. Training enables neural networks to learn the word embeddings. Word2Vec models can be trained using two different architectures or methods namely CBOW & Skip-gram.

2.8.1.1 CBOW

In this architecture, model is trained to predict a word based on its surrounding words called context. Here the input is context (set of words) and output is the target word. Here the aim is

to predict correctly the target word given the set of words called context. (Mikolov et al., n.d.; Kowsari et al., 2019; Peng et al., 2022)

2.8.1.2 Skip-gram

In this architecture, the process is reversed. Given a target word, model tries to predict the context words. Input is word and output are set of context words. Here model is trained with aim to predict context words when the target word is given.(Mikolov et al., n.d.; Kowsari et al., 2019; Peng et al., 2022)

2.9 Algorithms

In the field of text classification, the algorithm takes a most significant role, taking on the primary task of automatically assigning pre-defined categories or labels to text documents. Its function within the context of text classification can be described as follows:

1. **Feature Extraction**: In this step, the algorithm pulls out important details from the text data. These details can be single words, groups of words, or even more complex representations like word embeddings. This helps the algorithm grasp the content and structure of the text documents.
2. **Training & Model building:** During the training phase, the algorithm is given a dataset with labels, where each text document is linked to a known category or label. The algorithm studies this dataset to understand how the extracted features are related to the assigned categories. During training process, the algorithm develops a model capable of making predictions using the patterns it has learned. This model is developed in a way so that it can generalize on unseen data also. It can easily classify unseen text documents into the appropriate categories.
3. **Testing and Evaluation:** After training phase, the algorithm is tested on a separate test dataset to assess the performance of model. It makes predictions for the text documents in this set, and the accuracy of these predictions is compared to the known labels. Most common evaluation metrics are accuracy, precision, recall, F1-score, and others, depending on the specific classification problem.
4. **Inference:** Once the algorithm is trained and properly evaluated, it can be used for text classification on new, unlabelled documents. It extracts the features from given document and uses the model to predict the category.
5. **Fine-Tuning & Deployment:** Sometimes there may be need to further optimize or fine tune model using techniques like hyperparameter tuning, retraining or updating model on newly acquired data or using techniques like cross-validation. Once the developed model gives satisfactory performance. It is ready for deployment in real-world scenarios to automatically do text classification job. Example like sentiment analysis, spam detection, topic categorization and such other tasks.

2.10 Classification of Algorithms

Further, Text classification algorithms can be classified based on various factors, like underlying techniques, complexity, and its applications. Following shown are some of the main classification approaches for text classification: (Kowsari et al., 2019; Peng et al., 2022)

1. **Machine Learning Algorithms:**
    - **Statistical Models:** Algorithms like Naive Bayes, Logistic Regression, and Support Vector Machines (SVMs) are used extensively for text classification with good performance.
    - **Ensemble Methods:** Algorithms include Random Forests and Gradient Boosting (e.g., XGBoost) can also be used for text classification.
    - **Rule-Based Models:** Decision Trees and Rules-Based systems can also be employed.
    - **Distance-Based Algorithms:** k-Nearest Neighbors (k-NN) can be used for text classification tasks.
2. **Deep Learning Architecture:**
    - **Feedforward Neural Networks:** Simple neural networks with fully connected layers can be used for text classification.
    - **Convolutional Neural Networks (CNNs):** CNNs can also be used for text classification where the spatial arrangement of words matters, such as text classification with fixed-length input (e.g., text sentiment analysis).
    - **Recurrent Neural Networks (RNNs):** RNNs, including LSTM and GRU variants, are suitable for sequential data like text classification with variable-length input (e.g., document classification). RNN are better that other deep learning models because they develop contextual understanding of sequence of text. (Sutskever et al., 2011; Kowsari et al., 2019; Peng et al., 2022)
    - **Transformers:** Models like BERT, GPT, and RoBERTa have revolutionized text classification and are known for their ability to capture context and relationships between words effectively.
3. **Based on Data availability for training:**
    - **Small Data:** Some algorithms are more suitable for text classification tasks with limited labeled data, such as Naive Bayes or logistic regression.
    - **Large Data:** Deep learning models, especially transformers, tend to perform well when you have access to extensive labeled data.

4. **Binary vs. Multi-Class Classification:**
    - Some algorithms are designed primarily for binary classification (two classes), while others can handle multi-class classification (more than two classes).
5. **Rule-Based vs. Statistical Approaches:**
    - Rule-based approaches rely on predefined rules or patterns, while statistical methods learn from data through mathematical modeling.
6. **Text Data Representation:**
    - Algorithms may differ in how they handle text data representation, including bag-of-words (BoW), term frequency-inverse document frequency (TF-IDF), word embeddings, or contextual embeddings.
7. **Interpretability:**
    - Algorithms, like logistic regression and decision trees, are more and easily interpretable, while deep learning models like CNN, RNN, transformers are often considered black-box models because of inability to interpret these models. This also pose challenge to justify classification based decision in high stake situations.

The selection of a text classification algorithm in based on various criteria, like data accessibility, computational capabilities, and the desired balance required between model interpretability and performance. It's a standard practice to conduct experiments with different algorithms and methodologies to ascertain the most suitable approach for the given problem.

2.11 Overview of Learning techniques used in Text Classification:

Shallow learning and deep learning techniques in text classification are two main approaches for building TC models to perform the task of text classification. These approaches differ fundamentally in terms of the complexity of process involved and the extent to which they automatically extract features from the textual data. Here's an overview of both techniques in brief:

2.11.1 Shallow Learning Techniques:

- **Shallow learning**, also known as conventional or traditional machine learning, involves using comparatively simple models that do not have deep neural network architectures and mostly based on statistical modelling techniques. These models use features that are created manually and therefore have limited capacity to understand and catch complex relationships in textual data.
- **Feature Engineering:** These techniques demands significant feature engineering efforts. Features can be word counts, term frequency-inverse document frequency (TF-IDF) values, n-grams, and various other manually designed features.
- **Algorithms:** Common shallow learning algorithms for text classification are:
  - **Naive Bayes:** Assumes independence between features and is particularly suitable for text data.
  - **Logistic Regression:** A linear model used for binary and multi-class classification.
  - **Support Vector Machines (SVM):** Effective for separating data points in high-dimensional spaces.
- **Interpretability:** These Shallow learning models are more interpretable than deep learning models, as they rely on explicit features and weight coefficients.
- **Resource Efficiency:** These models require less computational resources and less data in comparison to deep learning models.
- **Use Cases:** Shallow learning techniques can work well for simpler text classification tasks with limited data and straightforward feature representations. Also, for cases where interpretability is very important.

Most used Algorithms in Shallow learning techniques are as follows:

1. **Naive Bayes:** Naive Bayes classifiers are simple and effective for text classification tasks. They are based on Bayes' theorem and assume that the features (words) in a

document are conditionally independent given the class label. Multinomial Naive Bayes and Bernoulli Naive Bayes are some variants of Naive Bayes Algorithm.

2. **Logistic Regression:** Logistic regression is a very efficient linear model that can be used for binary and multi-class text classification tasks. It basically models the probability that a document belongs to a particular class or not. It uses a logistic function for generating these probabilities with further decides classification.
3. **Support Vector Machines (SVM):** SVMs are powerful classifiers that can work well for text classification. The basic target of this algorithm is to find a hyperplane that best separates the different classes in a high-dimensional space. The kernel trick can be used to map data into a higher-dimensional space for better separation.
4. **Decision Trees:** These classifiers partition the feature space based on the features' values, ultimately leading to a decision about the class label. Decision trees can be interpreted easily and are suitable for certain text classification tasks.
5. **Random Forest:** This is a ensemble learning method that combines multiple decision trees to make predictions. This Algorithm is known for its robustness and ability to handle noisy data.
6. **k-Nearest Neighbors (k-NN):** This is a simple instance-based learning algorithm that assigns a class label to a text document based on the majority class labels among its k nearest neighbors in the given feature space.

2.11.2 Ensemble Techniques:

Ensemble learning methods can be efficiently used in text classification assignments to enhance model performance through the combination of prediction made by several base classifiers. Below are several ensemble learning strategies frequently utilized in text classification:

1. **Bagging :**
    - Random Forest is one the most popular ensemble technique in the field of text classification. It uses multiple decision trees, with each tree being trained on a distinct portion of the training data using randomized feature selection method. This approach mitigates overfitting and enhances overall generalization capabilities of the model.
2. **Boosting:**
    - **AdaBoost:** This is a machine learning ensemble method mainly used for classification tasks. AdaBoost works by combining the predictions of multiple

weak learners, often decision trees or stumps (shallow trees with only one split), to create a strong classifier.

- **Gradient Boosting:** Gradient boosting algorithms like XGBoost, LightGBM, and CatBoost are effective for text classification. It builds a strong predictive model by combining the outputs of multiple weak learners, typically decision trees. Gradient Boosting aims to correct the errors of previous models in an additive manner, iteratively improving the overall prediction performance.

2.11.3 Deep Learning Techniques:

- **Deep learning** is a subfield of machine learning that focuses on training artificial neural networks to perform complex tasks. Which typically require human level of intelligence. It involves the use of neural networks with multiple hidden layers to automatically learn complex representations of textual data. Deep learning models, known as artificial neural networks, are basically inspired by the structure and function of the human brain.
- **Feature Learning:** These models learn features automatically from textual data. They do not rely on handcrafted features. This makes them suitable to different types of text classification.
- **Architectures:** DL models for text classification include:
  - **Convolutional Neural Networks (CNNs):** Effective in capturing local patterns in text, such as n-grams.
  - **Recurrent Neural Networks (RNNs):** Suitable for sequential data and capturing contextual information in textual data.
  - **Transformers:** State-of-the-art models like BERT, GPT, and their variants have set new standards in text classification by effectively capturing global and contextual information.
- **Interpretability:** This becomes a challenge in Deep learning models, especially large transformer models are often considered less interpretable due to their complex architectures and high-dimensional embeddings.
- **Resources:** DL models, especially large transformers, require very high computational resources (e.g., GPUs or TPUs) and large amounts of data for training.

- **Use Cases:** DL techniques excel in complex text classification tasks where large datasets and high-quality representations are available. They have achieved state-of-the-art performance in various NLP tasks, that includes sentiment analysis, document classification etc.

In summary, the choice between shallow and deep learning techniques for text classification depends on factors such as the complexity of the task, the availability of data, computational resources, and the need for interpretability. Shallow learning is suitable for simpler tasks with limited data, while deep learning is more suited for highly complex tasks with significant and easily available data and where automatic feature learning is more advantageous for model training.

Following is overview of each DL technique:

Deep learning techniques have become increasingly popular for text classification tasks due to their ability to automatically learn complex representations from textual data. Here are some of the key algorithms and architectures used in deep learning for text classification:

2.11.3.1 Convolutional Neural Networks (CNNs):

- **Text-CNN:** CNNs are well-known for their effectiveness in image processing, but they can also be applied to text classification. Text-CNNs use convolutional filters to capture local patterns in text, such as n-grams, and then apply max-pooling to down sample the output. This architecture is especially useful for tasks where the spatial arrangement of words matters, such as text sentiment analysis. (Kim, n.d.; Peng et al., 2022)

2.11.3.2 Recurrent Neural Networks (RNNs):

- **Long Short-Term Memory (LSTM):** LSTMs are a type of RNN designed to capture sequential dependencies in data. They are widely used for text classification tasks that involve variable-length input sequences, like document classification.(Hochreiter and Schmidhuber, 1997; Peng et al., 2022)
- **Gated Recurrent Unit (GRU):** Similar to LSTMs, GRUs are a variant of RNNs that can be used for sequential text classification tasks. They are computationally efficient and often perform well.(Chung et al., 2014; Kowsari et al., 2019)

2.11.3.3 Transformers:

- **BERT (Bidirectional Encoder Representations from Transformers):** BERT is a transformer-based model pre-trained on a large corpus of text data. It has

revolutionized text classification by capturing contextual information bidirectionally, making it highly effective for various NLP tasks.

2.11.4 Deep Learning Frameworks:

- Deep learning frameworks like TensorFlow, PyTorch, and Hugging Face Transformers provide tools and pre-trained models that facilitate the implementation of deep learning algorithms for text classification.

The choice of algorithm depends on the specific text classification task at hand, the availability of data, and the required level of model complexity. Pretrained models, such as BERT, have become popular for many text classification tasks due to their strong performance and transfer learning capabilities. However, it's very important to fine-tune these models to suit specific use cases.

2.12 Model Evaluation Parameters

Evaluating the performance of text classification models is crucial for assessing their effectiveness in various natural language processing (NLP) tasks. Several evaluation metrics are commonly used to measure the performance of classification models. The choice of metric depends on the specific goals and characteristics of the task. Here are some of the most common evaluation metrics for text classification:

2.12.1 Accuracy:

Accuracy is the basic evaluation metric and measures the proportion of correctly classified data points among all data points in the dataset. It's suitable for balanced datasets but can be misleading when classes are imbalanced.

2.12.2 Precision:

Precision is the ratio of true positives (correctly predicted positive data points) to the total number of predicted positive data points. It measures the accuracy of positive predictions. High precision indicates low false positives.

Precision = TP / (TP + FP)

2.12.3 Recall (Sensitivity or True Positive Rate)

Recall measures the ability of the model to correctly identify all positive instances in the dataset. It is the ratio of true positives to the total number of actual positive instances.

Recall = TP / (TP + FN)

2.12.4 F1-Score

The F1-score is harmonic mean of Precision and Recall. It provides a balanced measure of a model's performance that considers both false positives and false negatives. It is especially useful when class distribution **is imbalanced.**

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

2.12.5 Classification Report

A classification report provides a comprehensive summary of multiple metrics, including precision, recall, F1-score, and support (the number of occurrences of each class). It is often generated for each class in multi-class classification tasks.

The choice of evaluation metric is based on the specific characteristics of the text classification task, including class imbalance, the importance of false positives and false negatives, and the desired trade-offs between precision and recall. It's normal to use multiple metrics to gain a more detailed intuitive understanding of a model's performance.

2.13 Optimization in NLP classification tasks:

Optimizers play a very important role in NLP (Natural Language Processing) classification tasks, as they are responsible for adjusting the model's parameters during training to minimize the loss function. The choice of optimizer can heavily impact the convergence speed, stability, and final performance of the classification model. Following is description of different roles and importance of optimizers in training process of a deep learning model.

I. **Parameter Updates**:
   - Optimizers update the parameters of the classification model during the training process. The updates are based on the gradients of the loss function with respect to these parameters.
   - Gradients represent the direction and magnitude of changes needed to minimize the loss. Optimizers adjust the parameters incrementally in the direction that reduces the loss.

II. **Convergence Speed**:
   - Different optimizers have varying convergence speeds. The convergence speed refers to how quickly the model reaches a point where the loss stops decreasing significantly.
   - Optimizers, like vanilla stochastic gradient descent (SGD), converge slowly. They tend to take smaller steps toward the optimal solution, hence take more iterations to converge.
   - Adaptive optimizers, such as Adam and RMSprop, often converge faster because they adapt the learning rates for each parameter based on the recent gradient history. This adaptability helps speed up convergence.

III. **Stability**:
   - Optimizers are also very critical for stability during the training process. In deep neural networks, gradients can become extremely small (vanishing gradients) or extremely large (exploding gradients) during the training process.
   - Adaptive optimizers like Adam and RMSprop help overcome these issues. They keep adjusting the learning rates for each parameter independently based on the magnitude of past gradient updates. This adaptability help in avoiding learning rates from becoming too small or too large. Hence, contribute to a more stable training.

IV. **Adaptability**:

- Adaptive optimizers continuously adjust learning rates for each parameter during the training process. This adaptability is beneficial for complex NLP models with a large number of parameters.
- For example, in a DL model with millions of parameters, some parameters may require larger learning rates for faster convergence, while others may need smaller rates to avoid overshooting the optima. Adaptive optimizers address these different needs of different parameters in training a DL model.

V. **Hyperparameter Tuning**:

- Optimizers have their own hyperparameters, such as learning rates, momentum terms, and epsilon values. These hyperparameters need to be tuned for optimal performance on the specific NLP classification task.
- The learning rate is one of the most critical hyperparameters and often requires careful tuning. It influences the step size during parameter updates and directly impacts training stability and convergence speed.

VI. **Transfer Learning and Model Generalization**:

- Optimizers can also influence how well a model generalizes to unseen data. Optimizers that converge quickly without overfitting on the training data tend to lead to models with better generalization performance on new, unseen examples.

In summary, the role of optimizers in NLP classification tasks is to guide the training process by controlling how model parameters are updated. The choice of optimizer and its hyperparameters should be made based on practical experimentation, considering the specific characteristics of the NLP task and the model architecture. The right optimizer can significantly impact training stability, convergence speed, and the model's overall performance.

CHAPTER 3 : RESEARCH METHODOLOGY

In the realm of natural language processing and machine learning, the optimization of text classification models represents a critical pursuit. This research methodology outlines a comprehensive investigation into the performance of optimization algorithms within TensorFlow's Keras library for deep learning-based text classification tasks. Our primary objective is to assess and compare the efficacy of these algorithms, guiding the selection of the most suitable optimizer for such tasks. This methodical exploration encompasses data collection, preprocessing, model selection, hyperparameter tuning, and evaluation metrics, providing a framework for empirical analysis and insights that can inform the optimization strategies employed in real-world text classification scenarios.

3.1 Objective:

The primary and overarching objective of this thesis research is to undertake a comprehensive and analysis of optimization algorithms available within TensorFlow's Keras library, specifically concerning their efficacy when applied to the intricate realm of text classification tasks. This endeavor is driven by the aim to gain profound insights into the performance variations among these optimizers in the context of deep learning models for text classification.

3.2 Research Questions:

Which optimization algorithm exhibits the most superior and reliable performance in text classification tasks, particularly within the area of deep learning models?" This central question encompasses a broader set of inquiries that delve into the intricacies of optimizing text classification models for real-world applications.

3.3 Dataset Details:

The research is based on evaluating the models on “Customer Review of Amazon Products” dataset. This is a rich and publicly accessible repository of textual data under license CC BY-NC-SA 4.0 . This is a list of over 34,000 consumer reviews for Amazon products like the Kindle, Fire TV Stick, and more provided by Datafiniti's Product Database. The dataset includes basic product information, rating, review text, and more for each product. This dataset is chosen for its vastness, and relevance to real-world text classification scenarios.

3.4 Data Preprocessing

In the preparatory part of this research, detailed attention and time will be devoted to data preprocessing tasks. These operations are essential to ensure that the raw textual data is transformed into a structured and sanitized form, thereby laying the strong foundation for further analysis.

Basic cleaning procedures will be done first.

- Text Normalization will be done first. Here the text will be converted to lowercase so that regardless of case words are treated same. Then, Punctuation, symbols and non-alphanumeric characters will be removed that don't convey meaningful information. Lastly, abbreviations will be dealt with.
- Any URLs, substrings with @, hyphens, non-alphanumeric characters will also be removed.
- Stop Word Removal will be done to remove common stop words (e.g., "the", "and") that don't add significant information to textual data. Stop words list from English language will be used.
- Lemmatization or stemming will be applied to reduce words to their root form. This helps in treating different inflections of a word as the same (e.g., "running" to "run"). Lemmatization is generally preferred over stemming because it maintains valid word forms.

While the emphasis will primarily be on cleaning procedures, such as noise removal and data formatting, these measures are pivotal in facilitating subsequent word embedding generation.

3.5 Feature Engineering

The core of this research will leverage the powerful capabilities of the Keras embedding function. Keras tokenizer function will be used to pre-process textual data then, embedding function will be employed to transform the unstructured text data into numerical vectors, a process known as word embedding. This is a main step as it converts textual information into a suitable format for deep learning models. This enables DL models to comprehend and understand semantic meanings inside the data and make predictions based on this in-depth learning. This takes into consideration both individual words and their relationship with each other. This gives model capabilities to do very well.

3.6 Model Selection

A salient facet of this research involves the exploration and deployment of a diverse spectrum of neural network architectures. These architectures encompass:

- A Basic Neural Network, serving as a benchmark and baseline.
- The Simple Recurrent Neural Network (RNN), which integrates sequential information into the modeling process.
- The Long Short-Term Memory (LSTM) architecture, known for its superior ability to capture long-range dependencies in sequential data.
- A potential inclusion, the Gated Recurrent Unit (GRU), contingent on available time and resources. GRUs are favored for their computational efficiency while retaining comparable performance characteristics.

3.7 Hyperparameter Tuning

The optimization of model hyperparameters is a pivotal aspect of this research. This process will be executed in two distinct phases to ensure the fine-tuning of model performance. Initially, a manual tuning approach will be deployed, drawing upon domain expertise and insights gleaned from preliminary experiments. This phase will involve iterative adjustments of hyperparameters to arrive at optimal configurations. Subsequently, if resources permit, we will delve into the realm of more sophisticated hyperparameter search techniques. A prime candidate is grid search, a methodical exploration of hyperparameter combinations within predefined ranges, aiming to identify the most favorable settings that maximize model performance.

3.8 Evaluation Metrics

To assess the performance of the models and related optimizers comprehensively, Different evaluation metrics will be used, each offering a unique perspective on model effectiveness trained using different optimizers:

- Accuracy will serve as a global indicator of correct classifications.
- Monitoring Training Loss and Validation Loss over epochs – Training Loss will be monitored to understand the convergence behavior of DL models with different optimizers. Validation Loss will be monitored to understand the model ability to generalize on unseen data and effects of training over epochs.

- Comparison of Final Validation Loss and Validation Accuracy achieved among different optimizers – An overview and comparison will be achieved in comparison to other optimizers.
- Training Time Comparison over fixed epochs - Time required for training models using different optimizers will be monitored.
- For each optimizer, Training Loss comparison with Validation Loss and Training Accuracy comparison with Validation Accuracy over the epochs. – Help understand training behaviour by evaluating training curves of optimizer.

3.9 Cross-Validation

Although cross-validation is a widely employed technique to assess model generalization, this study will not incorporate it due to the specific research focus and resource constraints. Instead, model validation will be conducted on separate, held-out test datasets to ascertain their real-world applicability.

3.10 Experiments and Comparison

The central thrust of this research revolves exclusively around the evaluation and comparison of optimization algorithms within the context of text classification tasks. As such, this study will not involve direct comparisons with baseline models. Rather, the focus will be on establishing a comprehensive understanding of how different optimizers affect model performance in a real-world text classification setting.

3.11 Ethical Considerations and related implications

This research operates within the framework of openly accessible and publicly available data. Therefore, ethical considerations relating to privacy, bias, or data sensitivity do not apply.

3.12 Tools and Libraries

Practical implementation of this research will be facilitated by a carefully selected set of tools, programming languages, and libraries:

- Python will serve as the primary programming language, known for its versatility and extensive ecosystem.
- TensorFlow and Keras will provide the foundational infrastructure for deep learning model development, leveraging their robustness and ease of use.
- scikit-learn will complement the analysis with additional machine learning capabilities.
- NumPy and pandas will empower the manipulation and structuring of data, ensuring its readiness for model ingestion.
- Finally, Kaggle's platform will serve as the primary tool for executing code experiments, providing an environment conducive to rigorous model evaluation and experimentation. This Platform removes requirements of any specific Hardwares.
- Following are details of all python packages

I. Tensorflow v 2.12.0
II. python v3.10
III. matplotlib v3.7.0
IV. seaborn v0.12.2
V. Numpy v1.24.2
VI. NLTK v3.8
VII. SpaCy v3.4
VIII. Gensim v3.2.0

This research methodology outlines a systematic exploration of optimization algorithms within TensorFlow's Keras library for text classification. The study aims to determine the most effective optimizer for deep learning-based text classification tasks. The methodology encompasses data preparation, model selection, hyperparameter tuning, and the use of comprehensive evaluation metrics. Through this methodical approach, the research seeks to provide valuable insights for optimizing text classification models, contributing to the advancement of natural language processing and machine learning applications.

CHAPTER 4 : IMPLEMENTATION

In this important chapter, we delve into the core of our research, starting on an insightful analysis of optimizers available for text classification tasks using DL models. We systematically dissect each element of our methodology, from data preprocessing to model implementation and hyperparameter tuning. Through rigorous evaluation metrics, we assess the performance of optimization algorithms and compare our findings to the research objectives.

4.1 Data Preprocessing and Preparation

In this section, we thoroughly analyze the data preprocessing and preparation steps. First of all, important libraries for pre-processing the data is imported. Libraries like Numpy, Pandas, seaborn, plotly, NLTK, sklearn, tensorflow, keras and other important modules like os, time imported in Kaggle notebook.

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LinearRegression

from collections import Counter
import nltk
import seaborn as sns
import string
from nltk.corpus import stopwords

# import re
# from autocorrect import spell
import regex as re

#from keras.preprocessing.sequence import pad_sequences

# Import all other relevant libraries

from tensorflow.keras.models import Sequential
from keras.preprocessing.text import Tokenizer
from tensorflow.keras.layers import Embedding, Dense, Flatten, LSTM, Dropout
from keras.layers import Dense
from keras.backend import eval
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Dense, Flatten
from tensorflow.keras.optimizers import SGD, Adam, RMSprop, Adagrad, Nadam, Adadelta, Adamax, AdamW, Adafactor, Ftrl
import time

import matplotlib.pyplot as plt
import plotly.graph_objects as go
import plotly.express as px

import os
print(os.listdir("../input"))
```

Figure 4.1 Code Block – Import required libraries

The dataset contains 34659 rows and 21 columns. Each column gives unique details of reviews. Details of each column is available in screenshot below:

Table 4.1 Dataset information

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 34660 entries, 0 to 34659
Data columns (total 21 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   id                    34660 non-null  object
 1   name                  27900 non-null  object
 2   asins                 34658 non-null  object
 3   brand                 34660 non-null  object
 4   categories            34660 non-null  object
 5   keys                  34660 non-null  object
 6   manufacturer          34660 non-null  object
 7   reviews.date          34621 non-null  object
 8   reviews.dateAdded     24039 non-null  object
 9   reviews.dateSeen      34660 non-null  object
 10  reviews.didPurchase   1 non-null      object
 11  reviews.doRecommend   34066 non-null  object
 12  reviews.id            1 non-null      float64
 13  reviews.numHelpful    34131 non-null  float64
 14  reviews.rating        34627 non-null  float64
 15  reviews.sourceURLs    34660 non-null  object
 16  reviews.text          34659 non-null  object
 17  reviews.title         34654 non-null  object
 18  reviews.userCity      0 non-null      float64
 19  reviews.userProvince  0 non-null      float64
 20  reviews.username      34653 non-null  object
dtypes: float64(5), object(16)
memory usage: 5.6+ MB
```

Dataset is opened and checked for structure and what information is available in which column.

