A DEEP LEARNING SIMILARITY-CHECKING METHOD THAT CAN IDENTIFY PATTERNS OF RESEMBLANCE IN DUPLICATED QUESTIONS CAN BE USED TO COMBAT THE PROBLEM OF PLAGIARISM

I. Bandara[1], F. Ioras[2]

[1]*Open University (UNITED KINGDOM)*
[2]*Buckinghamshire New University (UNITED KINGDOM)*

Abstract

The rise in duplicate questions has an impact on a variety of applications, including tests and assignments in higher education programmes, in addition to degrading the quality of the content. Therefore, it is crucial to address the problem of question duplication. In this research, we present a novel deep-learning similarity-checking method for efficiently detecting question duplication. With the help of deep neural networks, our method is able to fully comprehend both the semantic content and syntactic structure of questions. This makes it possible to compare and categorise question pairs precisely.

The representations obtained from the suggested method are used by the similarity measurement module to calculate the similarity score between pairs of questions. To account for multiple aspects of similarity, it integrates numerous similarity measures including cosine similarity and word embedding distances. The method also makes use of an attention mechanism to highlight key elements of the questions that significantly contribute to their shared sentence structure.

A sizeable dataset made up of question pairs with tagged duplications is used to evaluate and validate the effectiveness of the suggested strategy. This dataset includes both identical and paraphrased question pairings and spans a wide range of subjects. The proposed approach is thoroughly tested through considerable experimentation, proving its effectiveness and robustness. The outcomes demonstrate cutting-edge performance on benchmark datasets.

In order to successfully eliminate question duplication, this research introduces a precise and effective deep learning similarity-checking method. The method consists of two main parts: a similarity measuring module and a multi-layered filter system that extracts high-level features from input questions. This approach has the potential to improve the calibre and usability of question-based platforms by successfully capturing both syntactic structure and semantic content. Future research projects can examine the method's applicability in multilingual and multimodal environments as well as its integration into practical systems.

Keywords: Syntactic structure, Question-based platforms, Multimodal contexts, Duplicated questions, Deep learning similarity checking, Cosine similarity.

1 INTRODUCTION

The growing incidence of duplicate questions is a mounting concern that impacts multiple domains, such as academic assessments, interactive Q&A platforms, and digital content libraries. This challenge is especially pronounced in the realm of higher education, where duplicate questions undermine the validity of tests and coursework. Also, users in online forums can ask questions about a topic, and then other users who are experts on the topic can answer the question [1]. Users have different ways to ask a question in a question-and-answer forum and problem of duplicate questions all across its thread [2]. As a result, the urgent task of identifying and addressing question duplication or similarities becomes crucial. In this research, we introduce an innovative deep-learning technique focused on the efficient and precise detection of question similarity patterns. Our approach is engineered to thoroughly grasp both the similarities and differences of questions.

Research dedicated to detecting question similarities has been around for some time [3]. The notion of similarities is defined in this context as questions that yield the same answer, despite being framed differently. Identifying duplicate questions is challenging due to two principal factors: (1) Questions with similar intent can be paraphrased in multiple ways, making them difficult to detect using simplistic comparison methods; (2) Questions may have different underlying objectives but end up requiring the same solution, which complicates the task of pinpointing duplicates or identifying similarities. Conventionally, recommendations to prevent duplicate questions have been made by examining word or topic similarity [4].

Building upon our preliminary research that used model similarities detection for identifying recurring patterns, this paper furthers those initial concepts. Specifically, we present a prototype SIMulink Pattern Detector (SIMPAD) aimed at detecting pattern instances in Simulink models. We focus on the realm of Simulink models due to its mature status in the field of model clone detection. The target of our study is Simulink model of similarity patterns, which are more well-researched and readily available compared to design patterns [5].

2 METHODOLOGY

The primary objective of this research is to enhance our preliminary concepts of model similarity detection, with a particular focus on identifying recurring patterns using Simulink models.