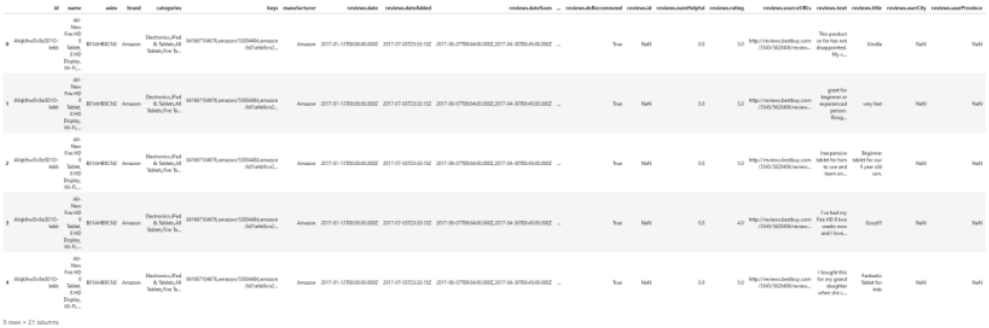


Figure 4.2 Overview of Dataset

A basic grouping operation is done on “reviews.rating” column to understand the distribution ratings. 23775 users gave 5 rating and 8541 gave 4 rating. Hence, a total of 32316 reviews can be considered as positive. 1499 gave a rating of 3 which seems neutral. Similarly, 410 gave 1 rating and 402 users gave 2 rating so in total 812 users posted a negative review on product.

	reviews.rating	No of Users
0	5.0	23775
1	4.0	8541
2	3.0	1499
3	1.0	410
4	2.0	402

Figure 4.3 Distribution of reviews

After a review of dataset it is clear that reviews are available in “reviews.text” column and rating is available in “reviews.rating”. Also dropped the rows with no information.

	reviews.rating	reviews.text	reviews.title	reviews.username
0	5.0	This product so far has not disappointed. My c...	Kindle	Adapter
1	5.0	great for beginner or experienced person. Boug...	very fast	truman
2	5.0	Inexpensive tablet for him to use and learn on...	Beginner tablet for our 9 year old son.	DaveZ
3	4.0	I've had my Fire HD 8 two weeks now and I love...	Good!!!	Shacks
4	5.0	I bought this for my grand daughter when she c...	Fantastic Tablet for kids	explore42

A suitable label is created to categorize the ratings into two categories. All reviews having ratings more than or equal to 3 is treated as positive and other as negative reviews.

	reviews.rating	reviews.text	reviews.title	reviews.username	label
0	5.0	This product so far has not disappointed. My c...	Kindle	Adapter	1
1	5.0	great for beginner or experienced person. Boug...	very fast	truman	1
2	5.0	Inexpensive tablet for him to use and learn on...	Beginner tablet for our 9 year old son.	DaveZ	1
3	4.0	I've had my Fire HD 8 two weeks now and I love...	Good!!!	Shacks	1
4	5.0	I bought this for my grand daughter when she c...	Fantastic Tablet for kids	explore42	1

Now reviews are available in two categories.

```
label
1    33803
0      811
Name: count, dtype: int64
```

Now, it is very important to do extensive cleaning of the text reviews because it may have multiple type of data issues. To do multiple process simultaneously on the text a simple and efficient function is written which returns clean documents.

```python
# Set stop words
stop = set(stopwords.words('english'))

# define function to do multiple cleaning process in one go.
def clean_document(doco):
    punctuation = string.punctuation
    punc_replace = ''.join([' ' for s in punctuation])
    doco_link_clean = re.sub(r'http\S+', '', doco)
    doco_clean_and = re.sub(r'&\S+', '', doco_link_clean)
    doco_clean_at = re.sub(r'@\S+', '', doco_clean_and)
    doco_clean = doco_clean_at.replace('-', ' ')
    doco_alphas = re.sub(r'\W +', ' ', doco_clean)
    trans_table = str.maketrans(punctuation, punc_replace)
    doco_clean = ' '.join([word.translate(trans_table) for word in doco_alphas.split(' ')])
    doco_clean = doco_clean.split(' ')
    p = re.compile(r'\s*\b(?=[a-z\d]*([a-z\d])\1{3}|\d+\b)[a-z\d]+', re.IGNORECASE)
    doco_clean = ([p.sub("", x).strip() for x in doco_clean])
    doco_clean = [word.lower() for word in doco_clean if len(word) > 2]
    doco_clean = ([i for i in doco_clean if i not in stop])
    doco_clean = ([p.sub("", x).strip() for x in doco_clean])
    return doco_clean
```

Figure 4.4 Code Block – Cleaning of text data

The code above is a function that cleans a document by removing certain types of elements such as URLs, special characters, and stop words, and by performing other text processing operations. Here is a detailed explanation of each step:

I. `stop = set(stopwords.words('english'))`:
   - This line initializes a set `stop` that contains common English stopwords. These stopwords are words that are considered too common and are removed during text processing to focus on the significant words.

II. `def clean_document(doco)`:
   - This line begins the definition of a function called `clean_document` that takes an input parameter `doco`, which represents the document to be cleaned.

III. Cleaning steps inside the function:

- `punctuation = string.punctuation`: This line initializes a variable `punctuation` that stores all the punctuation marks in the string module.
- `punc_replace = ''.join([' ' for s in punctuation])`: This line creates a string `punc_replace` with spaces in place of punctuation marks, essentially preparing for the removal of punctuation.
- `doco_link_clean = re.sub(r'http\S+', '', doco)`: This line uses regular expressions to remove any URLs from the document.
- `doco_clean_and = re.sub(r'&\S+', '', doco_link_clean)`: This line removes any special characters starting with '&'.
- `doco_clean_at = re.sub(r'@\S+', '', doco_clean_and)`: This line removes any words starting with '@', typically used for mentions in social media or emails.
- `doco_clean = doco_clean_at.replace('-', ' ')`: This line replaces hyphens with spaces.
- `doco_alphas = re.sub(r'\W +', ' ', doco_clean)`: This line removes any non-alphanumeric characters except spaces.
- `trans_table = str.maketrans(punctuation, punc_replace)`: This line creates a translation table to replace punctuation with spaces.
- `doco_clean = ' '.join([word.translate(trans_table) for word in doco_alphas.split(' ')])`: This line applies the translation table to remove punctuation from the document.
- `p = re.compile(r'\s*\b(?=[a-z\d]*([a-z\d])\1{3}|\d+\b)[a-z\d]+', re.IGNORECASE)`: This line compiles a regular expression pattern to find repetitive characters or digits in the document.
- `doco_clean = ([p.sub("", x).strip() for x in doco_clean])`: This line removes repetitive characters or digits from the document.
- `doco_clean = [word.lower() for word in doco_clean if len(word) > 2]`: This line converts all words to lowercase and keeps only those words that have a length greater than 2.
- `doco_clean = ([i for i in doco_clean if i not in stop])`: This line removes any stopwords from the cleaned document.
- `doco_clean = ([p.sub("", x).strip() for x in doco_clean])`: This line again removes any repetitive characters or digits from the document.

- `return doco_clean`: This line returns the cleaned document as the output of the function.

In short `clean_document` function essentially performs a series of text processing steps, including removing URLs, special characters, repetitive characters or digits, and stopwords, as well as converting all words to lowercase and removing words with a length less than or equal to 2. The function then returns the cleaned document as a list of words.

4.2 Embedding Creation

Embedding creation is now done for that `Tokenizer` class from Keras library is used:

Code working is as follows:

```
# Keras Tokenizer function
tokenizer = Tokenizer()
tokenizer.fit_on_texts(sentences)

text_sequences = np.array(tokenizer.texts_to_sequences(sentences))
sequence_dict = tokenizer.word_index
word_dict = dict((num, val) for (val, num) in sequence_dict.items())
```

Figure 4.5 Code Block - Embedding Creation

I. **Tokenizer Initialization:**
   - The line `tokenizer = Tokenizer()` initializes a Tokenizer object, which is used for converting text data into a format suitable for deep learning models.

II. **Fitting on Texts:**
   - The line `tokenizer.fit_on_texts(sentences)` fits the tokenizer on the provided list of `sentences`. This step involves tokenizing the text and creating an internal vocabulary based on the words in the `sentences` list. Each word is assigned a unique integer based on its frequency in the text corpus.

III. **Texts to Sequences Conversion:**
   - The line `text_sequences = np.array(tokenizer.texts_to_sequences(sentences))` converts the `sentences` into sequences of integers. Each sequence represents the words in the sentences with their corresponding indices based on the tokenizer's internal word index. The resulting sequences are converted into a NumPy array and stored in the variable `text_sequences`.

IV. **Sequence Dictionary Creation:**
   - The line `sequence_dict = tokenizer.word_index` retrieves the word-to-index mappings from the tokenizer. The `word_index` attribute provides a dictionary where the keys are words, and the values are their corresponding integer indices. This dictionary is stored in the variable `sequence_dict`.

V. **Word Dictionary Creation:**

- The line `word_dict = dict((num, val) for (val, num) in sequence_dict.items())` creates a dictionary `word_dict` by reversing the key-value pairs of the `sequence_dict`. This dictionary maps the integer indices back to the corresponding words, facilitating the mapping of indices to their original textual representations.

This code segment uses the Tokenizer class to convert text data into sequences of integers, and then create dictionaries for mapping words to indices and vice versa.

The `pad_sequences` function is now used on the textual data to achieve following things:

I. Maintaining Uniform Sequence Lengths: DL models require input data of uniform length. In the context of text or sequence data, different samples may have varying lengths. Padding ensures that all sequences have the same length, enabling the data to be efficiently processed in a batch during model training.

II. Sequences for Input into RNNs and LSTMs**:** Recurrent Neural Networks (RNNs) and Long Short-Term Memory networks (LSTMs) are commonly used for processing sequential data. These networks expect fixed-length sequences as input. Padding is necessary to ensure that all sequences are of the same length before feeding them into these models.

III. Batch Processing in Deep Learning: In deep learning, data is often processed in batches to improve computational efficiency. Having sequences of the same length allows for easier batch processing.

Now, further data is shuffled to randomize the order of reviews.

```
# Shuffing of the position of reviews
np.random.seed(1024);
random_posits = np.arange(len(X))
np.random.shuffle(random_posits);
```

```
X = X[random_posits];
Y = Y[random_posits];
```

By shuffling the positions of the reviews, it is ensured that the order of the reviews is randomized, which can be beneficial for various purposes, such as creating randomized training and test sets, preventing any bias that might arise from the original ordering.

After this randomization Train, Test and Validation split is done.

```
# define proportion of splits
train_cap = int(0.70 * len(X));
dev_cap = int(0.85 * len(X));

# code to make train, test & validation splits
X_train, Y_train = X[:train_cap], Y[:train_cap]
X_dev, Y_dev = X[train_cap:dev_cap], Y[train_cap:dev_cap]
X_test1, Y_test1 = X[dev_cap:], Y[dev_cap:]
```

Figure 4.6 Code Block - Data Splitting

Now all pre-processing work is completed, and we have data ready for training the deep learning models using different optimizers and settings.

From above operations the following targets are achieved:
Cleaning, stop word removal, and pre-processing streamlined the data, reduced noise, and enhanced the relevance of features. These steps ensured that we have meaningful dataset, enabling the model to capture the essential patterns within the text. By normalizing the text and removing irrelevant words, this will improve the accuracy and efficiency of the models being trained in next step. It will lead to more robust and reliable text classification results.

What were the effects of tokenization and word embedding creation on the data's structure?
With tokenization and word embedding, the following things are achieved. Tokenization played most important role in structuring textual data by segmenting text into smaller units, also facilitated normalization, handled punctuation, and reshaped data into sequences of tokens. While word embedding creation further impacted the data's structure by reducing dimensionality, preserved semantic information, enabled similarity measurement, and facilitated representation learning, ultimately this will help DL models to comprehend and process this text classification task in a more efficient and context-aware manner.

During data preparation stages no specific challenges faced. One thing which is noticed that the negative reviews are quite less compared to positive reviews. This dataset is imbalanced, here we are not taking up extensive efforts to make this dataset balanced since our main focus is to assess the performance of different optimizers in training different DL models. This may be further evaluated during course of this research. In case there is need to address imbalanced dataset task will be taken if time permits.

4.3 Experiment Creation - Setting up different Model for evaluating optimizers.

This sub-section is focused on creating Neural Networks and setting up standard models. These standard models will be trained on the same data keeping all the data and parameters same in a control manner. Only optimizers will be changed, and the training data will be recorded to understand the effects and efficacy of these optimizers in training deep neural networks in text classification tasks.

4.3.1 Basic NN Model

```
def create_model(optimizer):
    model= Sequential()
    model.add(Embedding(len(word_dict) + 1, max_cap, input_length=max_cap))
    model.add(Flatten())  # Flatten the output to connect to Dense layers
    model.add(Dense(25, activation='relu'))
    model.add(Dense(2, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    return model
```

Figure 4.7 Code Block - Basic NN Model

The first basic model for evaluation purpose is as shown above. This defines a simple sequential deep learning model for text classification. It initializes a Sequential model and adds layers sequentially, starting with an Embedding layer for word embeddings, where the length of the word dictionary plus one is used as the input dimension and max_cap as the input length. The Flatten layer is then used to flatten the output for connection to Dense layers. Two Dense layers follow, the first with 25 neurons activated by the rectified linear unit (ReLU) function, and the final layer with 2 neurons activated by the sigmoid function for binary classification. The model is compiled with the binary cross-entropy loss function, the specified optimizer (which will be iterated over a list in following section), and accuracy as the metric for evaluation.

4.3.2 Recurrent Neural Networks with LSTM Units

```
def create_model(optimizer):
    model = Sequential();
    model.add(Embedding(len(word_dict)+1, max_cap, input_length=max_cap));
    #adding a LSTM layer of dim 1--
    model.add(LSTM(25, return_sequences=True));
    model.add(LSTM(25, return_sequences=False));
    #adding a dense layer with activation function of relu
    model.add(Dense(25, activation='relu'));#best 50,relu
    #adding the final output activation with activation function of softmax
    model.add(Dense(2, activation='sigmoid'));
    model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    return model
```

Figure 4.8 Code Block - RNN with LSTM Units

The second model is a sequential deep learning model for text classification. The model begins with an embedding layer, which converts input sequences of text represented by integers into continuous-valued word embeddings. It is followed by two Long Short-Term Memory (LSTM) layers, each consisting of 25 units, with the first LSTM layer returning sequences and the second one returning only the final output. Subsequently, a dense layer with a rectified linear unit (ReLU) activation function is added, which serves as a hidden layer for feature transformation. The model concludes with a dense output layer employing a sigmoid activation function, suitable for binary classification tasks. The model is compiled with the binary cross-entropy loss function, the specified optimizer (which will be iterated over a list in following section), and accuracy as the metric for evaluation.

4.3.3 Recurrent Neural Networks with GRU Units

```
def create_model(optimizer):
    model = Sequential();
    model.add(Embedding(len(word_dict)+1, max_cap, input_length=max_cap));
    #adding a LSTM layer of dim 1--
    model.add(GRU(25, return_sequences=True));
    model.add(GRU(25, return_sequences=False));
    #adding a dense layer with activation function of relu
    model.add(Dense(25, activation='relu'));#best 50,relu
    #adding the final output activation with activation function of softmax
    model.add(Dense(2, activation='sigmoid'));
    model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    return model
```

Figure 4.9 RNN with GRU Units

The third model is a sequential deep learning model for text classification. The model initiates with an embedding layer, transforming input sequences of text represented by integers into continuous-valued word embeddings. It subsequently integrates two Gated Recurrent Unit

(GRU) layers, each with 25 units, where the first GRU layer returns sequences, while the second one returns only the final output. Following the GRU layers, a dense layer is included, utilizing the rectified linear unit (ReLU) activation function to facilitate feature transformation. Finally, the model is equipped with a dense output layer employing a sigmoid activation function, rendering it suitable for binary classification tasks. The model is compiled with the binary cross-entropy loss function, the specified optimizer (which will be iterated over a list in following section), and accuracy as the metric for evaluation.

Here, in this part three different types of Neural networks suitable for text classification are defined. These 3 neural networks will be further trained using list of all currently available optimizers in Keras Library. Following is list of optimizers that will be used to train the DL neural network.

1. SGD
2. Adam
3. RMSprop
4. Adagrad
5. Nadam
6. Adadelta
7. Adamax
8. AdamW
9. Adafactor

Of these Adam is most famous and widely used. Other relatively new AdamW and Adafactor will be evaluated for their performance.

4.4 Hyperparameter Tuning

Here, we assess the impact of hyperparameter optimization on training capabilities of different optimizers.

3 different standard models were trained using the same learning rate = 0.001 for all optimizers. Other unique hyperparameters of each optimizer were either left to default values or not tuned. As per available heuristic, these unique hyperparameters of optimizers are updated. Now with optimizers with tuned hyperparameters. Earlier created Basic NN model trained, and the performance will be compared in next section.

```
# Loop through each optimizer
for optimizer_name in optimizers:
    optimizer = None
    if optimizer_name == 'SGD':
        # Adjust learning rate and momentum
        optimizer = SGD(lr=0.01, momentum=0.9)
    elif optimizer_name == 'Adam':
        # Adjust learning rate, beta_1, beta_2, and epsilon
        optimizer = Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-07)
    elif optimizer_name == 'RMSprop':
        # Adjust learning rate and rho
        optimizer = RMSprop(lr=0.001, rho=0.9)
    elif optimizer_name == 'Adagrad':
        # Adjust learning rate
        optimizer = Adagrad(lr=0.01, epsilon=1e-07)
    elif optimizer_name == 'Nadam':
        # Adjust learning rate, beta_1, beta_2, and epsilon
        optimizer = Nadam(lr=0.002, beta_1=0.9, beta_2=0.999, epsilon=1e-07)
    elif optimizer_name == 'Adadelta':
        # Adjust learning rate, rho, and epsilon
        optimizer = Adadelta(lr=1.0, rho=0.95, epsilon=1e-07)
    elif optimizer_name == 'Adamax':
        # Adjust learning rate, beta_1, beta_2, and epsilon
        optimizer = Adamax(lr=0.002, beta_1=0.9, beta_2=0.999, epsilon=1e-07)
    elif optimizer_name == 'AdamW':
        # Adjust learning rate, beta_1, beta_2, epsilon, and weight_decay
        optimizer = AdamW(lr=0.002, beta_1=0.9, beta_2=0.999, epsilon=1e-07, weight_decay=0.01)
    elif optimizer_name == 'Adafactor':
        # Adjust learning rate, decay_rate, beta_1, epsilon1, and epsilon2
        optimizer = Adafactor(lr=0.002, beta_2_decay=-0.8, epsilon_1=1e-30, epsilon_2=0.001)
```

Figure 4.10 For Loop - Tuned hyperparameters of Optimizers

Updated hyperparameters in the provided code are aligned with common heuristics for text classification tasks. Here is a brief explanation for each optimizer:

I. **SGD (Stochastic Gradient Descent):** The learning rate and momentum are adjusted. A higher momentum value (0.9) can help accelerate convergence, especially in the presence of noisy gradients.

II. **Adam:** The learning rate, beta_1, beta_2, and epsilon are adjusted. Common choices for beta_1 and beta_2 is 0.9 and 0.999, respectively, and epsilon is usually set to a small value (e.g., 1e-07).

III. **RMSprop:** The learning rate and rho are adjusted. Rho is typically set to 0.9 for text classification tasks.

IV. **Adagrad (Adaptive Gradient Algorithm):** The learning rate is adjusted. Adagrad adapts the learning rate to the parameters, often requiring a higher initial learning rate.

V. **Nadam (Nesterov-accelerated Adaptive Moment Estimation):** The learning rate, beta_1, beta_2, and epsilon are adjusted. Nadam combines Adam with Nesterov's accelerated gradient, making it suitable for complex optimization tasks.

VI. **Adadelta:** The learning rate, rho, and epsilon are adjusted. Adadelta is known for its robustness to different settings and requires less manual tuning of the learning rate.

VII. **Adamax:** The learning rate, beta_1, beta_2, and epsilon are adjusted. Adamax is a variant of the Adam algorithm that is often used in settings with sparse gradients.

VIII. **AdamW:** The learning rate, beta_1, beta_2, epsilon, and weight_decay are adjusted. AdamW is an adaptation of the Adam algorithm that incorporates weight decay regularization.

IX. **Adafactor:** The learning rate, decay_rate, beta_1, epsilon1, and epsilon2 are adjusted. Adafactor is known for its ability to adapt to the geometry of the loss surface and can handle large learning rates without instabilities.

These adjustments are based on general heuristics for optimizing different optimizers for text classification tasks. Most of time adjustments to these hyperparameters are necessary depending on the specific characteristics of the dataset and the dynamics observed during the training process.

4.5 Evaluation Metrics & Criteria for evaluating optimizer performance.

This section is dedicated to analyzing performance and working of different optimizers in training text classification models.

4.5.1 Criteria 1: Monitoring Training Loss and Validation Loss over epochs

Monitoring both training loss and validation loss over epochs in deep learning models is very important. Main reasons are listed below:

1. **Overfitting Detection:** Comparing the trends of training loss and validation loss helps to identify overfitting. If the training loss continues to decrease while the validation loss starts to increase or remains stagnant, it indicates that the model is overfitting the training data, emphasizing the need for adjustments such as regularization techniques or model simplification.
2. **Generalization Assessment:** Validation loss represents the model's performance on unseen data, allowing for the evaluation of its generalization capability. A consistent or decreasing validation loss implies that the model is performing well on data it has not been trained on, ensuring that the model has learned meaningful patterns rather than memorizing the training data.
3. **Hyperparameter Tuning and Model Selection:** Analyzing both training and validation losses assists in selecting the best model architecture and hyperparameters. This ensures that the chosen model not only fits the training data well but also generalizes effectively to unseen data, leading to improved overall model performance and reliability.
4. **Optimization Evaluation:** Observing both training and validation losses provides insights into the effectiveness of the optimization algorithm being used. A well-chosen optimization algorithm should result in a gradual decrease in both training and validation losses, indicating that the model is learning effectively without overfitting or underfitting the data.
5. **Performance Evaluation and Interpretability:** Both training and validation losses provide a comprehensive understanding of the model's learning progress and performance. By analyzing these losses, one can assess the model's behavior, interpret its learning process, and make informed decisions about potential improvements or adjustments to enhance its overall performance.

When training a neural network, the ideal behavior of training loss and validation loss is as follows:

1. **Training Loss Behavior:** The training loss should generally exhibit a decreasing trend over epochs. This indicates that the model is learning from the data and is converging towards an optimal solution. However, an excessively fast decrease or erratic behavior may suggest that the learning rate is too high, potentially causing the model to overshoot the optimal parameters or struggle to converge effectively.
2. **Validation Loss Behavior:** The validation loss should initially decrease or remain stable as the model learns from the training data. As training progresses, the validation loss should continue to decrease or stabilize, indicating that the model is not only learning the training data but also generalizing well to unseen data. A slight increase in validation loss is acceptable, but a consistent increase or a significant gap between training and validation losses might suggest overfitting.

Therefore, the best behavior for training loss and validation loss is as follows:

1. Training loss steadily decreases or stabilizes, indicating effective model convergence.
2. Validation loss decreases or stabilizes, suggesting good generalization and a model that is not overfitting the training data.

Ideally, **both the training and validation losses should decrease and stabilize in tandem.** This demonstrates that the model is learning the underlying patterns of the data without overfitting or underfitting. Such behavior indicates a well-generalized model that can perform effectively on unseen data.

To achieve this Plotly is used to plot this convergence graphs and also related dataframe is printed.

```python
# Plot training loss over time for each optimizer
fig = go.Figure()
for i, (optimizer_name, history) in enumerate(history_dict.items()):
    fig.add_trace(go.Scatter(x=list(range(epochs)), y=history['loss'], mode='lines', name=optimizer_name, line=dict(color=colors[i % len(colors)])))
fig.update_layout(title='Training Loss Convergence', xaxis_title='Epoch', yaxis_title='Training Loss')
fig.show()


# Prepare the data
data = {'Epoch': list(range(epochs))}
for i, (optimizer_name, history) in enumerate(history_dict.items()):
    data[optimizer_name] = history['loss']

# Create a DataFrame
df = pd.DataFrame(data)

# Transpose the DataFrame
df_transposed = df.set_index('Epoch').T

# Print the transposed DataFrame
print(df_transposed)
```

Figure 4.11 Code Block – Plotting Training Loss convergence & data

```python
# Plot validation loss over time for each optimizer
fig = go.Figure()
for i, (optimizer_name, history) in enumerate(history_dict.items()):
    fig.add_trace(go.Scatter(x=list(range(epochs)), y=history['val_loss'], mode='lines', name=optimizer_name, line=dict(color=colors[i % len(colors)])))
fig.update_layout(title='Validation Loss Convergence', xaxis_title='Epoch', yaxis_title='Validation Loss')
fig.show()


# Prepare the data for validation loss
val_data = {'Epoch': list(range(epochs))}
for i, (optimizer_name, history) in enumerate(history_dict.items()):
    val_data[optimizer_name] = history['val_loss']

# Create a DataFrame for validation loss
val_df = pd.DataFrame(val_data)

# Transpose the DataFrame for validation loss
val_df_transposed = val_df.set_index('Epoch').T

# Print the transposed DataFrame for validation loss
print(val_df_transposed)
```

Figure 4.12 Code Block – Plotting Validation Loss Convergence and Data

Based on the plots generated in multiple experimental settings. Optimizer performance based on these criteria will be discussed in Results section.

4.5.2 Criteria 2: Comparison of Final Validation Loss and Validation Accuracy achieved among different optimizers.

As discussed earlier, Validation loss and validation accuracy are very important in evaluating the performance and generalization capability of a deep learning model. Validation loss reflects how well the model performs on unseen data, while validation accuracy measures the proportion of correctly classified samples. Discrepancies between high accuracy and increasing loss can indicate overfitting, emphasizing the need for model adjustments. These metrics are instrumental in hyperparameters tuning and model selection, guiding decisions to strike a balance between effective learning and generalization. Monitoring these measures is crucial for ensuring the model's reliable performance on new, unseen data.

```
# Plot the validation loss and accuracy results
fig = go.Figure()
fig.add_trace(go.Bar(x=results['optimizer'], y=results['loss'], name='Validation Loss', marker_color=colors))
fig.update_layout(title_text='Validation Loss', yaxis_title='Loss')
fig.show()

fig = go.Figure()
fig.add_trace(go.Bar(x=results['optimizer'], y=results['accuracy'], name='Validation Accuracy', marker_color=colors))
fig.update_layout(title_text='Validation Accuracy', yaxis_title='Accuracy')
fig.show()

#######################################

# Create dictionaries for the data
data_loss = {'Optimizer': results['optimizer'], 'Validation_Loss': results['loss']}
data_accuracy = {'Optimizer': results['optimizer'], 'Validation_Accuracy': results['accuracy']}

# Create DataFrames
df_loss = pd.DataFrame(data_loss)
df_accuracy = pd.DataFrame(data_accuracy)

# Merge the DataFrames
df_combined = pd.merge(df_loss, df_accuracy, on='Optimizer')

# Print the combined DataFrame
print(df_combined)
```

Figure 4.13 Code Block – Final Validation Loss & Accuracy comparison

4.5.3 Criteria 3: Training Time Comparison

Comparing training times when training neural networks with different optimizers is essential for several reasons:

1. **Resource Allocation:** Understanding the training time differences help allocate computational resources effectively. Optimizers with shorter training times can enable faster experimentation and model development, which is crucial in time-sensitive projects or when testing multiple architectures and hyperparameters.
2. **Scalability and Deployment:** Models trained with faster optimizers can be more scalable, allowing for quicker deployment in real-world applications. Reduced training times can facilitate rapid updates and adaptations to evolving data, enabling timely model deployment in dynamic environments.
3. **Efficiency and Cost-Effectiveness:** Optimizers with shorter training times can lead to improved computational efficiency and cost-effectiveness. Reduced training times can directly translate to lower computational costs, making the training process more affordable and accessible, especially in resource-constrained environments.
4. **Performance Optimization:** Assessing training times can provide insights into the trade-offs between training efficiency and model performance. While faster optimizers might offer shorter training times, they might not necessarily result in the best-

performing models. Balancing training time and model performance is crucial for optimizing the training process and achieving the desired balance between training efficiency and model accuracy.

Considering the importance of these factors, training time comparison serves as a valuable metric for selecting the most suitable optimizer that aligns with the specific requirements and constraints of the given deep learning project.

```
# Find the maximum training time
max_time = max(training_times.values())

# Calculate percentages based on the maximum time
percentages = [round((time / max_time) * 100, 2) for time in training_times.values()]

# Create a bar chart to visualize training times as percentages relative to the longest time
fig = go.Figure()
fig.add_trace(go.Bar(x=list(training_times.keys()), y=percentages, marker_color=colors))
fig.update_layout(title_text='Training Time Comparison', yaxis_title='Training Time (%)')
fig.show()


# Find the maximum training time
max_time = max(training_times.values())

# Create lists to store the data
optimizer_names = list(training_times.keys())
original_times = list(training_times.values())
percentages = [round((time / max_time) * 100, 2) for time in training_times.values()]

# Create a dictionary for the data
data = {'Optimizer': optimizer_names, 'Time for trainings': original_times, 'Time_%_compared_to_max': percentages}

# Create a DataFrame
df = pd.DataFrame(data)

# Print the DataFrame
print(df)
```

Figure 4.14 Code Block – Plotting Training Time of Optimizers

Above Code Plots Training Time taken for training by different optimizers as percentage to maximum time taking optimizer.

4.5.4 Criteria 4: For each optimizer, Training Loss comparison with Validation Loss and Training Accuracy comparison with Validation Accuracy over the epochs.

As discussed earlier also, comparing training loss and accuracy with validation loss and accuracy for each optimizer is very important in deep learning as it enables the assessment of model performance, aiding in the detection of overfitting or under fitting issues. This comparison assists in optimizer selection, allowing for the identification of the most suitable

optimizer that yields low training and validation losses along with high training and validation accuracies, ensuring effective learning and generalization. Additionally, monitoring the consistency between training and validation metrics over epochs enables the assessment of model stability, ensuring the development of robust and well-performing deep learning models.

```
# Plot the training and validation loss for each optimizer
fig, axs = plt.subplots(len(history_dict), 2, figsize=(12, 4 * len(history_dict)))
for i, (optimizer_name, history) in enumerate(history_dict.items()):
    axs[i, 0].plot(history['loss'], label='Training Loss', linestyle='dashed')
    axs[i, 0].plot(history['val_loss'], label='Validation Loss')
    axs[i, 0].set_title(f'Training & Validation Loss ({optimizer_name})')
    axs[i, 0].set_xlabel('Epochs')
    axs[i, 0].set_ylabel('Loss')
    axs[i, 0].legend()
    axs[i, 0].grid(True)

    axs[i, 1].plot(history['accuracy'], label='Training Accuracy', linestyle='dashed')
    axs[i, 1].plot(history['val_accuracy'], label='Validation Accuracy')
    axs[i, 1].set_title(f'Training & Validation Accuracy ({optimizer_name})')
    axs[i, 1].set_xlabel('Epochs')
    axs[i, 1].set_ylabel('Accuracy')
    axs[i, 1].legend()
    axs[i, 1].grid(True)

plt.tight_layout()
plt.show()
```

Figure 4.15 Code Block – Individual Plot of each optimizer for training curves

Above code plots comparison of Training loss against validation loss and Training accuracy against validation accuracy. Further results will be discussed in next following sections.

4.6 Practical Implications

This experiment's practical implications include informed decision-making, resource optimization, improved model generalization, and robustness, leading to the development of effective and reliable deep learning models for real-world applications. There are multiple optimizers available for training deep learning neural networks. In this study, these optimizers working is evaluated in context of text classification. It is believed that this study will help researcher in future to conduct experiment on these line and industry folks to create real-world solution on text classification tasks. If we know when to use which optimizers for which tasks. It will help save precious environment and preserve its resources.

4.7 Limitations and Future Work

This work of evaluating all available optimizers in Keras is limited to text classification task and based on one single dataset. The dataset was also inclined to positive reviews also. Hence, generalizing the results and work of this study should be properly taken care. Related work should be searched for and studied before taking any decision. Some further similar studies will pave the way for generalization of these results. In future this work can be extended to other different datasets first and then on different types of text classification tasks. Very advanced techniques like BERT should also be evaluated with different optimizers.

4.8 Summary

Chapter 4 shows detailed implementation of the research methodology. The chapter begins with an introduction of the implementation process. The implementation steps are then presented in different sections, each addressing a specific aspect of the methodology. Each section provides details of code employed and explain the understanding on matter for completing the specified step of research methodology. This provides a comprehensive framework for evaluating and interpreting research findings.

CHAPTER 5 : RESULTS AND EVALUATION

5.1 Introduction

In this pivotal chapter, we delve into the heart of our research findings and provide a comprehensive evaluation of the outcomes of our text classification optimization study. We begin by presenting the empirical results of our experimentation, followed by an in-depth analysis of these results. Our analysis is structured to address key research questions, offer insights, and draw meaningful conclusions based on the data generated through our research methodology.

5.2 Data Cleaning and Pre-Processing

Data Cleaning and Pre-Processing is thoroughly done, and all steps and procedures are explained in Implementation Chapter 4. All these operations of cleaning and pre-processing resulted in a clean and suitable dataset for further experimentation. Special function defined for cleaning all reviews in single step. Later these cleaned reviews were used for creating suitable embeddings by tokenizer() function of Keras Library. Train, Test and validation split also done after shuffling the positions of the reviews. From all these operations the dataset became suitable for our requirement of experiments.

5.3 Experimentation Design – Evaluation of Optimizer on different types of Neural Networks

5.3.1 Experiment No. 1 – Basic Neural Networks

The Basis Neural Networks is implemented as discussed in implementation chapter. Later on this neural network is trained using different optimizers. The code below shows the process of iteratively training each optimizer on the same basic Neural Network.

This code block evaluates various optimizers in Basic NN model, records the training history, training times, and the final loss and accuracy results for each optimizer. Here's a detailed explanation:

1. The list optimizers includes the names of the optimizers to be evaluated.
2. Three dictionaries are initialized:
    - history_dict is used to store the training history for each optimizer.
    - training_times is used to store the training times for each optimizer.
    - results is used to store the final loss and accuracy results for each optimizer.
3. The variable epochs represent the number of training epochs for the model.
4. The code then loops through each optimizer, initializes the optimizer with the specified learning rate, and creates a model using the create_model function.
5. The model is trained using the fit method on the training data X_train and Y_train, with a specified batch size and number of epochs. The validation data is provided as well.
6. The start time and end time are recorded to calculate the training time, which is then stored in the training_times dictionary.
7. The training history, including the loss and accuracy over the epochs, is stored in the history_dict dictionary.
8. The optimizer name and the final loss and accuracy are appended to the results dictionary for each optimizer.

The code evaluates the performance of various optimizers in a Basic NN models and records relevant metrics for analysis and comparison.

```
# List of optimizers to evaluate
optimizers = ['SGD', 'Adam', 'RMSprop', 'Adagrad', 'Nadam', 'Adadelta', 'Adamax', 'AdamW', 'Adafactor']

# Dictionary to store training history
history_dict = {}

# Dictionary to store training times
training_times = {}

# Dictionary to store results
results = {'optimizer': [], 'loss': [], 'accuracy': []}

# Number of training epochs
epochs = 10
```

Figure 5.1 Code Block – Initialization of data structure

```
# Loop through each optimizer
for optimizer_name in optimizers:
    optimizer = None
    if optimizer_name == 'SGD':
        optimizer = SGD(lr=0.001)
    elif optimizer_name == 'Adam':
        optimizer = Adam(lr=0.001)
    elif optimizer_name == 'RMSprop':
        optimizer = RMSprop(lr=0.001)
    elif optimizer_name == 'Adagrad':
        optimizer = Adagrad(lr=0.001)
    elif optimizer_name == 'Nadam':
        optimizer = Nadam(lr=0.001)
    elif optimizer_name == 'Adadelta':
        optimizer = Adadelta(lr=.001)
    elif optimizer_name == 'Adamax':
        optimizer = Adamax(lr=0.001)
    elif optimizer_name == 'AdamW':
        optimizer = AdamW(lr=0.001)
    elif optimizer_name == 'Adafactor':
        optimizer = Adafactor(lr=0.001)
```

Figure 5.2 Code Block – For Loop for training models with each optimizer

```
    # create model
    model = create_model(optimizer)

    # Record start time
    start_time = time.time()

    history = model.fit(X_train, Y_train, batch_size=64, epochs=epochs, validation_data=(X_dev, Y_dev))

    # Record end time
    end_time = time.time()

    # Store the training time in the dictionary
    training_times[optimizer_name] = end_time - start_time

    # Store the training history in the dictionary
    history_dict[optimizer_name] = history.history

    # Store optimizer name and final loss/accuracy
    results['optimizer'].append(optimizer_name)
    results['loss'].append(history.history['val_loss'][-1])
    results['accuracy'].append(history.history['val_accuracy'][-1])
```

Figure 5.3 Code Block – Model creation, training and storing data

With the above code model is trained over 10 epochs with each of the optimizers available in 'optimizers' list. Relevant data is also appended and stored in 'history_dict', 'training_times' & 'results'.

Now, as defined earlier in previous chapter. Codes are used to plot and evaluate the optimizer performance. Total of 4 Criteria is decided for evaluation of optimizer performance. Following are results and evaluation of these criteria.

5.3.1.1 Criteria 1: Monitoring Training Loss and Validation Loss over epochs.

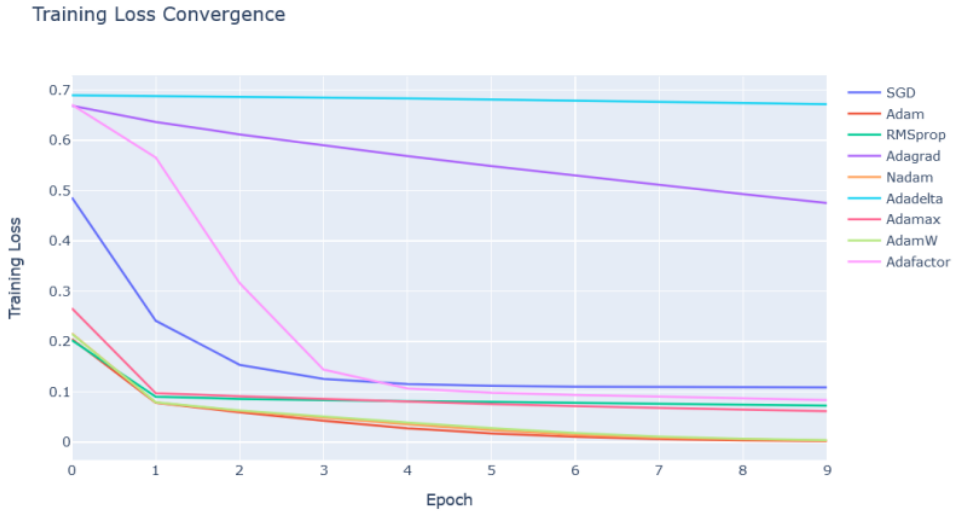


Figure 5.4 Experiment 1 - Training Loss Convergence

Table 5.1 Experiment 1 - Training Loss Data

Epoch	0	1	2	3	4	5	\
SGD	0.486509	0.241109	0.153496	0.125299	0.115436	0.111603	
Adam	0.204685	0.078049	0.059259	0.042371	0.027279	0.017095	
RMSprop	0.202553	0.090046	0.085716	0.083132	0.081233	0.079759	
Adagrad	0.668688	0.636690	0.611910	0.589610	0.568763	0.548917	
Nadam	0.216064	0.078526	0.061745	0.048085	0.035322	0.023535	
Adadelta	0.689552	0.688287	0.686809	0.685148	0.683322	0.681339	
Adamax	0.266055	0.096934	0.090855	0.085457	0.080338	0.075529	
AdamW	0.215155	0.078864	0.062810	0.050412	0.038874	0.027641	
Adafactor	0.671154	0.565893	0.315998	0.143975	0.106478	0.098100	

Epoch	6	7	8	9
SGD	0.109981	0.109242	0.108878	0.108686
Adam	0.010260	0.005814	0.003271	0.002033
RMSprop	0.078244	0.076517	0.074650	0.072195
Adagrad	0.529810	0.511274	0.493196	0.475500
Nadam	0.014475	0.008453	0.004841	0.002871
Adadelta	0.679207	0.676923	0.674485	0.671886
Adamax	0.071264	0.067393	0.064036	0.061121
AdamW	0.018028	0.010783	0.006273	0.003774
Adafactor	0.093526	0.089811	0.086477	0.083493

From the graph and Training Loss Data. It is found that:

- **Adam, AdamW & Nadam** are the best performing optimizers based on Training Loss Convergence Criteria.
- **SGD, Adafactor, RMSprop, Adamax** are performing mid-way.
- **Adadelta & Adagrad** are performing very poorly since no significant update happens over 10 epochs.

Validation Loss Convergence:

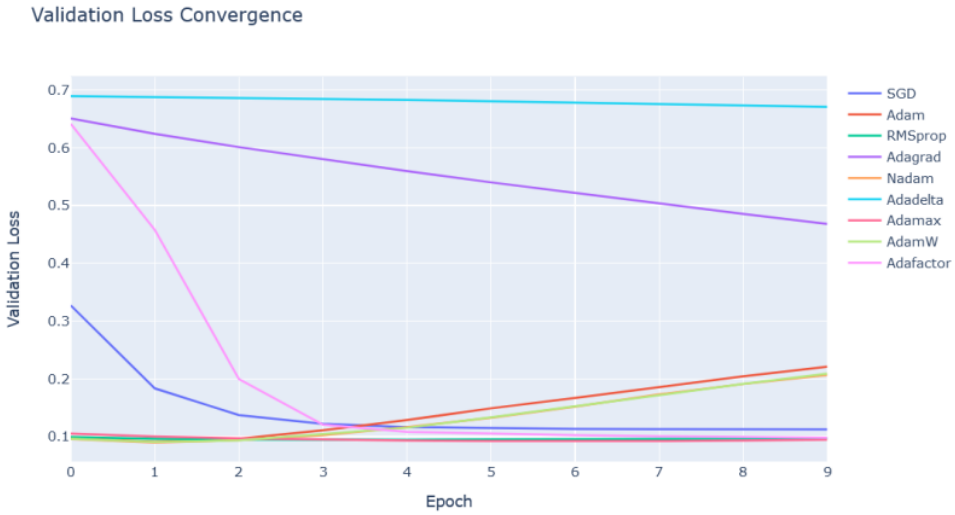


Figure 5.5 Experiment 1 – Validation Loss Convergence

Table 5.2 Experiment 1 – Validation Loss Data

Epoch	0	1	2	3	4	5
SGD	0.327054	0.183394	0.137177	0.121993	0.116441	0.114221
Adam	0.097307	0.091404	0.096063	0.111005	0.128539	0.148933
RMSprop	0.099338	0.096291	0.095064	0.094322	0.094475	0.094894
Adagrad	0.650737	0.624129	0.600974	0.579599	0.559391	0.540028
Nadam	0.095834	0.089768	0.093799	0.102419	0.116352	0.132636
Adadelta	0.689120	0.687748	0.686184	0.684447	0.682549	0.680498
Adamax	0.105242	0.100101	0.097151	0.094778	0.093150	0.092385
AdamW	0.096432	0.090666	0.093541	0.103755	0.115347	0.133169
Adafactor	0.641346	0.457755	0.199593	0.120549	0.108258	0.104634

Epoch	6	7	8	9
SGD	0.113291	0.112880	0.112681	0.112585
Adam	0.167066	0.186354	0.204339	0.221097
RMSprop	0.094903	0.095927	0.096280	0.096895
Adagrad	0.521313	0.503107	0.485316	0.467880
Nadam	0.151987	0.172914	0.191014	0.206483
Adadelta	0.678298	0.675945	0.673436	0.670766
Adamax	0.092011	0.092231	0.093154	0.094633
AdamW	0.152958	0.171428	0.191126	0.208784
Adafactor	0.102322	0.100517	0.099093	0.098169

From the graph and Validation Loss Data. It is found that:

- **Adam, AdamW & Nadam** are the reach their least loss in around 4 epochs only and afterwards loss started to increase. These 3 optimizers show nearly similar convergence behavior in this setting.

- **SGD, Adafactor** are performing mid-way. Validation Loss becomes stable around 4 epochs.
- **RMSprop, Adamax** validation loss reached stability in very early stage and kept that way till end of 10 epochs.
- **Adadelta & Adagrad** are performing very poorly in validation loss also since no significant update happens over 10 epochs.

From these evaluations, It is evident that Adam, AdamW & Nadam reach a good validation loss value around 4 epochs and also training loss reduction became slow after 4 epoch. Hence, this shows that data has started overfitting in this scenario. Hence, It can be deduced that Adam, AdamW & Nadam are efficient optimizer in text classification neural network but a careful early stopping mechanism should be utilized to save computation cost and reach a good generalized model else it will overfit the data and learn useless noise of data. Careful use of Adam, AdamW & Nadam can save energy and time significantly.

5.3.1.2 Criteria 2: Comparison of Final Validation Loss and Validation Accuracy achieved among different optimizers.

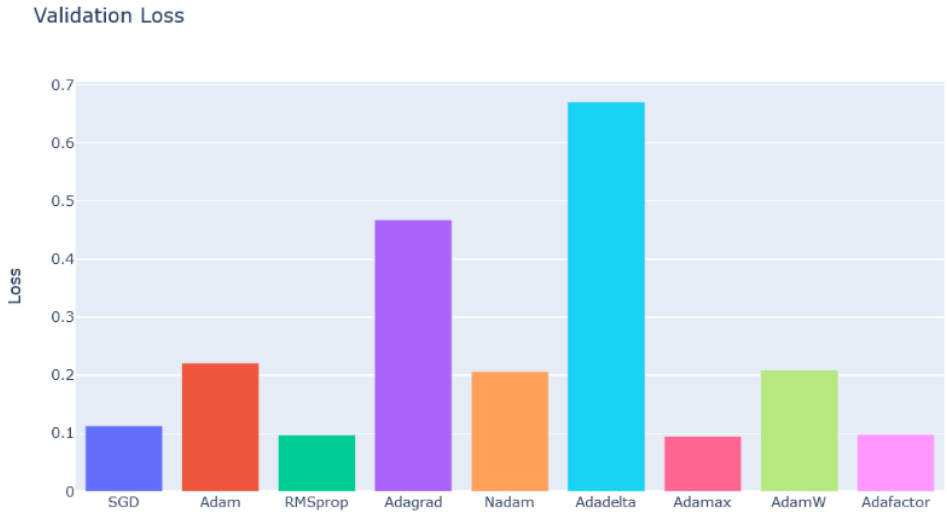


Figure 5.6 Experiment 1 – Final Validation Loss for optimizers

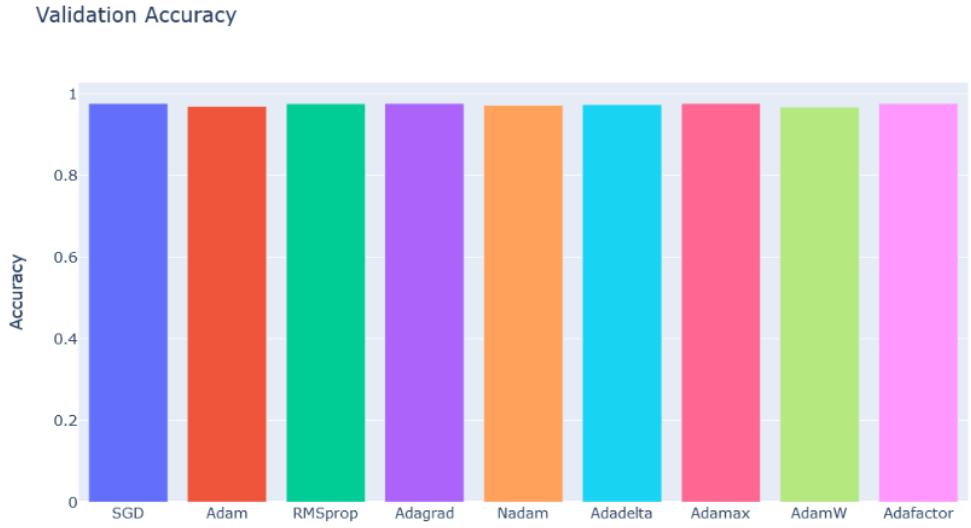


Figure 5.7 Experiment 1 – Final Validation Accuracy for optimizers

Table 5.3 Experiment 1 - Final Validation Loss & Accuracy

	Optimizer	Validation_Loss	Validation_Accuracy
0	SGD	0.112585	0.975924
1	Adam	0.221097	0.968991
2	RMSprop	0.096895	0.975154
3	Adagrad	0.467880	0.975924
4	Nadam	0.206483	0.970917
5	Adadelta	0.670766	0.972843
6	Adamax	0.094633	0.975732
7	AdamW	0.208784	0.967450
8	Adafactor	0.098169	0.975924

From the above graphs and related data. It can be seen that:

- Validation accuracy is almost the same for all optimizers after 10 epochs.
- **Adam, AdamW & Nadam** reached their least loss in around 4 epochs only and afterwards loss started to increase. Final Loss values are 0.2210, 0.2087 & 0.2064 respectively.
- **SGD, Adafactor** are performed mid-way. Final Loss is 0.1125 & 0.0981 respectively.
- **RMSprop, Adamax** validation loss reached stability in very early stage and kept that way till end of 10 epochs. Final loss are 0.0968 & 0.09463 respectively.
- **Adadelta & Adagrad** are performing very poorly in validation loss since no significant update happens over 10 epochs. Final loss compared to others remain quite high around 0.6707 & 0.4678 respectively.

5.3.1.3 Criteria 3: Training Time Comparison

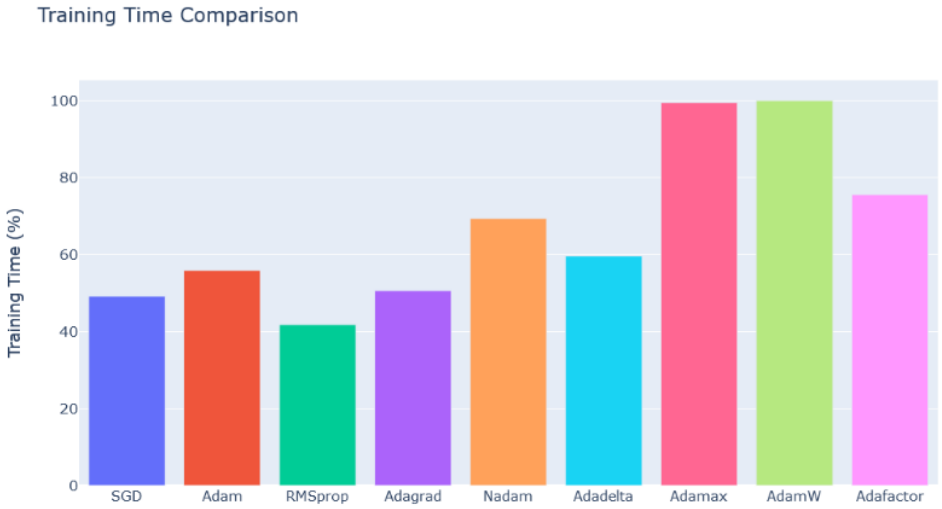


Figure 5.8 Experiment 1 - Training Time Comparison

Table 5.4 Experiment 1 - Training Time Comparison Data

	Optimizer	Time for trainings	Time_%_compared_to_max
0	SGD	10.456665	49.13
1	Adam	11.888843	55.86
2	RMSprop	8.891036	41.78
3	Adagrad	10.769455	50.60
4	Nadam	14.763378	69.37
5	Adadelta	12.687830	59.62
6	Adamax	21.160973	99.43
7	AdamW	21.282315	100.00
8	Adafactor	16.071468	75.52

From the above graph and data. It can be understood that:

- **Adam, AdamW & Nadam** reached their least loss in around 4 epochs only. Hence In case early stopping was employed. There optimizers would have taken around least timing in comparison to others.
- **SGD, RMSprop** performs very well on timing criteria.
- **Adafactor, Adamax** requires more time for training.

- **Adadelta & Adagrad** takes mid-way time for training over 10 epochs but as found earlier that there is no significant update in Validation Loss. So It will not be advisable to keep training the model when improvement is not happening.

5.3.1.4 Criteria 4: For each optimizer, Training Loss comparison with Validation Loss and Training Accuracy comparison with Validation Accuracy over the epochs.

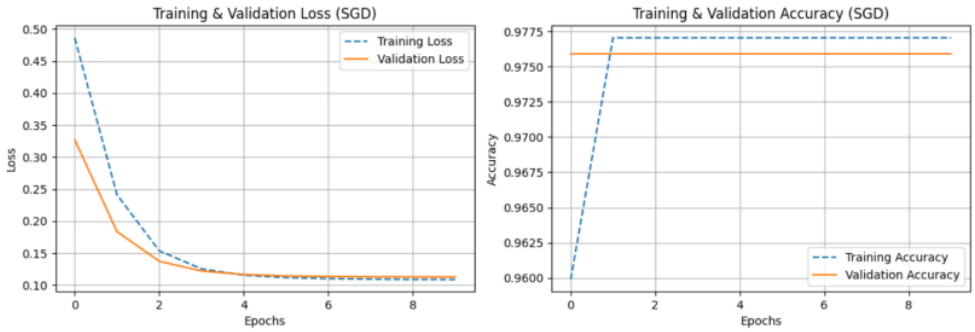


Figure 5.9 Experiment 1 – Training Curves - SGD

Optimizer SGD:

The observed trends in the training loss, validation loss, training accuracy, and validation accuracy over 10 epochs can be interpreted as follows:

- **Loss Trend:** Both the training and validation losses showed a substantial decrease in the initial five epochs, indicating effective learning. The overlap of the lines afterward suggests the model's convergence, implying that further training didn't significantly improve its performance.
- **Accuracy Trend:** The training accuracy increased initially, demonstrating successful learning. However, the subsequent stabilization suggested the model's saturation in learning. The consistent validation accuracy parallel to the stabilized training accuracy line indicated the model's consistent generalization ability without notable improvement beyond the initial epochs.

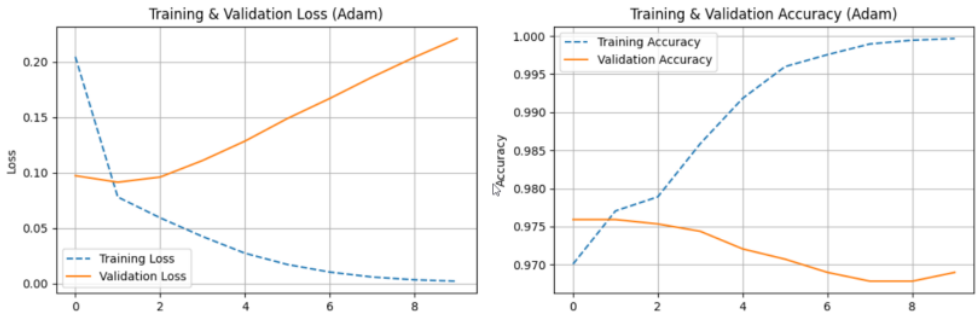


Figure 5.10 Experiment 1 – Training Curves - Adam

Optimizer Adam:

The given graphs give following information:

1. **Loss Trend:** The continual decrease in training loss throughout all epochs demonstrates effective learning. However, the increase in validation loss after the third epoch suggests that the model started to overfit, leading to reduced generalization to unseen data.
2. **Accuracy Trend:** The consistent increase in training accuracy until the final epoch indicates successful learning. Conversely, the decrease in validation accuracy after the third epoch signifies the model's reduced ability to generalize, likely due to overfitting, leading to more errors in classifying unseen data.

The observed trends indicate that the Adam optimizer initially facilitated effective learning, but as the training progressed, the model started overfitting, resulting in a decrease in its generalization ability, as evidenced by the increasing validation loss and decreasing validation accuracy. As discussed previously, early stopping seems a good solution to overcome this issue.

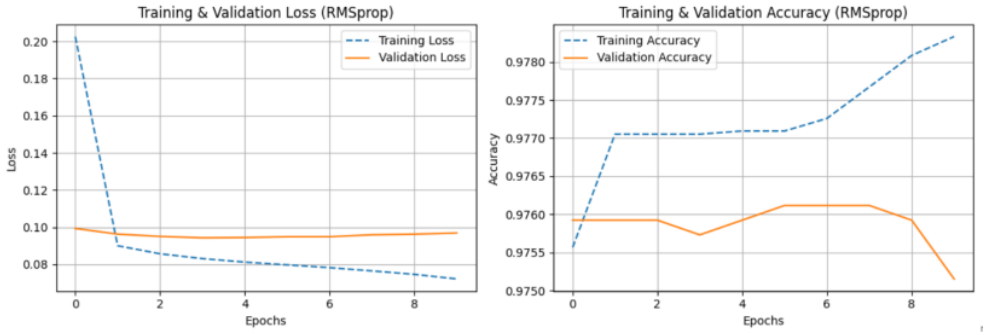


Figure 5.11 Experiment 1 – Training Curves – RMSprop

Optimizer RMSprop:

The observed trend explains the following:

1. **Loss Trends:** The significant reduction in training loss during the initial two epochs suggests effective learning. However, the subsequent decrease in the rate of reduction indicates a potential convergence to a minimum loss value, which may imply that the model's learning progress slowed down. The consistent values of the validation loss over all epochs suggest that the model's performance on the unseen validation data remained stable without significant improvements or deteriorations.
2. **Accuracy Trends:** The initial increase in training accuracy followed by a plateau suggests that the model successfully learned to classify the training data, but then it reached its learning capacity, indicating limited further improvement. The consistent values of the validation accuracy until the eighth epoch indicate that the model's generalization ability remained stable. However, the sudden plunge in validation accuracy afterward suggests a potential issue with the model's ability to generalize to new, unseen data, indicating a potential loss of performance or overfitting.

Overall, the trends indicate that the RMSprop optimizer facilitated initial effective learning, followed by a potential convergence to a minimum loss value and limited improvements in accuracy. The sudden plunge in validation accuracy in the later epochs may indicate a need for model adjustments to improve generalization and prevent overfitting.

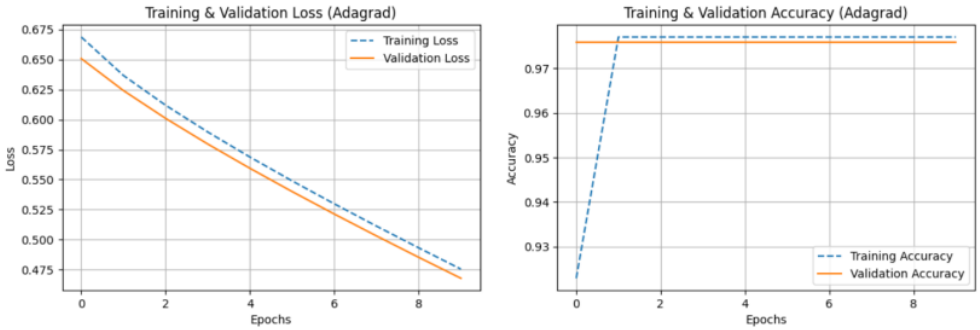


Figure 5.12 Experiment 1 – Training Curves - Adagrad

Optimizer Adagrad:

The observed trends explain the following:

1. **Loss Trends:** The consistent and simultaneous reduction of both training and validation loss along a negative 45-degree slope suggests a steady improvement in the model's performance. This consistent reduction indicates that the Adagrad optimizer effectively adjusted the learning rate for each parameter, allowing for significant progress in minimizing the loss over the epochs.
2. **Accuracy Trends:** The initial increase in training accuracy followed by a plateau suggests that the model successfully learned to classify the training data, but then it reached its learning capacity, with no significant improvements observed thereafter. The straight path of the validation accuracy, without any observable changes, implies that the model's generalization ability remained consistent without notable improvements or deteriorations throughout the training process.

Overall, the observed trends indicate that the Adagrad optimizer facilitated a steady and effective learning process, with consistent progress in minimizing the loss and maintaining a stable level of accuracy. The absence of significant fluctuations in both loss and accuracy suggests that the optimizer's adaptive learning rate mechanism effectively contributed to the model's stable and consistent performance.

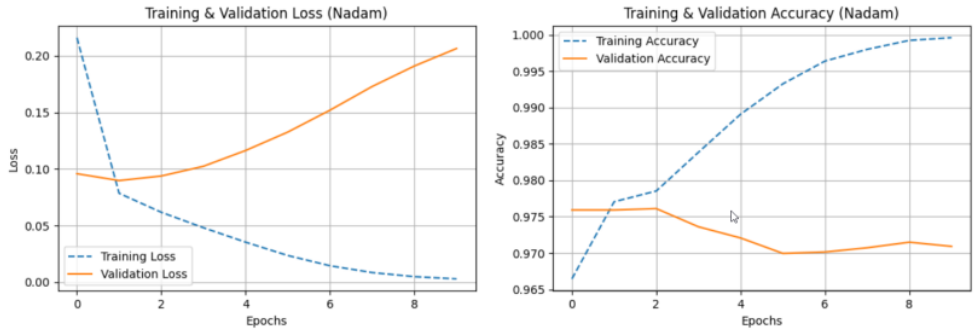


Figure 5.13 Experiment 1 – Training Curves - Nadam

Optimizer Nadam:

The observed trends for the Nadam optimizer over 10 epochs can be interpreted as follows:

1. **Loss Trends:** The initial sharp decrease in training loss followed by a continued reduction until the last epoch suggests rapid learning during the initial stages, with the subsequent decrease indicating ongoing model refinement. However, the steady increase in validation loss throughout all epochs suggests that the model's generalization to unseen data did not improve over time, potentially indicating overfitting or a failure to capture the underlying patterns in the validation data.
2. **Accuracy Trends:** The continuous and steady increase in training accuracy until the final epoch indicates the model's consistent learning and successful classification of the training data. The stability in the validation accuracy during the initial epochs, followed by a decrease and subsequent stabilization at a lower accuracy level, suggests that the model might have initially been overfitting, leading to a drop in generalization ability, which stabilized at a lower performance level.

Overall, the observed trends suggest that the Nadam optimizer facilitated rapid initial learning, as evidenced by the plunging training loss and increasing training accuracy. However, the increasing validation loss and the corresponding instability and reduction in validation accuracy indicate potential issues with the model's generalization to unseen data, emphasizing the need for further analysis and potential adjustments, such as regularization or model complexity modifications, to enhance the model's ability to generalize effectively.

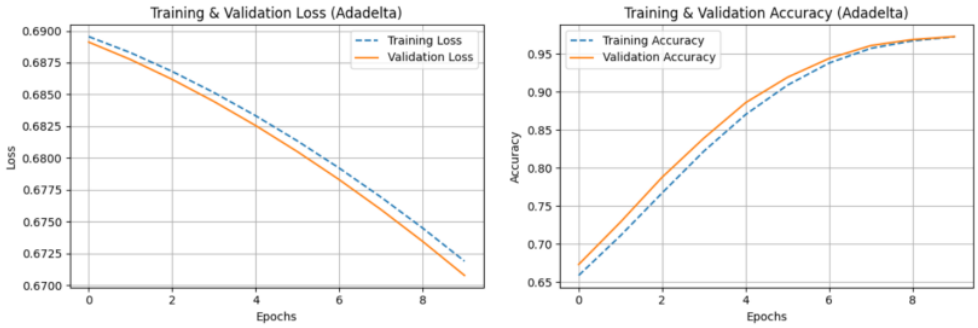


Figure 5.14 Experiment 1 – Training Curves - Adadelta

Optimizer Adadelta:

The observed trends for the Adadelta optimizer over 10 epochs can be interpreted as follows:

1. **Loss Trends:** The consistent and simultaneous reduction of both training and validation loss along a negative 45-degree slope suggests a steady and effective improvement in the model's performance. This consistent reduction indicates that the Adadelta optimizer successfully adjusted the learning rate and effectively minimized the loss over the epochs. The parallel paths of the training and validation loss suggest that the model's generalization to unseen data remained consistent throughout the training process.
2. **Accuracy Trends:** The consistent and steady increase in both training and validation accuracy at a 45-degree angle indicates continuous learning and improved classification performance of the model over the epochs. The convergence of both the training and validation accuracy lines towards the end suggests that the model achieved a stable and consistent level of accuracy, indicating that the model learned to generalize effectively without overfitting to the training data.

Overall, the observed trends suggest that the Adadelta optimizer facilitated a stable and effective learning process, resulting in consistent progress in minimizing the loss and achieving a stable and consistent level of accuracy. The observed patterns indicate that the optimizer's adaptive learning rate mechanism effectively contributed to the model's stable and consistent performance, enabling successful learning and generalization.

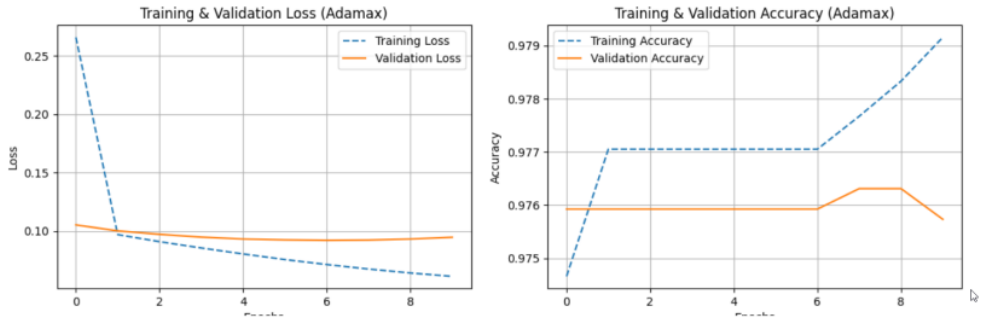


Figure 5.15 Experiment 1 – Training Curves - Adamax

Optimizer Adamax:

The observed trends for the Adamax optimizer over 10 epochs can be interpreted as follows:

1. **Loss Trends:** The significant initial plunge in the training loss followed by a continued reduction until the end of training signifies rapid and effective learning during the initial epochs, with ongoing refinement of the model's performance. On the other hand, the relatively stable validation loss, maintaining a consistent value throughout the training process, suggests that the model's generalization to unseen data remained consistent without notable improvements or deteriorations.
2. **Accuracy Trends:** The initial increase in training accuracy, followed by a plateau and subsequent increase, suggests the model successfully learned to classify the training data, reached a point of learning saturation, and then further improved its performance. The stability in validation accuracy until the seventh epoch, followed by an increase and subsequent drop, suggests that the model's generalization ability might have initially remained stable, experienced some improvement, and then encountered potential overfitting, leading to reduced performance in generalization.

Overall, the observed trends suggest that the Adamax optimizer facilitated rapid learning during the initial epochs, resulting in effective reduction of the training loss and improvement in training accuracy. However, the stable validation loss and accuracy indicate a potential need for model adjustments, such as regularization or complexity modifications, to enhance the model's generalization ability and prevent overfitting to the training data.

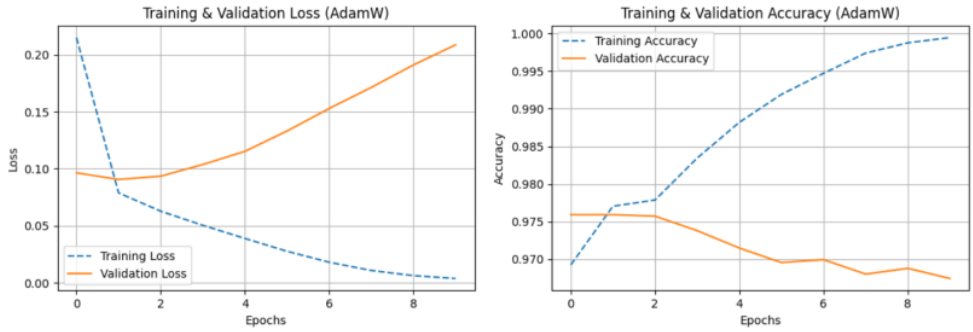


Figure 5.16 Experiment 1 – Training Curves - AdamW

Optimizer AdamW:

The observed graphs for the AdamW optimizer over 10 epochs shows following:

1. **Loss Trends:** The significant initial plunge in the training loss, followed by a continued and substantial reduction until the last epoch, suggests rapid and effective learning throughout the training process. In contrast, the initial decrease followed by a subsequent increase in the validation loss implies that the model's generalization to unseen data deteriorated over time, potentially due to overfitting or a failure to capture the underlying patterns in the validation data.
2. **Accuracy Trends:** The consistent increase in training accuracy throughout all epochs indicates the model successfully learned to classify the training data, without reaching a saturation point, thereby continuing to improve until the final epoch. The gradual decrease in validation accuracy over the training period suggests the model's declining ability to generalize to unseen data, potentially due to overfitting or a mismatch between the model's complexity and the data's underlying patterns.

Overall, the observed trends suggest that the AdamW optimizer facilitated rapid and effective learning, resulting in a substantial reduction of the training loss and consistent improvement in training accuracy. However, the increasing validation loss and the corresponding decrease in validation accuracy indicate potential issues with the model's generalization to unseen data, emphasizing the need for further analysis and potential adjustments, such as regularization or complexity modifications, to improve the model's generalization capability and prevent overfitting.

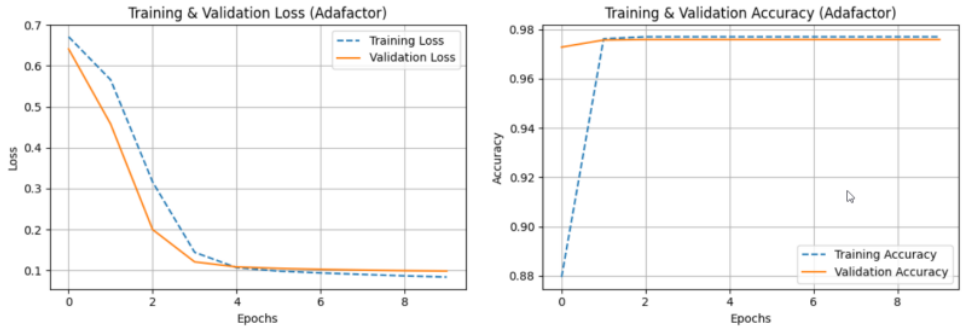


Figure 5.17 Experiment 1 – Training Curves - Adafactor

Optimizer Adafactor:

The observed graphs for the Adafactor optimizer over 10 epochs describe following:

1. **Loss Trends:** The rapid reduction in both training and validation loss during the initial five epochs indicates effective learning and significant improvements in the model's performance. The subsequent stabilization of both losses until the end of the training process suggests that the model reached a point of convergence, where further training did not significantly improve its performance. This convergence indicates that the model effectively learned the underlying patterns in the data and stabilized its predictive capability.
2. **Accuracy Trends:** The substantial increase in training accuracy during the initial epochs reflects the model's successful learning and improved ability to classify the training data. The consistent and stable validation accuracy, which matched the training accuracy after the initial training phase, suggests that the model demonstrated robust generalization to the unseen validation data, indicating its capacity to perform consistently well on previously unseen samples.

Overall, the observed graph suggests that the Adafactor optimizer facilitated rapid and effective learning during the initial epochs, resulting in significant improvements in both the loss reduction and accuracy increase. The subsequent stabilization of both the loss and accuracy indicates that the model effectively learned and generalized to the data, reaching a consistent and stable level of performance that was maintained throughout the rest of the training process.

5.3.2 Experiment No. 2 – Recurrent Neural Network with LSTM Units

In the second experiment, A Recurrent Neural Network is implemented using LSTM Units. LSTM is well known for its ability to model and process sequential data. Hence, the optimizer understudy are also evaluated for their working with these LSTM Units. Their performance in training RNN networks with LSTM is evaluated and the results are presented for predefined evaluation criteria. Results discussion follows:

5.3.2.1 Criteria 1: Monitoring Training Loss and Validation Loss over epochs.

Training Loss Convergence:

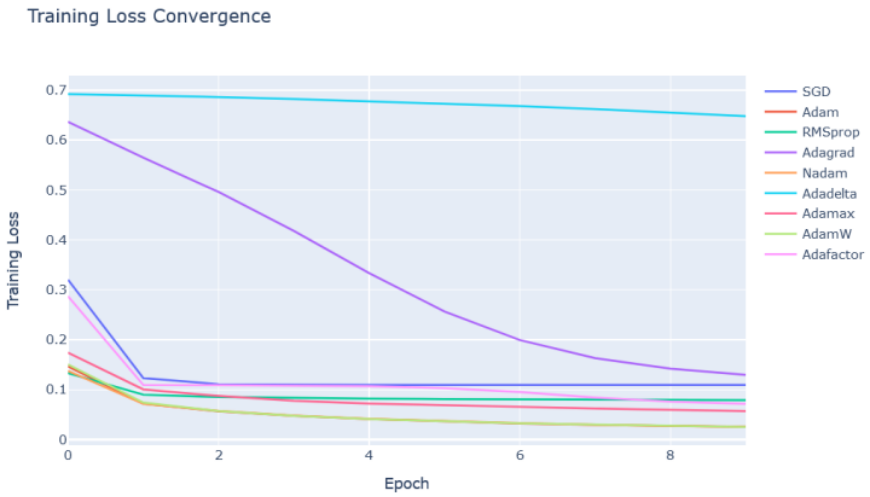


Figure 5.18 Experiment 2 - Training Loss Convergence

Table 5.5 Experiment 2 - Training Loss Data

Epoch	0	1	2	3	4	5
SGD	0.320059	0.122964	0.110820	0.109504	0.109325	0.109288
Adam	0.146686	0.071949	0.056345	0.047299	0.040981	0.036577
RMSprop	0.132932	0.089779	0.085124	0.083125	0.081954	0.081014
Adagrad	0.636440	0.564572	0.495889	0.417890	0.333446	0.256326
Nadam	0.137453	0.071311	0.057460	0.047746	0.041379	0.036413
Adadelta	0.691950	0.689440	0.686039	0.682153	0.677877	0.673147
Adamax	0.173929	0.100412	0.087272	0.077360	0.072163	0.068523
AdamW	0.150367	0.073958	0.056959	0.047561	0.041134	0.036644
Adafactor	0.287177	0.109239	0.108886	0.108207	0.106629	0.102779

Epoch	6	7	8	9
SGD	0.109282	0.109278	0.109275	0.109273
Adam	0.032206	0.029485	0.027079	0.025532
RMSprop	0.080299	0.079648	0.079290	0.078901
Adagrad	0.199277	0.163169	0.141845	0.129425
Nadam	0.032704	0.029488	0.027595	0.025300
Adadelta	0.667878	0.661983	0.655350	0.647831
Adamax	0.065346	0.062256	0.059414	0.056778
AdamW	0.032389	0.029855	0.027305	0.025254
Adafactor	0.095092	0.083984	0.075897	0.071596

From the graph and Training Loss Data. It is found that:

- **Adam, AdamW & Nadam** are performing very nice in this case. Training loss decrease fast and then stabilizes over the training schedule.
- **SGD, Adafactor, RMSprop, Adamax** are performing mid-way since these start with higher initial loss, which stabilizes soon after but still the loss values are higher than Adam, AdamW & Nadam.
- **Adadelta & Adagrad** perform poorly since they are taking quite long to understand the data and learn the pattern. It is clearly evident with Training loss curve.

Validation Loss Convergence:

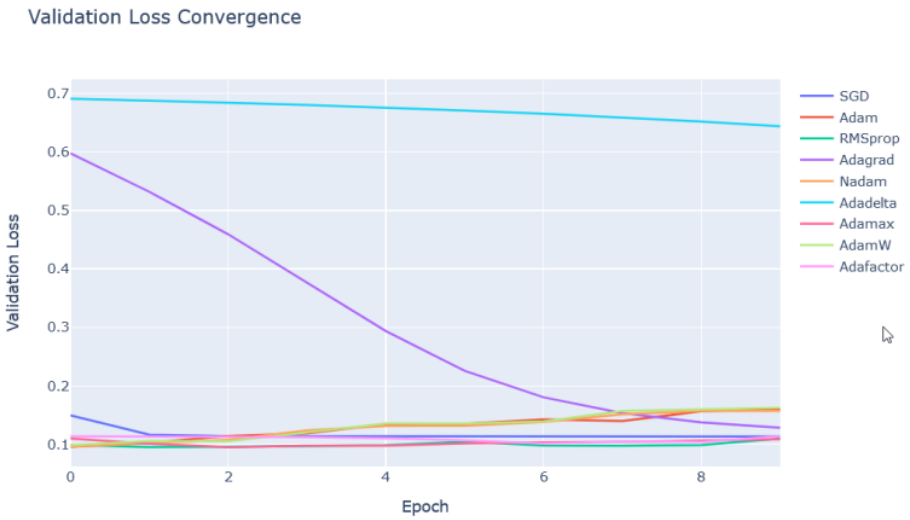


Figure 5.19 Experiment 2 - Validation Loss Convergence

Table 5.6 Experiment 2 - Validation Loss Data

Epoch	0	1	2	3	4	5
SGD	0.149513	0.116507	0.113775	0.113494	0.113476	0.113484
Adam	0.095841	0.102686	0.114277	0.118603	0.134991	0.135435
RMSprop	0.098888	0.095403	0.095642	0.096949	0.098377	0.104460
Adagrad	0.597612	0.531711	0.459144	0.376677	0.293499	0.225698
Nadam	0.095685	0.101653	0.107899	0.123742	0.131554	0.131676
Adadelta	0.690884	0.687857	0.684171	0.680107	0.675628	0.670654
Adamax	0.109838	0.101228	0.095062	0.097516	0.097568	0.101402
AdamW	0.098467	0.105587	0.105152	0.120493	0.135634	0.135617
Adafactor	0.113682	0.113127	0.112779	0.112401	0.110349	0.106722

Epoch	6	7	8	9
SGD	0.113483	0.113475	0.113476	0.113480
Adam	0.142549	0.139739	0.156637	0.160422
RMSprop	0.097928	0.097644	0.098982	0.109969
Adagrad	0.180346	0.153164	0.137416	0.128237
Nadam	0.138162	0.151415	0.157083	0.157291
Adadelta	0.665100	0.658873	0.651839	0.643837
Adamax	0.103370	0.104044	0.106433	0.108811
AdamW	0.139172	0.157303	0.160520	0.162361
Adafactor	0.101335	0.104724	0.104247	0.113922

From the graph and Validation Loss Data. It is found that:

- **Adamax, Adafactor, RMSprop** performed well on this criterion. Validation loss is less and stay stable during the training.
- **SGD** also performed validation loss decrease initially and then stabilize.
- **Adam, AdamW & Nadam** validation loss increased showing that overfitting is happening, model is losing its generalizing ability. Early stopping is recommended in this case to improve the model and also to save precious compute power.
- **Adadelta & Adagrad** are performing very poorly in validation loss criteria since no significant learning happened.

5.3.2.2 Criteria 2: Comparison of Final Validation Loss and Validation Accuracy achieved among different optimizers.

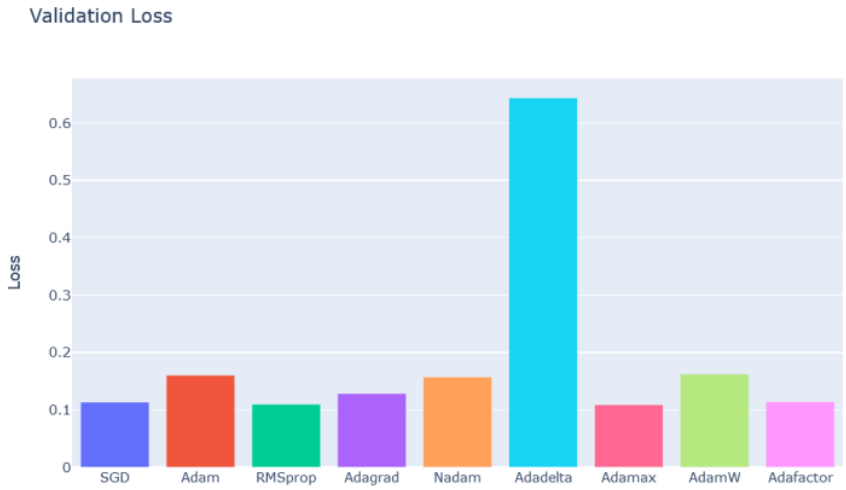


Figure 5.20 Experiment 2 - Final Validation Loss comparison

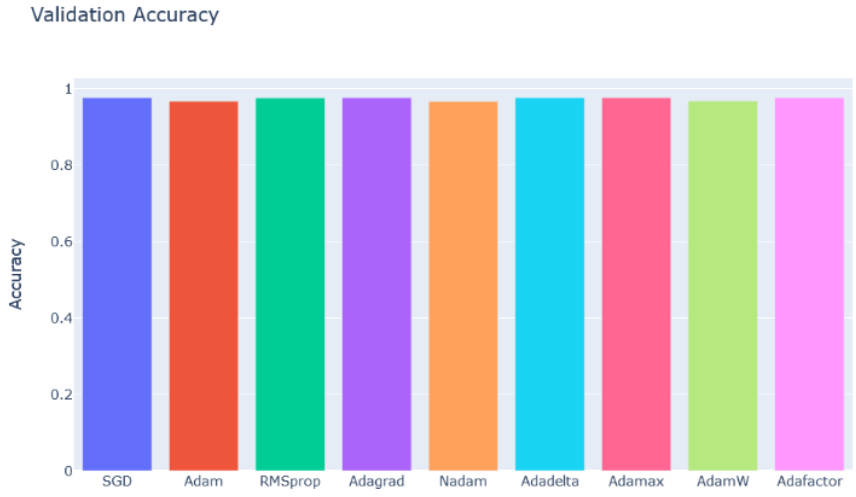


Figure 5.21 Experiment 2 - Final Validation Accuracy Comparison

Table 5.7 Experiment 2 - Final Validation Loss & Accuracy Data

	Optimizer	Validation_Loss	Validation_Accuracy
0	SGD	0.113480	0.975924
1	Adam	0.160422	0.967257
2	RMSprop	0.109969	0.975347
3	Adagrad	0.128237	0.975924
4	Nadam	0.157291	0.966294
5	Adadelta	0.643837	0.975924
6	Adamax	0.108811	0.976117
7	AdamW	0.162361	0.967835
8	Adafactor	0.113922	0.975924

From the above graphs and related data. It can be seen that:

- Validation accuracy is almost the same for all optimizers after 10 epochs.
- **Adam, AdamW & Nadam** reached their least losses in starting epochs only and afterwards loss started to increase. Final Loss values are 0.160, 0.162 & 0.157 respectively.
- **SGD** performed quite well. Final Loss is 0.1134.
- **RMSprop, Adamax, Adafactor** validation loss reached stability in very early stage and kept that way till end of 10 epochs. Some fluctuation occurred but contained to very small level. Final loss values are 0.109, 0.108 & 0.113 respectively.
- **Adadelta & Adagrad** are performing very poorly in validation loss. Adadelta had no significant training in 10 epochs and final loss value was 0.643. Adagrad though learned but took very long to update parameters and final loss was 0.128.

5.3.2.3 Criteria 3: Training Time Comparison

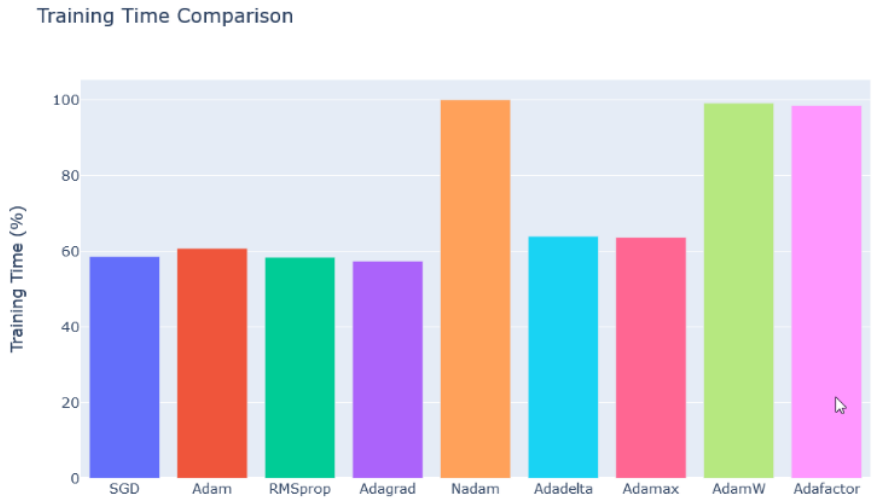


Figure 5.22 Experiment 2 - Training Time Comparison

Table 5.8 Experiment 2 - Training Time Data

	Optimizer	Time for trainings	Time_%_compared_to_max
0	SGD	51.103466	58.58
1	Adam	52.988075	60.74
2	RMSprop	50.922536	58.38
3	Adagrad	50.041588	57.37
4	Nadam	87.230623	100.00
5	Adadelta	55.779945	63.95
6	Adamax	55.469574	63.59
7	AdamW	86.485935	99.15
8	Adafactor	85.935569	98.52

From the given graph and data. It can be understood that:

- **Adam, AdamW & Nadam** reached their least losses for both training and validation sets in starting epochs only. Training time would have been significantly less, if early stopping criteria were used in this case.
- **SGD, RMSprop, Adagrad, Adadelta, Adamax** takes less time for training and performs well on timing criteria.
- **Adafactor** also got trained in initial epochs and validation loss decreased till 7 epochs and then started increasing. It shows here also early stopping can be employed.

Training time is very important criteria, it should be evaluated in conjunction with Models training loss and validation loss which give and indication of amount of training required or In other sense it shows whether there is any benefit in training model further. Since training further only overfits the model though training accuracy increase but validation accuracy reduce since model start overfitting the data. Here, Adam, AdamW, Nadam & Adafactor seems good performing optimizers.

5.3.2.4 Criteria 4: For each optimizer, Training Loss comparison with Validation Loss and Training Accuracy comparison with Validation Accuracy over the epochs.

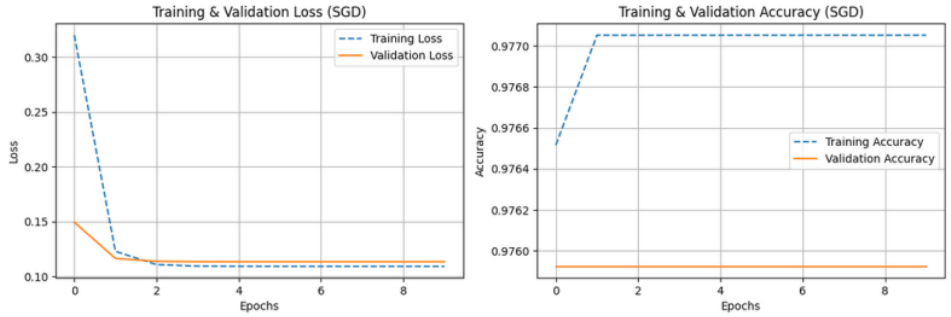


Figure 5.23 Experiment 2 - Training Curves - SGD

Optimizer SGD:

The observed trends for the Stochastic Gradient Descent (SGD) optimizer indicate the following patterns:

1. **Loss Trend:** The rapid decline in training loss during the initial two epochs followed by a stabilization suggests an effective initial learning phase, with subsequent epochs leading to minimal changes in the model's loss. Similarly, the validation loss demonstrates a reduction in the initial epochs followed by a stabilization, indicating the model's consistent performance on unseen data after the initial learning phase.
2. **Accuracy Trend:** The substantial increase in training accuracy during the initial epochs, followed by a plateau, suggests the model's rapid learning and subsequent saturation in its ability to improve on the training data. Concurrently, the consistent validation accuracy from the beginning until the end of the training process indicates that the model's performance on unseen data remains steady and does not significantly improve despite continued training.

These trends collectively suggest that the SGD optimizer enabled the model to quickly learn the training data, leading to a significant reduction in both training and validation loss. However, the subsequent stabilization of the loss and accuracy metrics indicates that the model's learning and generalization abilities reached a saturation point, with no significant improvements observed beyond the initial epochs.

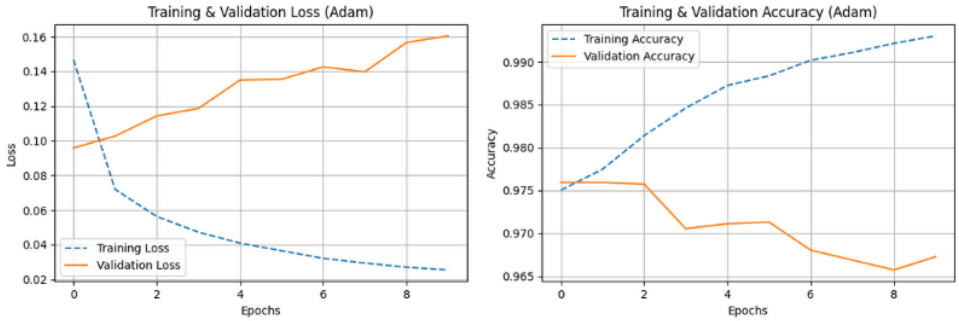


Figure 5.24 Experiment 2 - Training Curves - Adam

Optimizer Adam:

The observed trends for the Adam optimizer indicate the following patterns:

1. **Loss Trend:** The initial plunge followed by a continuous decrease in the training loss suggests effective learning and improved performance on the training data over time. In contrast, the fluctuating increase in the validation loss signifies challenges in generalizing to unseen data, indicating potential overfitting or difficulties in capturing complex patterns.
2. **Accuracy Trend:** The continual increase in training accuracy until the final epoch reflects the model's improved ability to correctly classify the training data. However, the intermittent decrease in validation accuracy, marked by step-like changes, suggests difficulties in generalizing to new data, potentially caused by overfitting or a lack of robustness in the model's learned representations.

These trends collectively suggest that the Adam optimizer facilitated effective learning on the training data, leading to a consistent decrease in the training loss and an increase in training accuracy. However, the fluctuating nature of the validation loss and the step-wise decrease in validation accuracy indicate challenges in the model's generalization ability, which may require regularization techniques or adjustments to the model's complexity to improve its performance on unseen data.

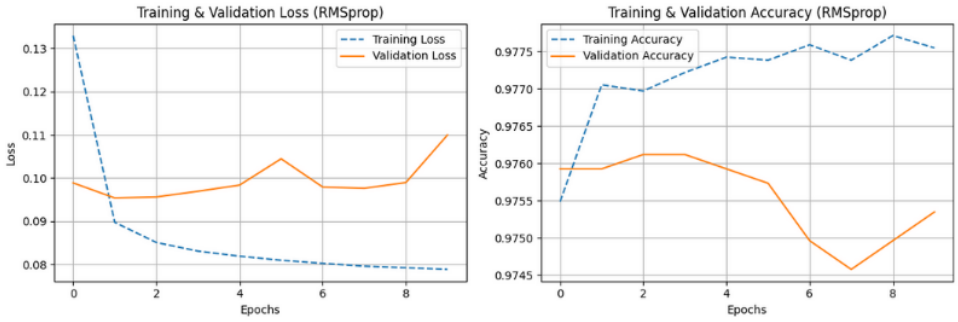


Figure 5.25 Experiment 2 - Training Curves - RMSprop

Optimizer RMSprop:

The observed trends for the RMSprop optimizer reveal the following insights:

1. **Loss Trend:** The significant and gradual reduction in the training loss throughout the epochs suggests effective learning and improved performance on the training data over time. However, the fluctuations and the increasing trend in the validation loss indicate challenges in generalizing to unseen data, potentially reflecting the model's struggle to capture the underlying patterns.
2. **Accuracy Trend:** The intermittent and fluctuating increase in training accuracy implies the model's ability to correctly classify the training data with variations across the epochs. In contrast, the decreasing trend in the validation accuracy, followed by a sudden increase, suggests that the model initially struggled to generalize to new data, but improved its performance after the 8th epoch.

These trends collectively suggest that the RMSprop optimizer facilitated effective learning on the training data, leading to a gradual decrease in the training loss and an increase in training accuracy. However, the fluctuations in the validation loss and the initial decrease in validation accuracy indicate challenges in the model's generalization ability, potentially due to fluctuations in the data or model complexity. Fine-tuning the model's architecture, adjusting hyperparameters, or introducing regularization techniques may help improve its overall performance on unseen data.

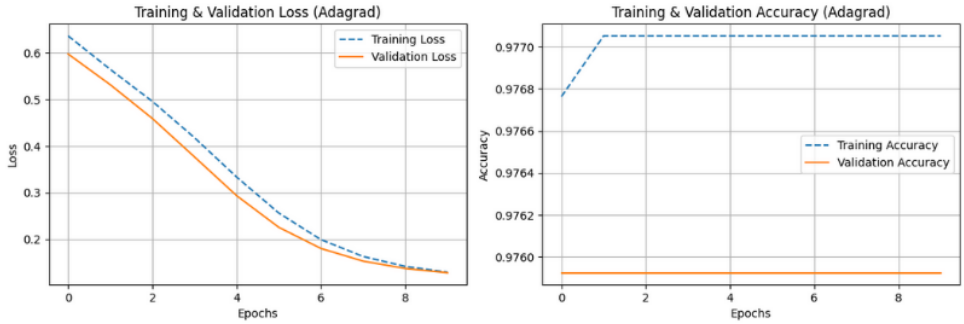


Figure 5.26 Experiment 2 - Training Curves - Adagrad

Optimizer Adagrad:

The observed trends for the Adagrad optimizer illustrate the following patterns:

1. **Loss Trend:** The concurrent decrease in both training and validation loss, forming a negative 45-degree slope, indicates an effective learning process. This demonstrates that the model successfully minimized errors and improved its performance on both the training and validation datasets over the training period.
2. **Accuracy Trend:** The initial improvement in training accuracy followed by a consistent level suggests that the model correctly classified the training data and maintained this performance throughout the training process. Additionally, the constant validation accuracy throughout the training duration signifies that the model's generalization ability remained stable and consistent without significant improvements.

These trends collectively indicate that the Adagrad optimizer facilitated effective learning and optimization, leading to a gradual reduction in both training and validation loss. The stability of the accuracy metrics implies that the model maintained a consistent and reliable performance on both the training and validation datasets. The balanced nature of these trends suggests that the Adagrad optimizer was successful in enabling the model to learn and generalize effectively without significant fluctuations or overfitting issues.

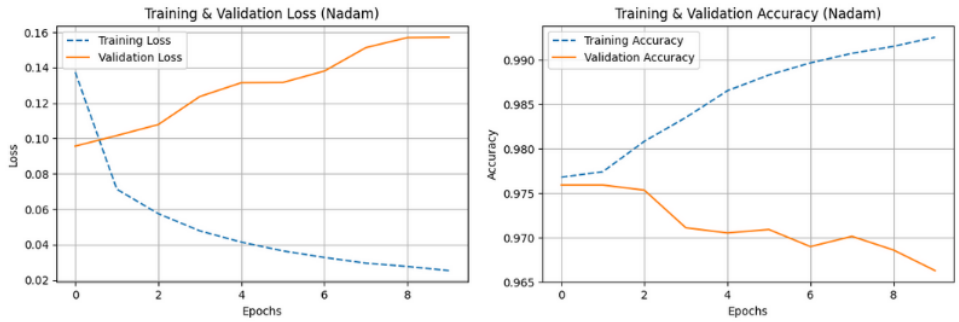


Figure 5.27 Experiment 2 - Training Curves - Nadam

Optimizer Nadam:

The observed trends for the Nadam optimizer demonstrate the following patterns:

1. **Loss Trend:** The consistent decrease in training loss throughout the epochs indicates the model's effective learning and improved performance on the training data over time. Conversely, the consistent increase in the validation loss suggests challenges in generalizing to unseen data, potentially indicating the model's struggles to capture complex patterns beyond the training dataset.
2. **Accuracy Trend:** The continual increase in training accuracy signifies the model's improved ability to correctly classify the training data and learn the underlying patterns within the dataset. On the other hand, the consistent decrease in validation accuracy, marked by slow, steady steps, implies challenges in generalization to new data, potentially reflecting the model's limitations in capturing complex patterns beyond the training dataset.

These trends collectively suggest that the Nadam optimizer facilitated effective learning on the training data, leading to a consistent decrease in the training loss and an increase in the training accuracy. However, the increasing validation loss and the decreasing validation accuracy indicate challenges in the model's generalization ability, which may require regularization techniques or adjustments to the model's complexity to improve its performance on unseen data. Further fine-tuning of the model's architecture and hyperparameters might be necessary to enhance its overall performance and generalization capabilities.

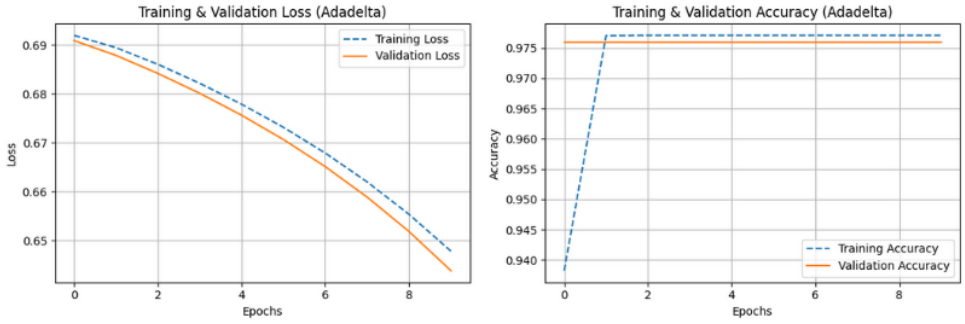


Figure 5.28 Experiment 2 - Training Curves - Adadelta

Optimizer Adadelta:

The trends observed for the Adadelta optimizer can be explained as follows:

1. **Loss Trend:** The smooth downward curve observed for both the training and validation loss signifies effective learning and optimization. This indicates that the model successfully minimized errors and improved its performance on both the training and validation datasets consistently throughout the training period.
2. **Accuracy Trend:** The continual increase followed by a plateau in training accuracy suggests that the model effectively learned the underlying patterns within the training data, achieving a consistent and stable performance. Similarly, the maintenance of similar values in the validation accuracy throughout the training process indicates that the model's generalization capability remained steady and consistent without significant variations.

These trends collectively suggest that the Adadelta optimizer facilitated effective learning and optimization, resulting in a gradual reduction in both training and validation loss. The consistency in the accuracy metrics indicates that the model was able to maintain a stable and reliable performance on both the training and validation datasets. The smooth and consistent nature of these trends suggests that the Adadelta optimizer was successful in enabling the model to learn and generalize effectively without significant fluctuations or overfitting issues.

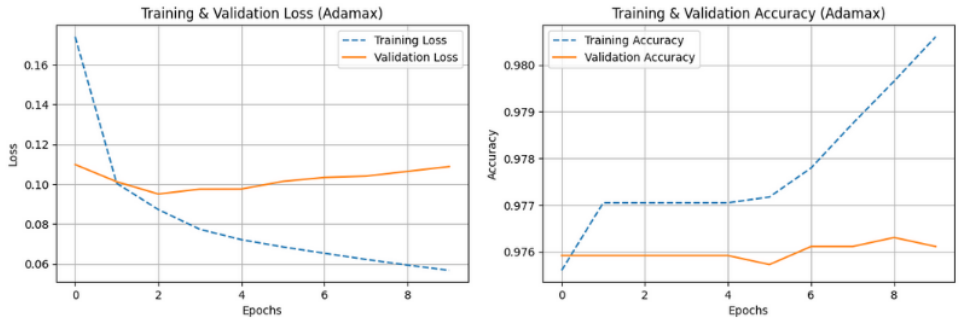


Figure 5.29 Experiment 2 - Training Curves - Adamax

Optimizer Adamax:

The observed trends for the Adamax optimizer can be explained as follows:

1. **Loss Trend:** The rapid and continual decrease in training loss suggests effective learning and optimization, indicating that the model successfully minimized errors and improved its performance on the training data over the entire training period. The slight increase in validation loss towards the end of the training process may indicate a slight overfitting tendency or challenges in capturing the complex patterns beyond the training dataset.
2. **Accuracy Trend:** The initial increase followed by a plateau and then a consistent rise in training accuracy suggests that the model effectively learned the underlying patterns within the training data, achieving a stable and then improved performance. Similarly, the maintenance of similar values in the validation accuracy with slight fluctuations throughout the training process indicates that the model's generalization capability remained steady and consistent without significant variations.

These trends collectively suggest that the Adamax optimizer facilitated effective learning and optimization, resulting in a continuous reduction in both training and validation loss. The consistent improvement in the accuracy metrics indicates that the model was able to maintain a stable and reliable performance on both the training and validation datasets.

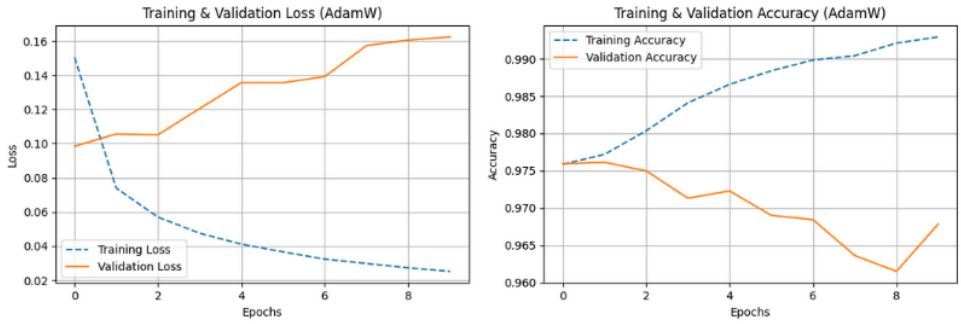


Figure 5.30 Experiment 2 - Training Curves - AdamW

Optimizer AdamW:

The observed trends for the AdamW optimizer can be elucidated as follows:

1. **Loss Trend:** The consistent decrease in training loss suggests effective learning and optimization, indicating that the model successfully minimized errors and improved its performance on the training data throughout the training period. In contrast, the increasing trend in the validation loss suggests challenges in generalizing to unseen data, potentially reflecting the model's struggle to capture complex patterns beyond the training dataset.
2. **Accuracy Trend:** The continual increase in training accuracy implies that the model effectively learned the underlying patterns within the training data, achieving a stable and improved performance. Conversely, the decreasing trend in validation accuracy indicates challenges in generalization to new data, potentially reflecting the model's limitations in capturing complex patterns beyond the training dataset.

These trends collectively suggest that the AdamW optimizer facilitated effective learning and optimization on the training data, leading to a consistent decrease in the training loss and an increase in the training accuracy. However, the increasing validation loss and the decreasing validation accuracy imply challenges in the model's generalization ability, which may require further investigation or adjustments to the model's architecture or hyperparameters.

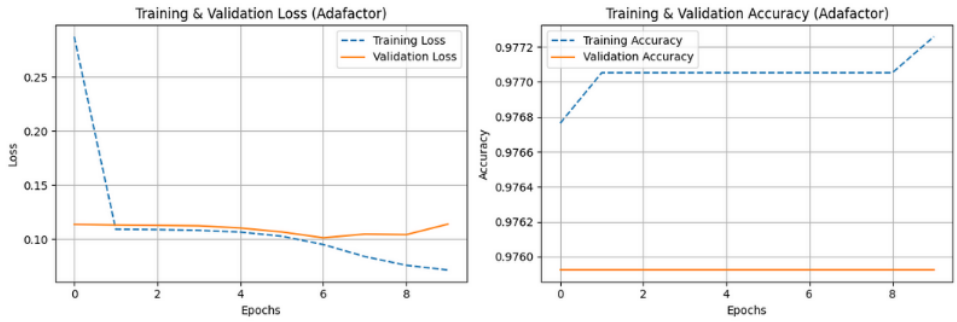


Figure 5.31 Experiment 2 - Training Curves - Adafactor

Optimizer Adafactor:

The observed trends for the Adafactor optimizer can be interpreted as follows:

1. **Loss Trend:** The initial plunge in the training loss, followed by a slow decrease over the training course, indicates effective learning and optimization, demonstrating the model's successful minimization of errors and improved performance on the training data. However, the subsequent slight increase in validation loss towards the end suggests a potential challenge in generalizing to unseen data, indicating the model's difficulty in capturing complex patterns beyond the training dataset.
2. **Accuracy Trend:** The initial increase in training accuracy, followed by a stabilization period and then a subsequent increase, suggests that the model effectively learned the underlying patterns within the training data, achieving a stable and improved performance. The maintenance of similar values in the validation accuracy throughout the training process indicates that the model's generalization capability remained consistent without significant variations.

These trends collectively suggest that the Adafactor optimizer facilitated effective learning and optimization, resulting in a gradual decrease in the training loss and an increase in the training accuracy. The stability in the validation accuracy indicates that the model maintained a consistent and reliable performance on unseen data. However, the slight increase in the validation loss towards the end of the training period may necessitate further analysis and fine-tuning of the model's architecture or hyperparameters to enhance its performance on unseen data.

5.3.3 Experiment No. 3 - Recurrent Neural Network with GRU Units

In the third experiment, A Recurrent Neural Network is implement using GRU Units. GRU is well known for its ability to model and process sequential data and are computationally efficient that LSTM. Also, they are good for short-term dependency modelling and better memory management GRU has simplified memory architecture compared to LSTM. Optimizers performance in training RNN networks with GRU is evaluated and the results are presented for predefined evaluation criteria. Results discussion follows:

5.3.3.1 Criteria 1: Monitoring Training Loss and Validation Loss over epochs.

Training Loss Convergence:

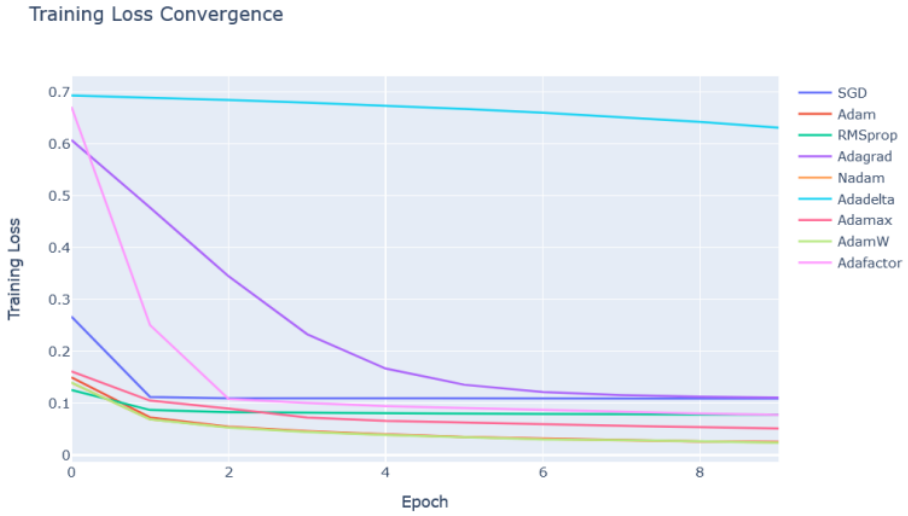


Figure 5.32 Experiment 3 - Training Loss Convergence

Table 5.9 Experiment 3 - Training Loss Data

Epoch	0	1	2	3	4	5	6	7	8	9
SGD	0.267123	0.112292	0.109383	0.109277	0.109265	0.109244	0.109246	0.109235	0.109242	0.109235
Adam	0.150068	0.072677	0.054696	0.046093	0.040093	0.035199	0.032119	0.029105	0.026574	0.025551
RMSprop	0.125478	0.086948	0.082735	0.081590	0.080810	0.079656	0.079280	0.079070	0.078258	0.077897
Adagrad	0.607208	0.477195	0.344672	0.233035	0.166772	0.135615	0.121807	0.115535	0.112526	0.111008
Nadam	0.139675	0.069183	0.053728	0.045378	0.039261	0.034636	0.031088	0.028416	0.026411	0.024516
Adadelta	0.692941	0.689212	0.684506	0.679278	0.673501	0.667061	0.659810	0.651559	0.642058	0.630985
Adamax	0.161368	0.105059	0.089787	0.072676	0.066039	0.062476	0.059156	0.056406	0.053942	0.051420
AdamW	0.139956	0.068876	0.053241	0.044649	0.039081	0.034815	0.030141	0.029141	0.026085	0.023890
Adafactor	0.671375	0.250605	0.108405	0.100319	0.094678	0.090473	0.086773	0.083490	0.080386	0.077543

From the graph and Training Loss Data. It is found that:

- **Adadelta & Adagrad** optimizer are taking quite a long time to train the model. No significant improvement is visible in Training loss for Adadelta optimizer till last. Adagrad also took a very long time compared to peer optimizers.
- **Adafactor** performance is mid-way compared to other optimizers. Training loss plunges in initially and then stabilizes.
- **SGD, Adamax, RMSprop** started with low training loss and tuned the model easily to reach the stable state.
- **Adam, AdamW, Nadam** are the best performers. Training loss graphs are nearly overlapping for these algorithms. Especially Nadam & AdamW are overlapping each other totally. Training started with low initial Loss. Then Loss of these optimizers reduced significantly and then also kept reducing through the training schedule.

Validation Loss Convergence:

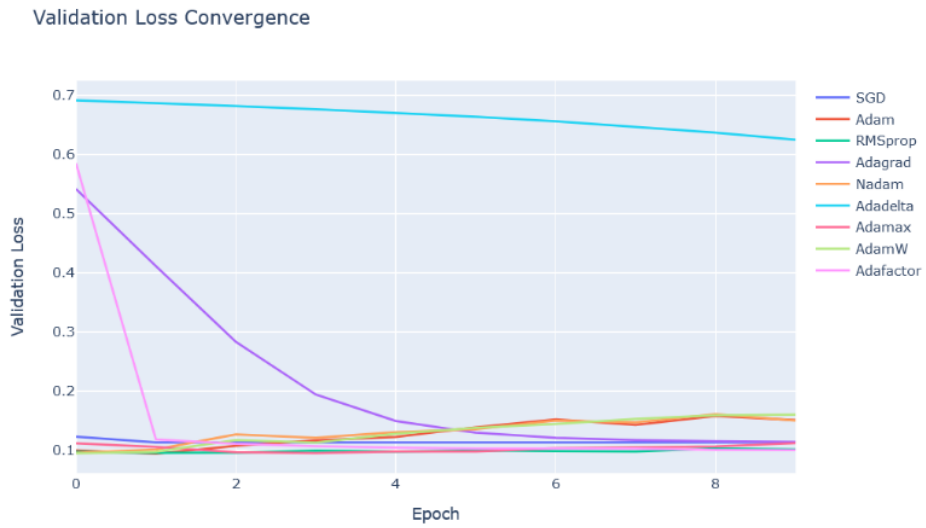


Figure 5.33 Experiment 3 - Validation Loss Convergence

Table 5.10 Experiment 3 - Validation Loss Data

Epoch	0	1	2	3	4	5
SGD	0.123348	0.113726	0.113465	0.113455	0.113441	0.113418
Adam	0.099952	0.094427	0.107712	0.116932	0.122892	0.139103
RMSprop	0.097968	0.095881	0.096148	0.099584	0.098037	0.100050
Adagrad	0.541711	0.411414	0.283286	0.194342	0.149651	0.129936
Nadam	0.095833	0.101514	0.127110	0.121392	0.131004	0.135792
Adadelta	0.691326	0.686966	0.681999	0.676524	0.670450	0.663647
Adamax	0.112022	0.105928	0.096843	0.095352	0.098348	0.098217
AdamW	0.094930	0.097775	0.117504	0.113276	0.128145	0.138129
Adafactor	0.585017	0.118461	0.111167	0.106977	0.104798	0.103277

Epoch	6	7	8	9
SGD	0.113434	0.113418	0.113427	0.113418
Adam	0.152449	0.143344	0.158275	0.151420
RMSprop	0.098937	0.098123	0.103218	0.101722
Adagrad	0.121276	0.117285	0.115385	0.114417
Nadam	0.150329	0.147552	0.161344	0.150269
Adadelta	0.655949	0.647140	0.636936	0.624979
Adamax	0.103754	0.105321	0.106559	0.112487
AdamW	0.144745	0.153105	0.159239	0.160322
Adafactor	0.103221	0.102887	0.100979	0.100583

From the graph and Validation Loss Data. It is found that:

- **Adadelta & Adagrad** are also performing on similar lines in validation loss. Since the model failed to learn with these optimizers. It is also seen failing to predict and generalize on unseen data.

- **Adafactor** performance is similarly mid-way compared to other optimizers. Model initially struggled to learn but then model was finally able to learn. Similarly, it is shown in Validation Loss initially loss was high, which reduced as training happens.
- **SGD, Adamax, RMSprop**
- **SGD, Adamax, RMSprop** started with low validation loss and kept the final loss till end in similar lines. It shows that model is able to generalize till starting and there is no significant loss in ability of model to generalize.
- **Adam, AdamW, Nadam** showed least validation losses in starting itself and as the training progress the validation loss increase. It shows that model ability to generalize kept decreasing with training.

From these evaluations, It can be deduced that Adam, AdamW & Nadam are quite performing well since the training loss for these optimizers started with low value and reduced to minimum in initial training itself. Validation loss also shows that model fitted initially is good enough and as training proceed ability of model to generalize reduce. Hence, it can be said that if early stopping criteria is implemented for these optimizers. They can perform very well and train complex models in less time saving energy and time significantly.

5.3.3.2 Criteria 2: Comparison of Final Validation Loss and Validation Accuracy achieved among different optimizers.

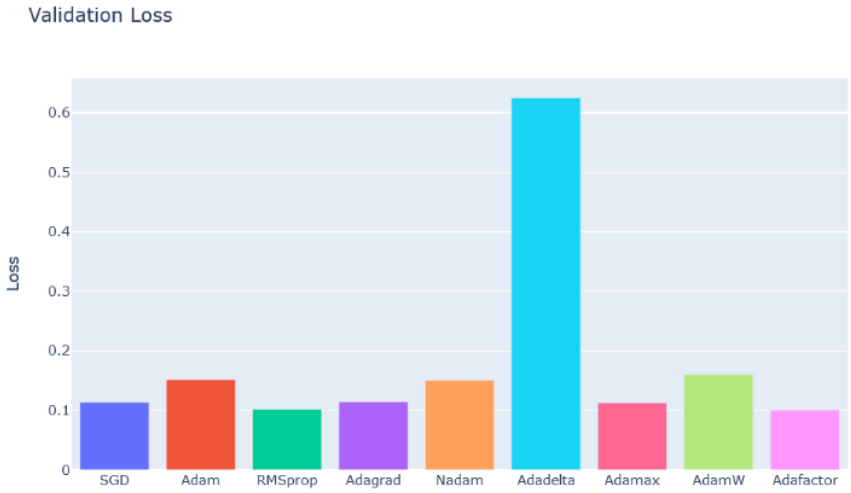


Figure 5.34 Experiment 3 - Final Validation Loss Comparison

Figure 5.35 Experiment 3 - Final Accuracy Comparison

Table 5.11 Experiment 3 - Final Validation Loss and Accuracy Data

	Optimizer	Validation_Loss	Validation_Accuracy
0	SGD	0.113418	0.975924
1	Adam	0.151420	0.971109
2	RMSprop	0.101722	0.973806
3	Adagrad	0.114417	0.975924
4	Nadam	0.150269	0.969569
5	Adadelta	0.624979	0.975924
6	Adamax	0.112487	0.973806
7	AdamW	0.160322	0.968028
8	Adafactor	0.100583	0.975924

From the above graphs and related data. It can be seen that:

- Validation accuracy is almost the same for all optimizers after 10 epochs. This shows that the models trained using different optimizers have nearly same predication capabilities.
- **Adam, AdamW & Nadam** reached their least losses in starting epochs only and afterwards losses started to increase. Final Loss values are 0.151, 0.150 & 0.160 respectively. Initially Losses were around 0.09 for all three.
- **SGD** performed quite well, and its final Loss is 0.1134.
- **Adadelta** has the highest loss earlier also checked that it failed to train the model properly. The final loss is 0.624.
- **Adamax, RMSprop, Adagrad, Adafactor** showed lesser final validation losses. These optimizers also showed good ability to train the model. Final Loss values are 0.112, 0.101, 0.114 & 0.100 respectively.

5.3.3.3 Criteria 3: Training Time Comparison

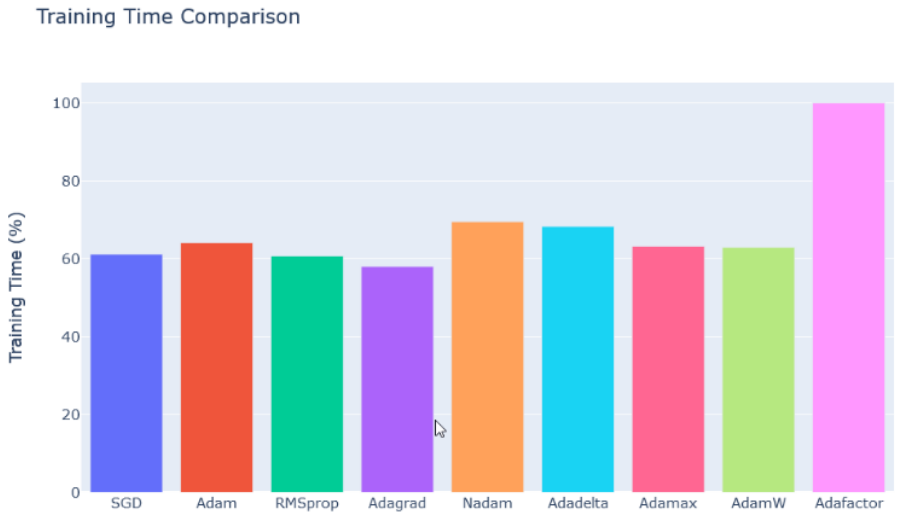


Figure 5.36 Experiment 3 - Training Time Comparison

Table 5.12 Experiment 3 - Training Time Data

	Optimizer	Time for trainings	Time_%_compared_to_max
0	SGD	53.203877	61.11
1	Adam	55.817967	64.11
2	RMSprop	52.844449	60.70
3	Adagrad	50.468906	57.97
4	Nadam	60.513444	69.50
5	Adadelta	59.464256	68.30
6	Adamax	55.018062	63.19
7	AdamW	54.809007	62.95
8	Adafactor	87.064111	100.00

From the given graph and data. It can be understood that:

- **Adam, AdamW & Nadam** reached their least losses for both training and validation sets in starting epochs only. Training time would have been significantly less if early stopping criteria were used in this case.
- **SGD, RMSprop, Adagrad, Adadelta, Adamax** all these optimizers took nearly same time to train the model. Hence, It can be seen that on timing criteria models are performing nearly same.

- **Adafactor** training and validation loss showed that this is also able to train model in initial training phase and later both the training and validation loss stabilized. Hence, early stopping can also be employed over here.

Training time is very important criteria, it should be evaluated in conjunction with Models training loss and validation loss which give an indication of amount of training required or In other sense it shows whether there is any benefit in training model further. Since training further only overfits the model though training accuracy increase but validation accuracy reduce since model start overfitting the data. Here, Adam, AdamW & Nadam showed the potential of good optimizer if early stopping criteria is applied.

5.3.3.4 Criteria 4: For each optimizer, Training Loss comparison with Validation Loss and Training Accuracy comparison with Validation Accuracy over the epochs.

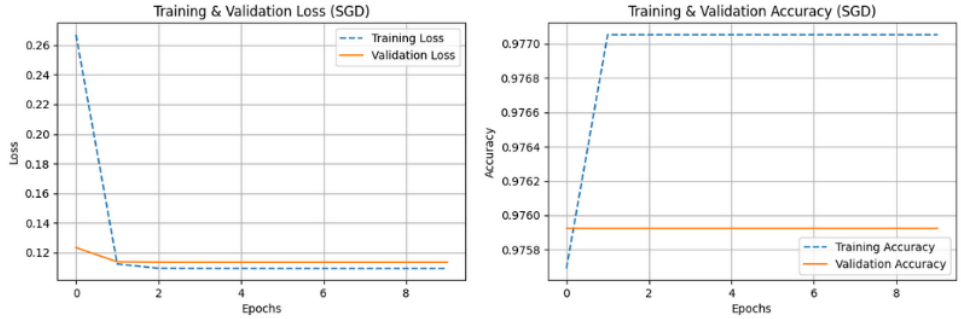


Figure 5.37 Experiment 3 - Training Curves - SGD

Optimizer SGD:

The convergence behavior of SGD optimizer for GRU is nearly same as LSTM and signifies same type of learning pattern. Good training happens and then stabilizes. Validation accuracy kept same value signifies that model learned but later no significant improvement observed.

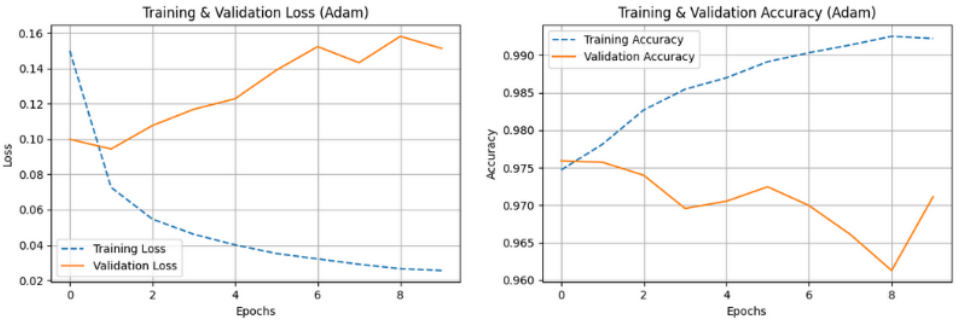


Figure 5.38 Experiment 3 - Training Curves - Adam

Optimizer Adam:

The convergence behavior of Adam optimizer for RNN with GRU. Is nearly same but the fluctuation is validation loss and accuracy has increased.

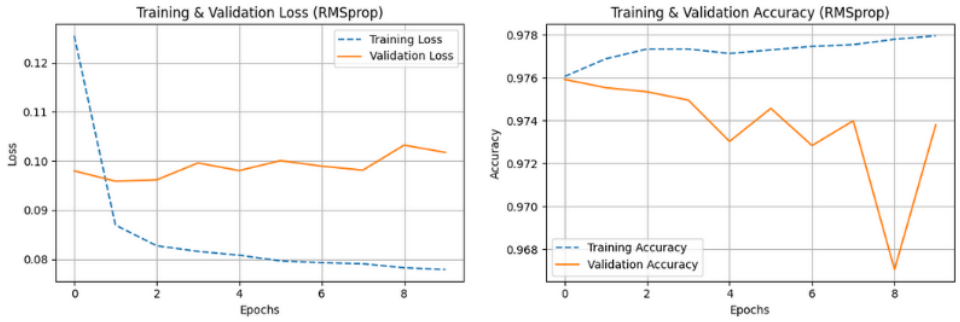


Figure 5.39 Experiment 3 - Training Curves - RMSprop

Optimizer RMSprop:

The graph reveals similar behavior as previous experiments. The notice worthy point is that the graphs have more fluctuations as compared to earlier. Validation accuracy suddenly dropped significantly at 9th epoch.

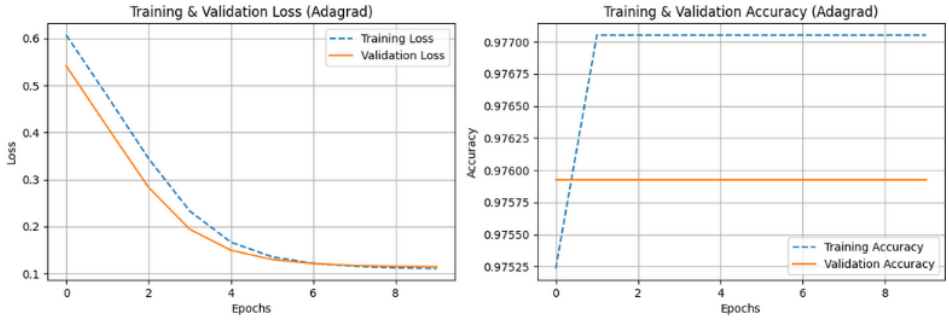


Figure 5.40 Experiment 3 - Training Curves - Adagrad

Optimizer Adagrad:

The graph for Adagrad also reveals similar behavior. Only change is that the training and validation loss stabilizes early around 6th epoch. Validation and training accuracy kept the same value through out training.

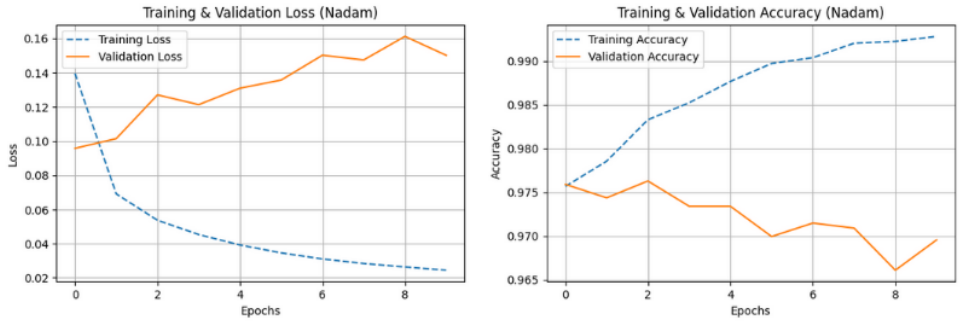


Figure 5.41 Experiment 3 - Training Curves - Nadam

Optimizer Nadam:

The graph shows nearly same training behavior of Nadam optimizer. Noteworthy points is that the fluctuation has increased significantly compared to previous.

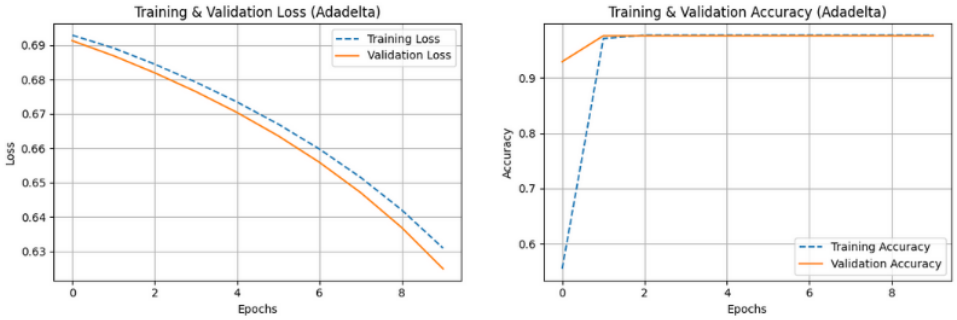


Figure 5.42 Experiment 3 - Training Curves - Adadelta

Optimizer Adadelta:

The above graphs show similar training behavior as LSTM model training curves. No significant change.

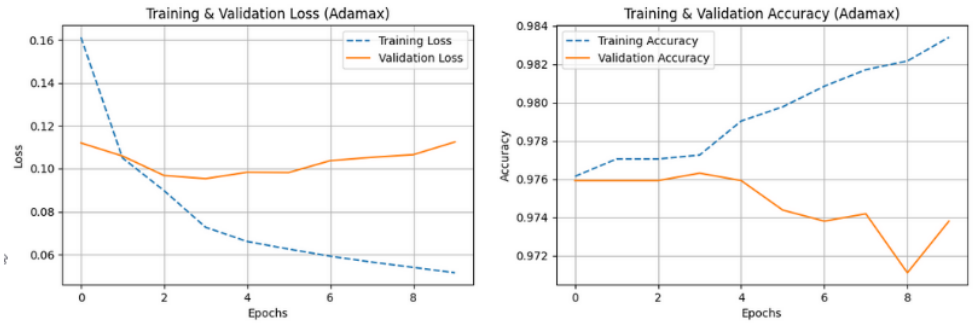


Figure 5.43 Experiment 3 - Training Curves - Adamax

Optimizer Adamax:

The above graphs show similar training characteristics as pervious. Only the fluctuation in training curve has increased.

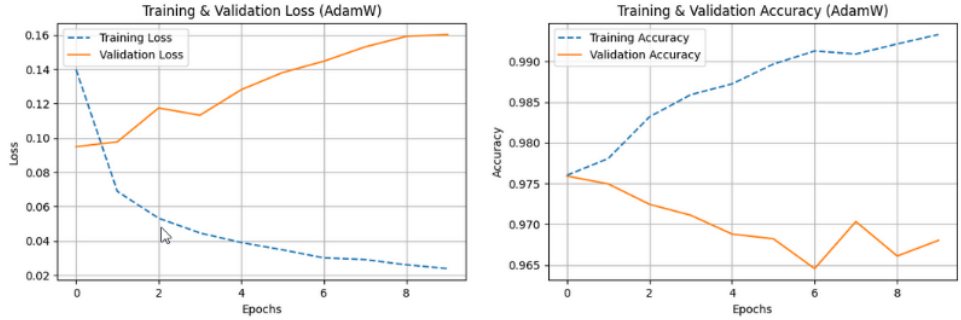


Figure 5.44 Experiment 3 - Training Curves - AdamW

Optimizer AdamW:

AdamW optimizer also shows similar training behavior as pervious training on LSTM.

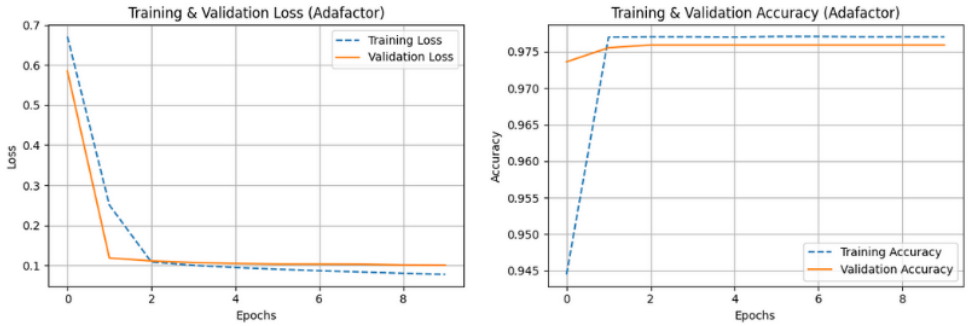


Figure 5.45 Experiment 3 - Training Curves - Adafactor

Optimizer Adafactor:

Adafactor optimizer also replicate the training behavior as previous trainings.

In summary, It can be seen that optimizer used for training RNN model with GRU Units exhibit nearly similar training behavior in terms of training loss, validation loss, training accuracy & validation accuracy. Important thing observed here is that the training behavior is similar but with added fluctuation in training graphs. Hence, it can be said that GRU units require more sophisticated training approach. Understanding and monitoring these fluctuations are essential for effectively managing the training process and ensuring that the model achieves the best possible performance and generalization on unseen data. Fine-tuning the model architecture, adjusting hyperparameters, and implementing regularization techniques can help stabilize these fluctuations and improve the overall model performance.

5.3.4 Experiment No. 4 – Basic Neural Network with tuned hyperparameter of optimizer

Tuning optimizer hyperparameters in deep learning neural networks has a huge impact on various aspects of the training process and model performance. Adjusting these parameters can significantly influence the convergence speed, stability, and generalization ability of the model, leading to faster convergence, smoother training, and improved capability to generalize to unseen data. By fine-tuning these hyperparameters, the model can become more robust to noise and variability in the training data, enabling it to learn meaningful patterns and make accurate predictions. Ultimately, optimizing optimizer hyperparameters is essential for achieving optimal model performance in terms of accuracy, precision, and overall predictive capability, ensuring the development of more effective and reliable deep learning models for diverse applications. Hence, optimizers are tuned for basic hyperparameters either to heuristics or to default value. Then the performance of each optimizer is evaluated against predefined criteria. Result & discussion follows.

5.3.4.1 Criteria 1: Monitoring Training Loss and Validation Loss over epochs.

Training Loss Convergence:

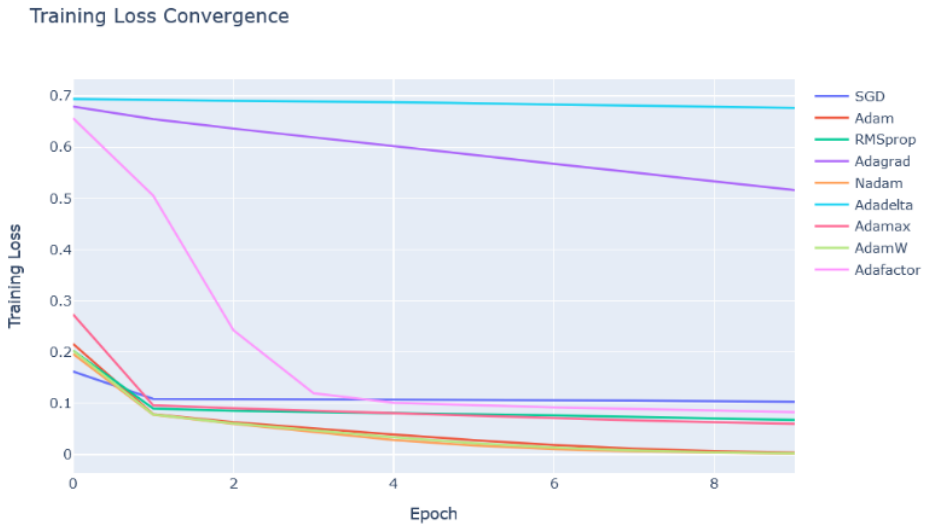


Figure 5.46 Experiment 4 - Training Loss Convergence

Table 5.13 Experiment 4 - Training Loss Data

Epoch	0	1	2	3	4	5
SGD	0.162086	0.108564	0.108182	0.107831	0.107362	0.106803
Adam	0.216065	0.078884	0.063165	0.050583	0.038921	0.027969
RMSprop	0.203141	0.089999	0.085481	0.082670	0.080445	0.078610
Adagrad	0.679271	0.654909	0.636244	0.618729	0.601571	0.584538
Nadam	0.196389	0.077664	0.059850	0.043779	0.028547	0.017533
Adadelta	0.693834	0.692538	0.691029	0.689350	0.687527	0.685572
Adamax	0.273560	0.096027	0.090196	0.085205	0.080170	0.075246
AdamW	0.202904	0.078093	0.060814	0.046921	0.033598	0.022269
Adafactor	0.656690	0.505309	0.242855	0.119503	0.101154	0.096107

Epoch	6	7	8	9
SGD	0.106143	0.105294	0.104299	0.102986
Adam	0.018377	0.011221	0.006540	0.003903
RMSprop	0.076329	0.073669	0.070835	0.067554
Adagrad	0.567526	0.550466	0.533323	0.516091
Nadam	0.010418	0.006038	0.003532	0.002178
Adadelta	0.683494	0.681301	0.678992	0.676567
Adamax	0.070636	0.066652	0.063119	0.059936
AdamW	0.013623	0.007856	0.004369	0.002624
Adafactor	0.092253	0.088851	0.085717	0.082763

From the graph and Training Loss Data. It is found that:

- **Adadelta & Adagrad** showed nearly a flat loss curve. It can be concluded that these optimizers failed to train these models well and performed very poorly since no significant training happened.
- **Adafactor** optimizer started with high loss value then showed significant reduction in training loss initially and then stabilized.
- **SGD, RMSprop, Adamax** started with moderate loss values and loss reduced in initial training phase and then model stabilized with slow reduction in loss values.
- **Adam, AdamW & Nadam** proved to be best optimizers since they started with quite less initial loss and good learning happened in early training phase and then afterwards also loss value kept reducing.

Validation Loss Convergence:

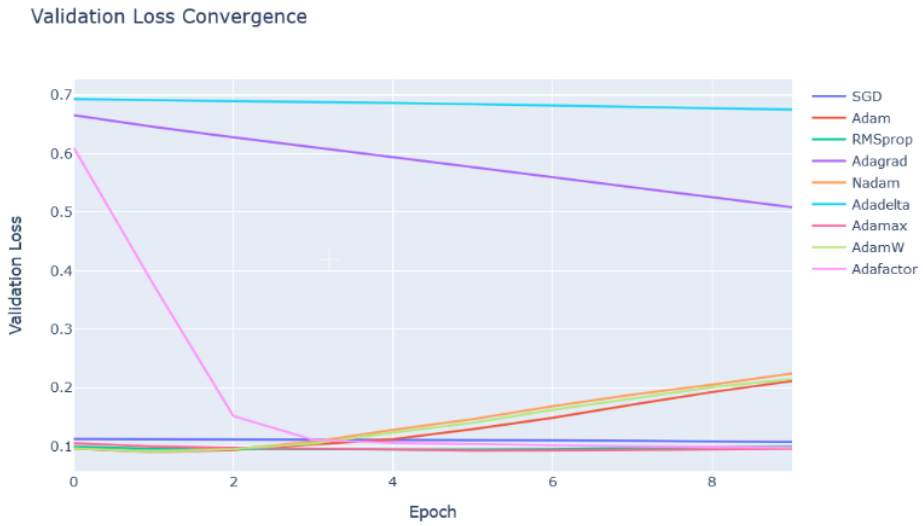


Figure 5.47 Experiment 4 - Validation Loss Convergence

Table 5.14 Experiment 4 - Validation Loss Data

Epoch	0	1	2	3	4	5
SGD	0.112585	0.112374	0.112131	0.111723	0.111323	0.110719
Adam	0.096091	0.090493	0.093949	0.103523	0.112277	0.129149
RMSprop	0.099359	0.095932	0.095269	0.094662	0.094460	0.094763
Adagrad	0.665451	0.645589	0.627779	0.610566	0.593560	0.576623
Nadam	0.096613	0.091017	0.094944	0.107738	0.128023	0.146336
Adadelta	0.692862	0.691458	0.689869	0.688124	0.686245	0.684241
Adamax	0.105215	0.099801	0.097358	0.095623	0.093873	0.092925
AdamW	0.096317	0.091079	0.095595	0.104842	0.123358	0.139924
Adafactor	0.610085	0.376338	0.151757	0.111163	0.105928	0.103430

Epoch	6	7	8	9
SGD	0.110067	0.109285	0.108346	0.107286
Adam	0.148470	0.171050	0.192643	0.211381
RMSprop	0.095529	0.097352	0.098811	0.099448
Adagrad	0.559668	0.542646	0.525532	0.508342
Nadam	0.168367	0.188345	0.205175	0.224124
Adadelta	0.682118	0.679879	0.677527	0.675056
Adamax	0.092998	0.093185	0.094539	0.095675
AdamW	0.162141	0.181013	0.200804	0.215029
Adafactor	0.101477	0.099942	0.098568	0.097676

From the graph and Validation Loss Data. It is found that:

- **Adadelta & Adagrad** performed poorly in validation loss also since it is evident that no significant training happened with these optimizers.
- **Adafactor** validation loss was quite high initially but decreased fast with training then stabilized. This shows that the model is able to generalize on unseen data and it is able to learn the patterns in data.
- **SGD, RMSprop, Adamax** achieved less validation loss since starting and kept nearly same values till end of training. This shows that models trained with these optimizers have good generalizing ability and this ability didn't face any change due to further training.
- **Adam, AdamW & Nadam** started with less validation loss values, which further reduced but after some time validation loss started increasing consistently. This shows that with further training models lost their ability to generalize and started overfitting. Early stopping should be employed in this case.

From these evaluations, it is evident that Adam, AdamW & Nadam reach a good validation loss value very initially in training and also training loss reduction slowed after initial training. Hence, this shows that data has started overfitting in this scenario. Hence, It can be deduced that Adam, AdamW & Nadam are efficient optimizer in text classification neural network but a careful early stopping mechanism should be utilized to save computation cost and reach a good generalized model else it will overfit the data and learn useless noise of data. Careful use of Adam, AdamW & Nadam can save energy and time significantly.

5.3.4.2 Criteria 2: Comparison of Final Validation Loss and Validation Accuracy achieved among different optimizers.

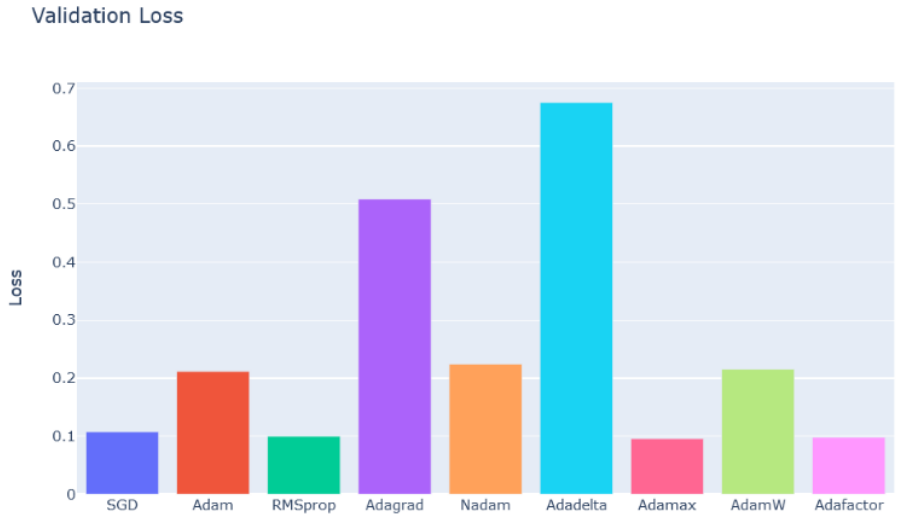


Figure 5.48 Experiment 4 - Final Validation Loss Comparison

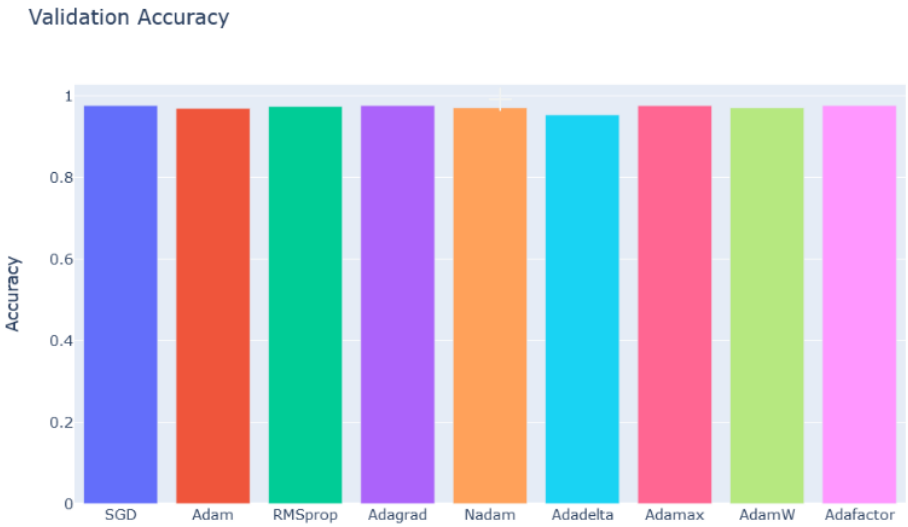


Figure 5.49 Experiment 4 - Final Validation Accuracy

Table 5.15 Experiment 4 - Final Validation Loss & Accuracy Data

	Optimizer	Validation_Loss	Validation_Accuracy
0	SGD	0.107286	0.975924
1	Adam	0.211381	0.969569
2	RMSprop	0.099448	0.973613
3	Adagrad	0.508342	0.975924
4	Nadam	0.224124	0.970339
5	Adadelta	0.675056	0.953390
6	Adamax	0.095675	0.975539
7	AdamW	0.215029	0.970339
8	Adafactor	0.097676	0.975924

From the above graphs and related data. It can be seen that:

- Validation accuracy is almost the same for all optimizers after 10 epochs.
- **Adam, AdamW & Nadam** reached their least loss in around initially and afterwards loss started to increase. Final Loss values are 0.211, 0.215 & 0.224 respectively.
- **SGD, RMSprop, Adamax & Adafactor** have similar final loss values. Final loss values are 0.107, 0.099, 0.095 & 0.097 respectively.
- **Adadelta & Adagrad** final losses values are significantly higher than peers 0.508 & 0.675 respectively. This is quite evident since these optimizers failed to train the models.

5.3.4.3 Criteria 3: Training Time Comparison

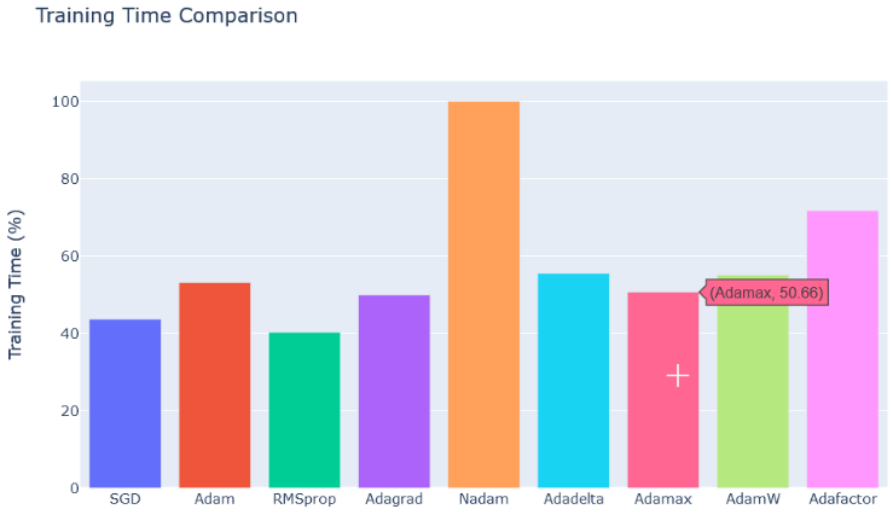


Figure 5.50 Experiment 4 - Training Time Comparison

Table 5.16 Experiment 4 - Training Time Comparison Data

	Optimizer	Time for trainings	Time_%_compared_to_max
0	SGD	9.426869	43.67
1	Adam	11.478672	53.18
2	RMSprop	8.692890	40.27
3	Adagrad	10.771333	49.90
4	Nadam	21.584642	100.00
5	Adadelta	11.977521	55.49
6	Adamax	10.934264	50.66
7	AdamW	11.884533	55.06
8	Adafactor	15.494371	71.78

From the above graph and data. It can be understood that:

- **Adam, AdamW & Nadam** reached their least loss in starting face of training. Hence In case early stopping was employed. There optimizers would have taken around the least timing in comparison to others.
- **SGD and RMSprop** performed well on timing criteria.
- **Adafactor** optimizer took quite a long time for training.
- **Adamax** took nearly the same time for training as other optimizers.

- **Adadelta & Adagrad** takes medium time for training over 10 epochs but as found earlier that there is no significant update in Validation Loss. So, it will not be advisable to keep training the model when improvement is not happening.

5.3.4.4 Criteria 4: For each optimizer, Training Loss comparison with Validation Loss and Training Accuracy comparison with Validation Accuracy over the epochs.

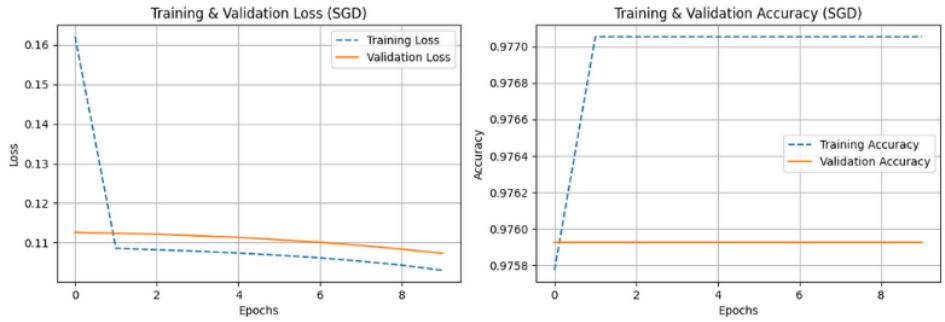


Figure 5.51 Experiment 4 - Training Curves - SGD

Optimizer SGD:

The above shows that tuned SGD optimizer is able to train the model easily and training and validation loss stabilizes early. Training and validation accuracy also stabilizes very early in training.

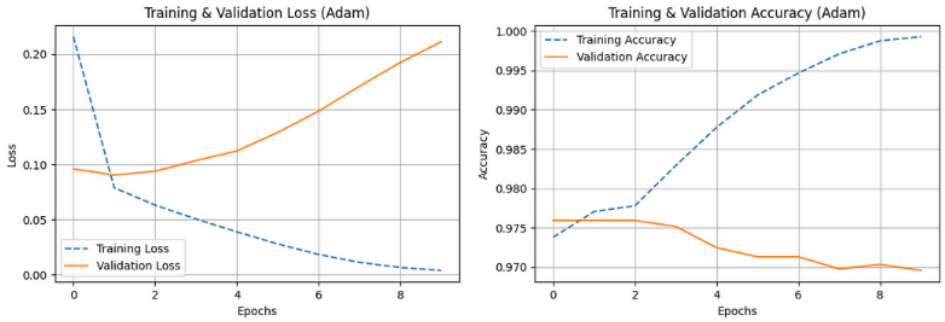


Figure 5.52 Experiment 4 - Training Curves - Adam

Optimizer Adam:

The graph reveals exactly the same training curve as basic NN model training. It may be because default values are similar to earlier trained Basic NN model.

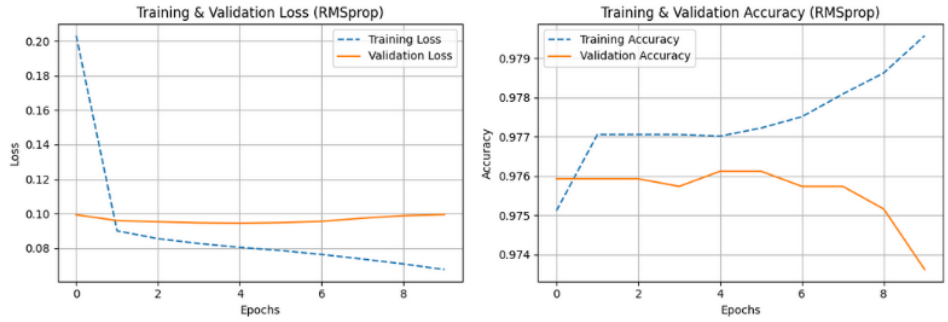


Figure 5.53 Experiment 4 - Training Curves - RMSprop

Optimizer RMSprop:

These graphs also reveal similar training curves as basis NN Model training.

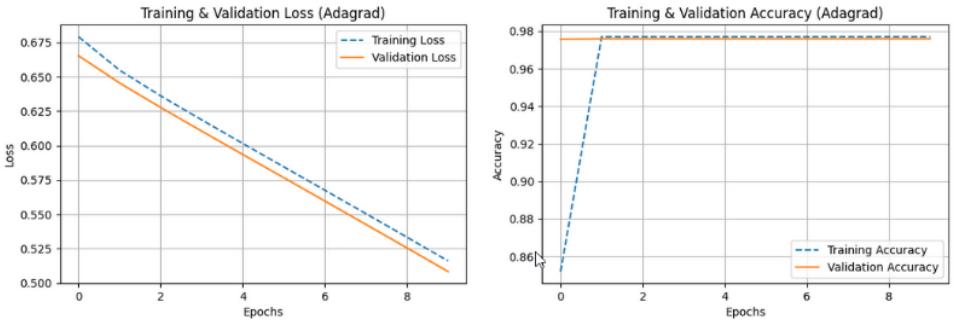


Figure 5.54 Experiment 4 - Training Curves - Adagrad

Optimizer Adagrad:

These graphs reveal similar training curve as basic NN Model training.

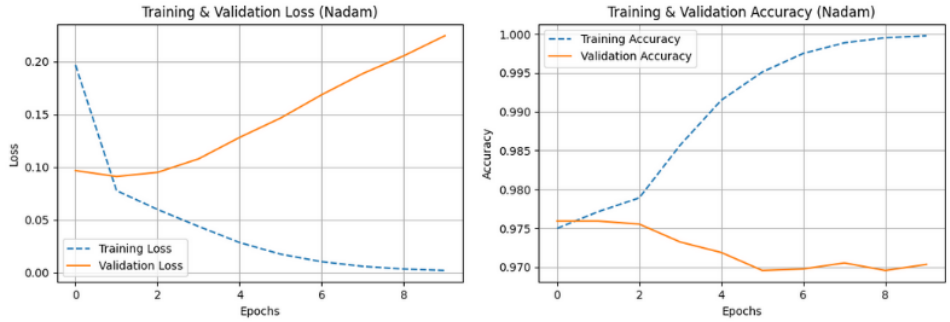


Figure 5.55 Experiment 4 - Training Curves - Nadam

Optimizer Nadam:

The graph of Nadam optimizer also reveals a similar training characteristic to Basic NN Model.

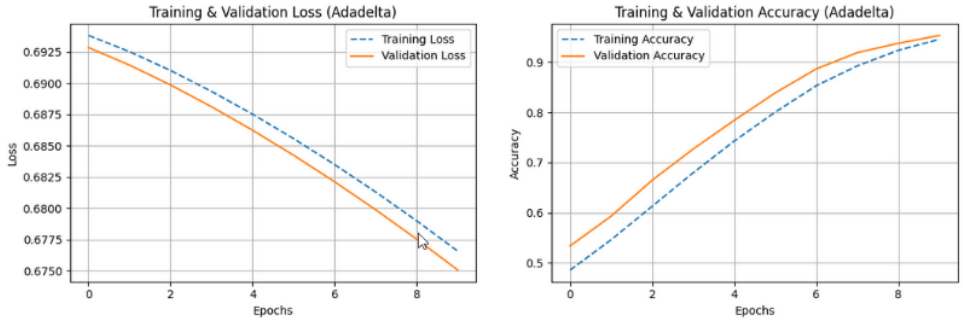


Figure 5.56 Experiment 4 - Training Curves - Adadelta

Optimizer Adadelta:

Training graphs of optimizer Adadelta shows a similar training characteristic as Basic NN Model. Curves are smooth, training and validation loss decrease smoothly over the training period and the Training and Validation accuracy increase smoothly over the training period.

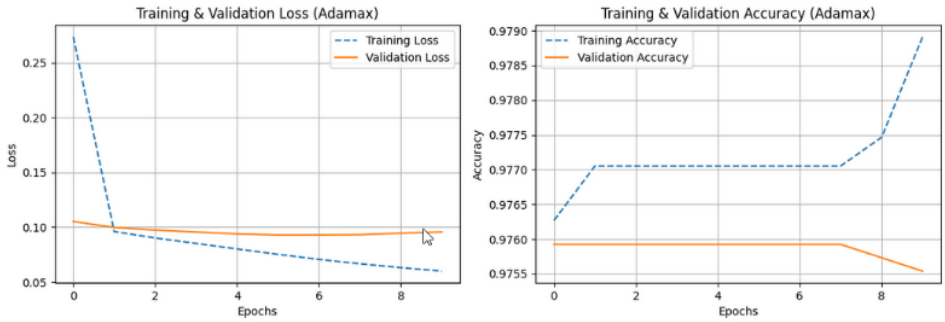


Figure 5.57 Experiment 4 - Training Curves - Adamax

Optimizer Adamax:

Adamax optimizer also shows similar behavior.

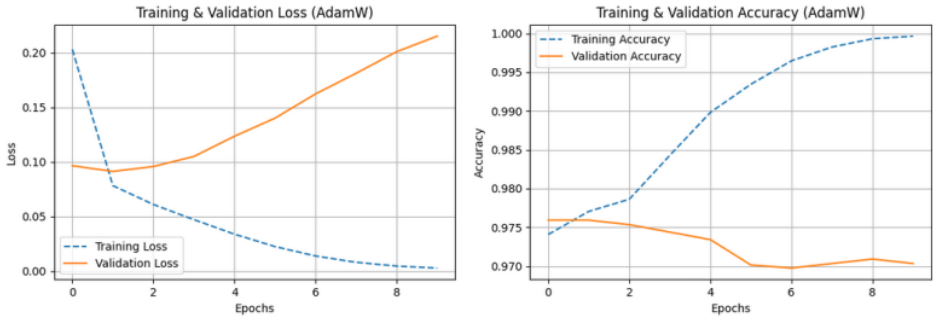


Figure 5.58 Experiment 4 - Training Curves - AdamW

Optimizer AdamW:

AdamW optimizer training graphs are similar to previous training graphs of Basic NN Models.

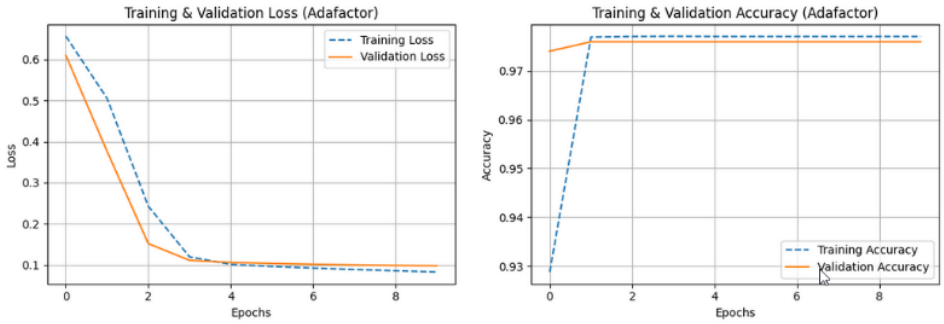


Figure 5.59 Experiment 4 - Training Curves - Adafactor

Optimizer Adafactor:

Adafactor optimizer also shows similar graph as pervious Basic NN Models but in this case training accuracy starts with better value.

5.4 Results of Experiment

After a comprehensive research work undertaken and in-depth study over 4 different set of experiments. Detailed analysis of Training Loss, Validation Loss, Training Accuracy & Validation Accuracy, Time comparison and internal comparison of Training curves of each optimizer. Following inferences can be drawn:

- **Adam, AdamW, and Nadam** demonstrated superior convergence and generalization performance.
- These optimizers exhibited stability, reduced fluctuations in training and validation losses, and faster convergence.
- Efficient training times were observed with Adam, AdamW, and Nadam, without compromising model performance.
- The study revealed the role of adaptive learning rates in capturing intricate textual patterns and nuances.
- Findings indicate that **Adam, AdamW, and Nadam** are optimal for text classification tasks, expediting training without compromising predictive power.

5.5 Summary

The results of the conducted experiments demonstrated the superior performance of Adam, AdamW, and Nadam optimizers in effectively training deep learning models for text classification tasks. Across various deep learning architectures, these optimizers exhibited accelerated convergence, improved generalization capabilities, and reduced training time, underscoring their suitability for handling textual data. These findings emphasize the promising potential of adaptive optimization algorithms in enhancing the efficiency and effectiveness of deep learning methodologies for text-based applications.

CHAPTER 6 : CONCLUSIONS AND RECOMMENDATIONS

"In this Research work, the efficacy of various optimization algorithms in the context of training deep neural networks for text classification tasks were investigated. Through a comprehensive analysis of the performance and behaviour of different optimizers, it was aimed to shed light on the intricate relationship between optimization techniques and the convergence dynamics of deep learning models. By addressing the important need for understanding the impact of optimizers on model training, this research contributes to the ongoing efforts in enhancing the efficiency and effectiveness of deep learning methodologies. In this concluding chapter, we summarize the key findings, discuss their implications, and propose potential avenues for future research in the field of optimization for deep neural networks."

6.1 Conclusion

This research work was thoroughly implemented. The research focus was to evaluate the old and new optimizers available in Keras library for their efficacy in training deep neural networks. Multiple different criteria were decided for creating a comprehensive evaluation so that complex behavior of these algorithms can be understood. A detailed discussion on these obtained results is done in the Results and Evaluation Chapter.

The following table is used to create an overview of the experiment and its results.
Based on comprehensive results and their evaluation following table is populated. Best, Good and poor 3 categories are used to classify the results.

Table 6.1 Summary of comparison of optimizers

Optimizer	Training and Generalizing ability	Time Requirement	Remarks
SGD	Good	Good	Good for training the model with average performance.
Adam	Best	Best	Efficient and quick in training with less time.
RMSprop	Good	Good	Average performance in training.
Adagrad	Poor	Good	Model will take very long time for training.
Nadam	Best	Best	Efficient and quick in training with less time.
Adadelta	Poor	Good	Unable to train model.
Adamax	Good	Good	Average performance in training.
AdamW	Best	Best	Efficient and quick in training with less time.
Adafactor	Good	Poor	Initial loss is high, but it can train.

In the course of this study, our investigation into the efficacy of optimization algorithms for text classification tasks involved the implementation and evaluation of four distinct deep learning models. Each model was trained using all available optimizers in Keras Library with their respective hyperparameters tailored to the intricacies of the text classification problem at hand.

The training process was closely monitored, and detailed records of training loss, validation loss, training accuracy, and validation accuracy were meticulously documented for each epoch. Notably, the experimental results consistently showcased the superiority of Adam, AdamW, and Nadam in facilitating faster convergence and superior generalization performance across all four deep learning architectures. These optimizers exhibited a remarkable ability to effectively navigate the complex optimization landscape, demonstrating reduced fluctuations in the training and validation losses, and maintaining a higher level of stability in the learning process compared to alternative optimization techniques.

Furthermore, the time required for training each model with the designated optimizers was carefully recorded and analyzed. The results demonstrated the efficiency of Adam, AdamW, and Nadam in significantly reducing the training time while ensuring robust model performance. The streamlined convergence patterns observed in the training and validation curves further accentuated the efficiency of these optimizers in facilitating rapid convergence without compromising the model's generalization capabilities.

The graphical behaviors of the optimization processes were thoroughly studied, revealing distinctive patterns in the learning dynamics of the different models. Notably, the smooth and consistent trajectories exhibited by the training and validation curves under the influence of Adam, AdamW, and Nadam underscored the stable and effective learning dynamics facilitated by these optimizers. The adaptive learning rates offered by these algorithms played a pivotal role in enabling the models to swiftly adapt to the underlying data distribution, effectively capturing intricate textual patterns and nuances crucial for accurate text classification.

Consequently, based on the comprehensive analysis of the experimental results, it is evident that the choice of Adam, AdamW, and Nadam as the preferred optimization algorithms for text classification tasks is substantiated by their collective ability to expedite training without compromising the model's predictive power. This study not only advances our understanding of optimization strategies in deep learning but also highlights the critical role played by adaptive optimization algorithms in the context of text analytics. While the findings of this research provide valuable insights into the efficacy of these optimizers, future studies could delve deeper into exploring their applicability in other domains and their potential interactions with different model architectures, thus fostering continued advancements in the field of deep learning optimization techniques.

6.2 Contribution to Knowledge

The findings of this study significantly contribute to the global knowledge pool by providing empirical evidence of the superior performance of Adam, AdamW, and Nadam optimizers in the context of training deep learning models for text classification tasks. Through detailed experimentation and analysis, this research not only reaffirms the effectiveness of adaptive optimization algorithms in enhancing the convergence speed and generalization capability of deep neural networks but also sheds light on their potential applications in various natural language processing domains. The documented insights into the nuanced behaviours of these optimizers, along with the comprehensive evaluation of their impact on different model architectures, serve to enrich the collective understanding of optimization techniques in the field of deep learning. Moreover, the meticulous exploration of the graphical behaviours and time requirements for training underscores the practical implications of employing these optimizers, offering valuable guidance to researchers and practitioners aiming to leverage deep learning methodologies for text-based applications.

6.3 Future Scope & Recommendations

As this research journey is concluded, not only summary is provided but also offer future work recommendations that can guide future research works in field of text classification optimizations. Moving forward, an extension of this research could involve exploring the behavior of Adam, AdamW, and Nadam optimizers in conjunction with other state-of-the-art model architectures, such as BERT and GPT. Investigating the convergence dynamics and generalization capabilities of these advanced architectures in diverse text classification tasks would provide valuable insights into the interplay between sophisticated models and adaptive optimization techniques, thereby fostering a comprehensive understanding of their combined efficacy in handling complex natural language processing challenges.

Additionally, the current study was conducted on a singular dataset that exhibited discernible patterns readily learnable by deep neural networks. To comprehensively assess the robustness and efficacy of the identified optimizers, future research endeavors should incorporate more challenging datasets characterized by intricate and diverse learning patterns. Introducing datasets with varied linguistic nuances, noisy text, and domain-specific complexities would enable a more comprehensive evaluation of the optimizers' performance across a spectrum of real-world text classification scenarios.

To further generalize the findings of this research, a comprehensive study involving multiple diverse textual datasets would be imperative. By systematically evaluating the performance of

Adam, AdamW, and Nadam across different domains, genres, and languages, a more robust understanding of the optimizers' adaptability and versatility in various text classification contexts can be established. This holistic analysis would provide a more nuanced perspective on the applicability and generalizability of the identified optimizers, contributing to a broader understanding of their effectiveness in addressing the intricacies inherent in diverse text-based applications.

Furthermore, an in-depth exploration of the hyperparameter settings and their impact on the optimization process could offer valuable insights into fine-tuning strategies for achieving optimal performance with the identified optimizers. Conducting systematic experiments to assess the sensitivity of the models to different hyperparameter configurations would facilitate the identification of robust optimization settings, thereby enhancing the reproducibility and reliability of the results across different experimental settings.

Overall, these suggested avenues for future research would not only advance our understanding of optimization techniques for text classification but also foster the development of more robust and efficient deep learning methodologies for a wide array of real-world applications.

6.4 Summary

In conclusion, this research has confirmed the superiority of Adam, AdamW, and Nadam optimizers in enhancing the training efficiency and generalization performance of deep learning models for text classification tasks. The findings highlight the adaptability of these optimization algorithms in effectively navigating the complexities of textual data, thereby facilitating accelerated convergence and robust model performance. However, to consolidate these findings, future research should explore the interaction of these optimizers with advanced model architectures like BERT and GPT, as well as evaluate their efficacy on more challenging datasets with intricate learning patterns. Moreover, a comprehensive study involving multiple diverse textual datasets would be instrumental in establishing the generalizability and applicability of the identified optimizers across various text classification contexts. This comprehensive understanding will contribute significantly to the advancement of optimization techniques for deep learning, fostering the development of more robust and efficient methodologies for a wide array of real-world applications in natural language processing.

References:

Al-Anzi, F.S., (2022a) An Effective Hybrid Stochastic Gradient Descent Arabic Sentiment Analysis with Partial-Order Microwords and Piecewise Differentiation. In: *2022 9th International Conference on Electrical and Electronics Engineering (ICEEE)*. IEEE, pp.408–411.

Al-Anzi, F.S., (2022b) An Effective Hybrid Stochastic Gradient Descent for Classification of Short Text Communication in E- Learning Environments. In: *2022 8th International Conference on Control, Decision and Information Technologies (CoDIT)*. [online] IEEE, pp.1096–1101. Available at: https://ieeexplore.ieee.org/document/9804138/.

Cahyani, D.E. and Nuzry, K.A.P., (2019) Trending Topic Classification for Single-Label Using Multinomial Naive Bayes (MNB) and Multi-Label Using K-Nearest Neighbors (KNN). In: *2019 4th International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE)*. [online] IEEE, pp.547–552. Available at: https://ieeexplore.ieee.org/document/9003944/.

Chung, J., Gulcehre, C., Cho, K. and Bengio, Y., (2014) Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. [online] Available at: https://arxiv.org/abs/1412.3555v1 [Accessed 22 Oct. 2023].

Daud, S., Ullah, M., Rehman, A., Saba, T., Damaševičius, R. and Sattar, A., (2023) Topic Classification of Online News Articles Using Optimized Machine Learning Models. *Computers*, 121, p.16.

Diab, S., (2019) *Optimizing Stochastic Gradient Descent in Text Classification Based on Fine-Tuning Hyper-Parameters Approach. A Case Study on Automatic Classification of Global Terrorist Attacks*. [online] Available at: https://sites.google.com/site/ijcsis/.

Goldberg, Y., Levy, O., Mikolov, T., Sutskever, I., Chen, K., Corrado, G. and Dean, J., (2014) *word2vecExplained: Deriving Mikolov et al.'sNegative-Sampling Word-Embedding Method*. [online] Available at: https://code.google.com/p/word2vec/.

Goswami, M. and Sabata, P., (2021) Evaluation of ML-Based Sentiment Analysis Techniques with Stochastic Gradient Descent and Logistic Regression. pp.153–163.

Gowri, S., Jenila, J., Reddy, B.S. and Sheela, M.A., (2021) Scrutinizing of Fake News using Machine Learning Techniques. In: *2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS)*. IEEE, pp.223–227.

Hassan, N., Gomaa, W., Khoriba, G. and Haggag, M., (2020) Credibility Detection in Twitter Using Word N-gram Analysis and Supervised Machine Learning Techniques. *International Journal of Intelligent Engineering and Systems*, 131, pp.291–300.

Hochreiter, S. and Schmidhuber, J., (1997) Long Short-Term Memory. *Neural Computation*, 98, pp.1735–1780.

Kim, Y., (n.d.) *Convolutional Neural Networks for Sentence Classification*. [online] Available at: http://nlp.stanford.edu/sentiment/.

Korenius, T., Laurikkala, J., Järvelin, K. and Juhola, M., (2004) Stemming and lemmatization in the clustering of finnish text documents. In: *Proceedings of the thirteenth ACM international conference on Information and knowledge management*. New York, NY, USA: ACM, pp.625–633.

Kowsari, Jafari Meimandi, Heidarysafa, Mendu, Barnes and Brown, (2019) Text Classification Algorithms: A Survey. *Information*, 104, p.150.

Madhfar, M.A.H. and Al-Hagery, M.A.H., (2019) Arabic Text Classification: A Comparative Approach Using a Big Dataset. In: *2019 International Conference on Computer and Information Sciences (ICCIS)*. IEEE, pp.1–5.

Mikolov, T., Chen, K., Corrado, G. and Dean, J., (n.d.) *Efficient Estimation of Word Representations in Vector Space*. [online] Available at: http://ronan.collobert.com/senna/.

Mikolov, T., Chen, K., Corrado, G. and Dean, J., (n.d.) Efficient Estimation of Word Representations in Vector Space. [online] Available at: http://ronan.collobert.com/senna/ [Accessed 22 Oct. 2023b].

Patel, A. and Meehan, K., (2021) Fake News Detection on Reddit Utilising CountVectorizer and Term Frequency-Inverse Document Frequency with Logistic Regression, MultinominalNB and Support Vector Machine. In: *2021 32nd Irish Signals and Systems Conference (ISSC)*. IEEE, pp.1–6.

Peng, H., Yu, P.S., Li, Q., Li, J., Xia, C., Yang, R., Sun, L., He, L., Peng, H., Li, J., Xia, C. and Yu, P.S., (2022) 31 A Survey on Text Classification: From Traditional to Deep Learning A Survey on Text Classification: From Traditional to Deep Learning. *ACM Transactions on Intelligent Systems and Technology*, [online] 132, p.31. Available at: https://doi.org/10.1145/3495162.

Pennington, J., Socher, R. and Manning, C.D., (n.d.) GloVe: Global Vectors for Word Representation. [online] pp.1532–1543. Available at: http://nlp. [Accessed 22 Oct. 2023].

Pimpalkar, A. and Raj, R.J.R., (2020) Evaluation of Tweets for Content Analysis Using Machine Learning Models. In: *2020 12th International Conference on Computational Intelligence and Communication Networks (CICN)*. IEEE, pp.454–459.

Plisson, J., Lavrac, N. and Mladenic, D., (n.d.) A Rule based Approach to Word Lemmatization.

Prasetijo, A.B., Isnanto, R.R., Eridani, D., Soetrisno, Y.A.A., Arfan, M. and Sofwan, A., (2017) Hoax detection system on Indonesian news sites based on text classification using SVM and SGD. In: *2017 4th International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE)*. IEEE, pp.45–49.

Roshinta, T.A., Hartatik, Fauziyah, E.K., Dinata, I.F., Firdaus, N. and A'la, F.Y., (2022) A Comparison of Text Classification Methods: Towards Fake News Detection for Indonesian Websites. In: *2022 1st International Conference on Smart Technology, Applied Informatics, and Engineering (APICS)*. IEEE, pp.59–64.

Salton, G. and Buckley, C., (1988) Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 245, pp.513–523.

Shah, K., Patel, H., Sanghvi, D. and Shah, M., (2020) A Comparative Analysis of Logistic Regression, Random Forest and KNN Models for the Text Classification. *Augmented Human Research*, 51, p.12.

Shahreen, N., Subhani, M. and Mahfuzur Rahman, M., (2018) Suicidal Trend Analysis of Twitter Using Machine Learning and Neural Network. In: *2018 International Conference on Bangla Speech and Language Processing (ICBSLP)*. IEEE, pp.1–5.

Sutskever, I., Martens, J. and Hinton, G., (2011) Generating Text with Recurrent Neural Networks.

Varshney, C.J., Sharma, A. and Yadav, D.P., (2020) Sentiment Analysis using Ensemble Classification Technique. In: *2020 IEEE Students Conference on Engineering & Systems (SCES)*. IEEE, pp.1–6.

Zhang, Y., Jin, R. and Zhou, Z.-H., (n.d.) Understanding bag-of-words model: a statistical framework.

APPENDIX A RESEARCH PROPOSAL

AN IMPROVED VERSION OF STOCHASTIC GRADIENT DESCENT ALGORITHM FOR PROCESSING TEXT DATA IN CLASSIFICATION TASKS.

SHAILENDRA KUMAR

Research Proposal

FEBRUARY 2023

Abstract

The Advent of digital era has made the vast and varied amount of information accessible to everyone. With every passing day the amount of textual data and the speed of its generation is exponentially increasing. Social media sites, blogs, newspaper, e-commerce, and e-learning sites are generating data around the clock. Therefore, there is huge requirement and demand of advanced systems and algorithm for classification of these textual data. Text Classification tasks comes under the field of NLP (Natural Language Processing). NLP is the subfield of Artificial Intelligence focused on understanding and interpreting natural language using machines. Logistic regression algorithm has outperformed most of all other algorithms including SGD in this use case. SGD (Stochastic Gradient Descent) is an optimization algorithm for training data. SGD has been versatile in deep learning due to its properties like scalability, efficient, easy implementation but it shows some limitation in this use case mostly attributed to sparse nature of training data. Here, an improved SGD algorithm is proposed for training textual data. This will be achieved through comparison of basic and improved SGD. This would help in efficient and accurate classification of textual data. Which consequently will help everyone to easily find content of their choice, enhance knowledge in desired field and make better informed decision.

Table of Contents

1. Background

Natural Language Processing is the sub field of artificial intelligence focussed on developing systems to understand and interpret human language. NLP involves multiple tasks like text classification, sentiment analysis, translation, text generation. Each of these tasks are very important as it handles some specific challenge in understanding human language by machine. This field is also rapidly evolving due to availability and generation of humungous data every second over internet (Hassan et al., 2020; Pimpalkar and Raj, 2020; Shah et al., 2020; Goswami and Sabata, 2021; Al-Anzi, 2022b).

Text classification is one such task involving classification of given text into predefined categories. Sentiment Analysis is also similar task which categorize given piece of text in positive or negative category. Many algorithms specifically Logistic Regression & Support Vector Machines have proven to be good enough to successfully do text classification up to desired level. Logistic Regression & Support Vector Machines algorithms performs better that the Stochastic Gradient Descent (Prasetijo et al., 2017; Madhfar and Al-Hagery, 2019; Goswami and Sabata, 2021; Roshinta et al., 2022; Daud et al., 2023). Stochastic Gradient Descent algorithm is an optimization algorithm. It has been widely used for training machine learning models due to its flexibility, efficiency, easy implementation, and convergence properties (Su and Kek, 2021). Its performance in text classification related task has not been up to mark. Opportunities for improvement in SGD are explored. This work is focussed on making some improvement in SGD algorithm so that it becomes easy to train text classification models. Due to generation of humongous data day by day. It becomes more pertinent to have a highly efficient and accurate algorithm to train such text classification-based models. It will save precious compute resource together with improved user experience due to better classification.

2. Problem Statement OR Related Research OR Related Work

In recent time, there has been huge addition to the corpus of textual data and it is ever increasing. A lot of online platforms like social, educational and e-commerece have created possibility of different types of textual data. Data generation has now become very easy. Other type of complex literature is also now very easily accessible to everyone. Thanks to the new information age. In this information age text classification has become even more important than ever. It is one of the subdivisions of Natural Language Processing (Hassan et al., 2020; Pimpalkar and Raj, 2020; Shah et al., 2020; Goswami and Sabata, 2021; Patel and Meehan, 2021; Al-Anzi, 2022b).

Text classification has been actively researched and new development are done using Machine Learning algorithms. This has been observed that Logistic Regression and other algorithms perform better than SGD algorithms in text classification or sentiment analysis (Madhfar and Al-Hagery, 2019; Goswami and Sabata, 2021; Roshinta et al., 2022). Different pre-processing steps text cleaning, tokenization, stop word removal, stemming or lemmatization is required for making data suitable for feature extraction. Then feature extraction is done using approaches like count vectorizer, n-gram, bag of words, partially ordered microword, TF-IDF. Of these techniques TF-IDF is most found to be most widely implemented and used (Prasetijo et al., 2017; Diab, 2019; Shah et al., 2020; Gowri et al., 2021; Patel and Meehan, 2021; Al-Anzi, 2022b; a). Then this data is used for training different machine learning models. Multiple classification algorithms have been implemented and evaluated by researchers over last 5 years for text classification tasks namely Logistic Regression, SGD Classifier, Decision Tree, Support Vector Machines, Random Forest, K Nearest Neighbhor, XGBoost, Naïve Bayes, Multinomial Naïve Bayes. It is found that Logistic Regression and sometimes SVM (Support Vector Machine) outperform other algorithms including SGD in most of the text classification tasks (Prasetijo et al., 2017; Madhfar and Al-Hagery, 2019; Goswami and Sabata, 2021; Roshinta et al., 2022; Daud et al., 2023).

Some studies have been conducted to improve and fine tune SGD algorithm for these types of tasks (Diab, 2019; Al-Anzi, 2022b). There seems some scope of improvement in SGD because researchers are continuously working to improve SGD due to its varied and extensive use in machine learning. Some improvements in SGD are also proposed and evaluated by the researchers (Diab, 2019; Al-Anzi, 2022b). Basic challenge in training on textual data is because of sparse nature of training data. Feature representation techniques like bag of word and tf-idf create sparse matrix. Sparse nature training data make it difficult for SGD algorithm to train efficiently and accurately on the dataset.

There has also been a lot of research done to address the challenges like efficient and accurate training using SGD Algorithm. Here we are trying to improve SGD algorithm by making some modification. We will use modified SGD to train on the dataset and try to establish and study effect of this modification empirically (Su and Kek, 2021).

3. Research Questions (If any)

In this thesis, A modification in SGD algorithm is proposed for better training on textual data. A comparative study with Benchmark Algorithms i.e., Logistic Regression and Support Vector Machines will be done on dataset to establish results.

4. Aim and Objectives

The main aim of this study is to propose an improved version of Stochastic Gradient Descent algorithm for efficient and accurate classification of textual data. The use of new improved Stochastic Gradient Descent algorithm will make training of textual data more efficient and accurate. Text documents can be classified into proper categories in less time with more accuracy.

The objectives of research for this study are as follows:

- To formulate the algorithm.
- To pre-process and do feature extraction on the dataset to create training data suitable for training and evaluation.
- To apply the basic and improved algorithm and train the classification models.
- To evaluate the performance of improved SGD compared to basic algorithms.

5. Significance of the Study

In today's time, text classification has become very important because of ever growing generation of data via different platform like website, blogs, social media, customer reviews and numerous others. It enables easy categorization and organization of vastly evolving amount data. This study would be beneficial in improving the performance of classification algorithm. SGD has been a versatile algorithm for training models but in textual data it faces challenge due to sparse nature of training data. An improved version of classification algorithm is expected both in terms of evaluation metrics and efficiency.

6. Scope of the Study

The study is focussed on proposing an improved version of Stochastic Gradient Descent Algorithm for training textual data. Modification in existing SGD algorithm is proposed. Improved SGD algorithm will be studied and compared with benchmark algorithms for evaluating improvement in working and convergence behaviour.

Study is not targeting to compare all different algorithms of classification. Most of basic hyperparameters will be kept constant to evaluate the effect of improved SGD.

7. Research Methodology

Our aim is to improve the stochastic Gradient Algorithm and evaluate the effect of new improved SGD in terms of accuracy and efficiency. A comparative analysis of improved SGD will also be done with benchmark algorithm for text classification i.e. Logistic Regression and Support Vector Machines.

7.1 Formulation of the algorithm

This section is involved with modification of existing SGD algorithm. In this research work a variant of stochastic gradient descent (SGD) approach will be used which is expected to give promising results. The improvement will be enforced by adding standard error in Adam and selecting values of weights and learning rates through grid search heuristics. These stated improvements will be then applied in updating rule. This would fasten rate of convergence algorithm considerably. This improvement is presented as Modified SGD algorithm. This modified version of SGD is expected to give better results compared to Adam, adaptive moment estimation with maximum (AdaMax), Nesterov-accelerated adaptive moment estimation (Nadam), AMSGrad and AdamSE algorithms. During calculation process, solution will converge iteratively. It is expected that Modified SGD algorithm will perform convergence in appreciably smaller number of steps. The results will be checked for enhanced convergence of proposed algorithm considering computational specifications. In conclusion, efficiency of Modified SGD will be validated through various applications (Su and Kek, 2021).

7.2 Data Set selection, Pre-processing, and feature extraction of the dataset

2 Different datasets are selected first one is reviews of consumer about different amazon products. This dataset has over 34,000 reviews of customer about products Fire TV, Kindle etc. This dataset is hosted in Kaggle platform. A very famous platform for Data Science competition. Other Dataset is Stock tweets for sentiment analysis and prediction. It is a collection of over 80K+ tweets for stock market sentiment analysis. It is also hosted on Kaggle platform. One of these datasets will be selected for further study based on suitability of conducting a comparative study between improved SGD and Benchmark algorithms for text classification tasks i.e. Logistic Regression & Support Vector Machines.

The dataset will be cleaned up using different techniques, so it becomes easier to understand. Also, this will help avoid any type of disturbance or error in training algorithm over the dataset. Which can possibly affect our results. Different techniques will be employed for cleaning the dataset like tokenization, removal of stop words, capitalization, stemming and lemmatization (Kowsari et al., 2019; Pimpalkar and Raj, 2020). Further feature representation of processed cleaned data will be done using TF-IDF technique. This technique is most widely used for feature extraction in sentiment analysis and text classification so this technique will provide good comparable benchmarks for comparison (Prasetijo et al., 2017; Diab, 2019; Hassan et al., 2020; Pimpalkar and Raj, 2020; Shah et al., 2020; Varshney et al., 2020; Gowri et al., 2021; Patel and Meehan, 2021; Al-Anzi, 2022b; a; Roshinta et al., 2022).

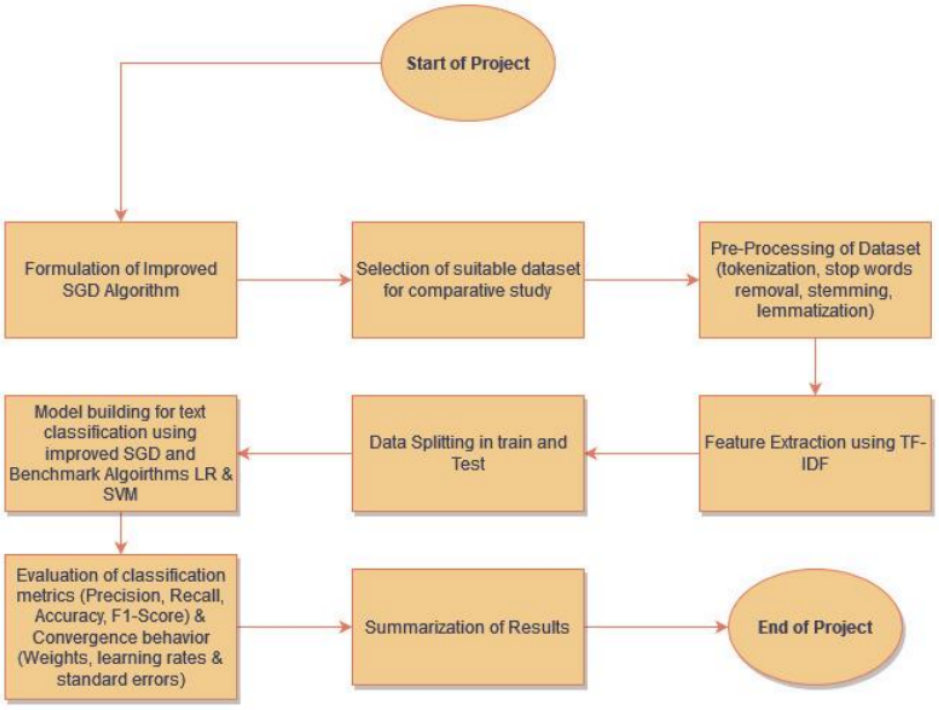
7.3 To apply basic and improved algorithm to train the classification model.

After feature extraction is done. Dataset will be split in training and test sets. Most general dataset split of 75% training and 25% test will be used. Further these datasets will be trained on improved SGD and Benchmark algorithms for text classification. After implementation of these different algorithms their training results will be captured for evaluation.

7.4 To evaluate the performance of improved SGD compared to earlier algorithms.

Classification algorithms are evaluated based on four most widely used metrics i.e., Precision, Recall, Accuracy and F1-Score. The performance of algorithms will be evaluated based on Precision, Recall, Accuracy and F1-score (Diab, 2019; Hassan et al., 2020; Pimpalkar and Raj, 2020; Patel and Meehan, 2021; Al-Anzi, 2022b). Weights, learning rates and standard errors will also be compared to get the idea about convergence behavior of algorithms. Results will be summarized using different table. Comparison charts between different parameters and algorithms will also be prepared for easy comprehension.

Following is the flowchart of Research Methodology for easy understanding the flow of work.



Flow chart of Research Methodology

8. Requirements Resources

Dataset size seems reasonable. Consumer Reviews of Amazon Products is 48.99 MB & Stock Tweets for Sentiment Analysis and Prediction is 18.94 MB in size. Therefore, following configuration of system will be sufficient. 16 GB RAM, i7 Processor with clock speed of 2.2 GHz or greater.

An equivalent AWS or Google Colab system can also be used.

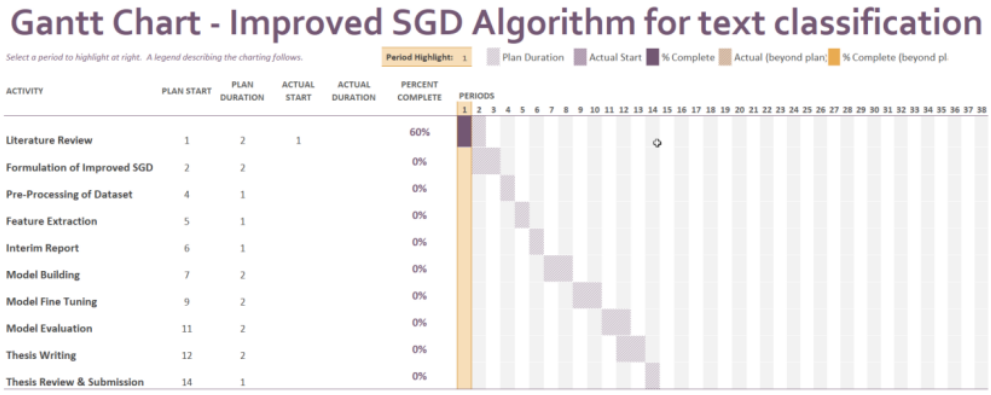
Other software resources (offline or online):

- Anaconda distribution package 2022.10
- Jupyter notebook v6.4.12
- python v3.10
- matplotlib v3.7.0
- seaborn v0.12.2
- Numpy v1.24.2

- NLTK v3.8
- SpaCy v3.4
- Gensim v3.2.0

9. Research Plan

Following Gantt Chart shows the research plan. 6 weeks available for Interim Report and 8 weeks for Final Thesis.



10. Risk Assessment & Mitigation

There are following risk and possible mitigation efforts are as follows:

- Data Loss or corruption - Data must be properly backed up on cloud and model development should be done on GitHub.
- Time Constraints – Project Scope should be properly adhered to avoid any issue of time constraints.
- Resources delay – Personal system can be used in meantime if AWS resources are delayed.

References

Al-Anzi, F.S., (2022a) An Effective Hybrid Stochastic Gradient Descent Arabic Sentiment Analysis with Partial-Order Microwords and Piecewise Differentiation. In: *2022 9th International Conference on Electrical and Electronics Engineering (ICEEE)*. IEEE, pp.408–411.

Al-Anzi, F.S., (2022b) An Effective Hybrid Stochastic Gradient Descent for Classification of Short Text Communication in E- Learning Environments. In: *2022 8th International Conference on Control, Decision and Information Technologies (CoDIT)*. [online] IEEE, pp.1096–1101. Available at: https://ieeexplore.ieee.org/document/9804138/.

Daud, S., Ullah, M., Rehman, A., Saba, T., Damaševičius, R. and Sattar, A., (2023) Topic Classification of Online News Articles Using Optimized Machine Learning Models. *Computers*, 121, p.16.

Diab, S., (2019) *Optimizing Stochastic Gradient Descent in Text Classification Based on Fine-Tuning Hyper-Parameters Approach. A Case Study on Automatic Classification of Global Terrorist Attacks*. [online] Available at: https://sites.google.com/site/ijcsis/.

Goswami, M. and Sabata, P., (2021) Evaluation of ML-Based Sentiment Analysis Techniques with Stochastic Gradient Descent and Logistic Regression. pp.153–163.

Gowri, S., Jenila, J., Reddy, B.S. and Sheela, M.A., (2021) Scrutinizing of Fake News using Machine Learning Techniques. In: *2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS)*. IEEE, pp.223–227.

Hassan, N., Gomaa, W., Khoriba, G. and Haggag, M., (2020) Credibility Detection in Twitter Using Word N-gram Analysis and Supervised Machine Learning Techniques. *International Journal of Intelligent Engineering and Systems*, 131, pp.291–300.

Kowsari, Jafari Meimandi, Heidarysafa, Mendu, Barnes and Brown, (2019) Text Classification Algorithms: A Survey. *Information*, 104, p.150.

Madhfar, M.A.H. and Al-Hagery, M.A.H., (2019) Arabic Text Classification: A Comparative Approach Using a Big Dataset. In: *2019 International Conference on Computer and Information Sciences (ICCIS)*. IEEE, pp.1–5.

Patel, A. and Meehan, K., (2021) Fake News Detection on Reddit Utilising CountVectorizer and Term Frequency-Inverse Document Frequency with Logistic Regression, MultinominalNB and Support Vector Machine. In: *2021 32nd Irish Signals and Systems Conference (ISSC)*. IEEE, pp.1–6.

Pimpalkar, A. and Raj, R.J.R., (2020) Evaluation of Tweets for Content Analysis Using Machine Learning Models. In: *2020 12th International Conference on Computational Intelligence and Communication Networks (CICN)*. IEEE, pp.454–459.

Prasetijo, A.B., Isnanto, R.R., Eridani, D., Soetrisno, Y.A.A., Arfan, M. and Sofwan, A., (2017) Hoax detection system on Indonesian news sites based on text classification using SVM and SGD. In: *2017 4th International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE)*. IEEE, pp.45–49.

Roshinta, T.A., Hartatik, Fauziyah, E.K., Dinata, I.F., Firdaus, N. and A'la, F.Y., (2022) A Comparison of Text Classification Methods: Towards Fake News Detection for Indonesian Websites. In: *2022 1st International Conference on Smart Technology, Applied Informatics, and Engineering (APICS)*. IEEE, pp.59–64.

Shah, K., Patel, H., Sanghvi, D. and Shah, M., (2020) A Comparative Analysis of Logistic Regression, Random Forest and KNN Models for the Text Classification. *Augmented Human Research*, 51, p.12.

Su, S.S.W. and Kek, S.L., (2021) An Improvement of Stochastic Gradient Descent Approach for Mean-Variance Portfolio Optimization Problem. *Journal of Mathematics*, 2021, pp.1–10.

Varshney, C.J., Sharma, A. and Yadav, D.P., (2020) Sentiment Analysis using Ensemble Classification Technique. In: *2020 IEEE Students Conference on Engineering & Systems (SCES)*. IEEE, pp.1–6.

Refer: Harvard Referencing Guide

Shailendra Kumar_Final Thesis_LJMU_Oct2023.pdf

ORIGINALITY REPORT

9%	1%	1%	9%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Liverpool John Moores University Student Paper	9%
2	"Proceedings of International Conference on Communication and Computational Technologies", Springer Science and Business Media LLC, 2023 Publication	1%

Exclude quotes	Off	Exclude matches	< 1%
Exclude bibliography	On		