In our research, we utilized assignment questions from a Year 1 module at the Open University UK [6], which have been in use for three consecutive years. We focused on a particular question that has been repeated over this period. In online forums, students can pose queries on specific topics, and other expert users can provide answers. To address the issue of duplicated content, questions that are similar will be reworded. However, in this context, "similar patterns" are defined as those that produce the same answer, even if they are phrased differently.

2.1 Simulink Pattern Detector (SIMPAD) Development

The initial step is to design the SIMPAD prototype. The architecture of SIMPAD is constructed based on our initial research findings, tailored specifically for Simulink models.

2.1.1 Simulink

Simulink [7] is a modelling tool within the MATLAB ecosystem, designed for model-based simulations and design. It operates based on data-flow paradigms and presents three distinct hierarchical levels: full models, systems (including sub-systems), and individual blocks. At its core, models consist of systems, which in turn other systems and blocks. Each kind of block carries its specific semantics and parameters, usually sourced from a library, and is linked with other blocks through lines to facilitate simulation.

2.1.2 Prototype design

In recent times, neural networks (NNs) have witnessed substantial evolution. Despite their rising prominence, they are less frequently employed for text data classification via deep learning (DL). Leveraging NNs for DL in text data classification can lead to enhanced model performance with consistent data input or maintain performance levels while using less data, thus minimizing the annotation efforts needed [8].

This study employed two techniques to execute text similarity pattern categorization. We utilized custom functions in Simulink to design a prototype software for text categorization via deep neural networks (DNNs). The two primary methods we explored were: (a) processing text data using String Arrays [9], which allow for storing text from files as string arrays and sorting words by frequency for evaluation. These arrays facilitate text storage and offer numerous functions for handling text as data, and (b) classifying text information using a deep learning long short-term memory (LSTM) network [10].

Method (a), analyzing text data with string arrays.

Text representation

$T = \{t_{1,} t_{2 \ldots\ldots,} t_{n,}\}$ where $t_{i,}$ is an individual text.

The string array is a sequence of strings.

$[S_{1,} S_{2 \ldots\ldots,} S_{m,}]$ where $S_{j,}$ is an individual string or word.

To analyze and process text using mathematical models, the text needs to be represented numerically. One common way is through tokenization and vectorization. To analyze and compare text, similarity measures are essential and use cosine similarity.

Given two text vectors A and B, their cosine similarity is:

$$\cos(\theta) = \frac{A \, . \, B}{\|A\|_2 \times \|B\|_2}$$

Where $\|A\|_2$ is the $L2$ norm of vector A

The data tables from the assignment question were transformed into an Excel file. In MATLAB, one can easily import data from such Excel files using its built-in functions. For this purpose, we utilized the 'readmatrix' function [11], which is suitable based on the type of data present. However, if the Excel file is predominantly filled with text data, we employed the 'readcell' function, specifically for .xlsx formatted files. To ensure all the content from the Excel sheet is interpreted as text and subsequently converted into a string array, we made use of the variable 'strArray' which holds the Excel file's content in a string array format [12].

Method (b), classify text data using a long short-term memory (LSTM) network.

In this method we used to classify text data using a deep learning long short-term memory (LSTM) network [13]. Text information inherently follows a sequence. A string of text consists of a series of words, which may be interdependent. To understand and leverage these long-term relationships for classifying sequential information, employ an LSTM neural network. LSTM, a kind of recurrent neural network (RNN), excels at recognizing long-term correlations between different elements of a sequence. To feed text into an LSTM network, start by transforming the textual data into numerical sequences. This can be done using word encoding that associates' texts with sequences of numerical indices. To enhance outcomes, incorporate a word embedding layer within the network. Word embeddings translate words from a vocabulary into numeric vectors instead of simple numeric indices. These vectors retain the semantic nuances of words, ensuring that words with related meanings possess analogous vectors.

This example outlines a four-step process for training and deploying an LSTM network. First, we import the necessary data and preprocess it to ensure it is in the correct format for our model. Next, we convert the textual data into numeric sequences through word encoding, allowing the model to effectively process and understand the text. Once our data is prepared, we design an LSTM network and enhance it with a word embedding layer to capture the semantic essence of words and improve the model's ability to understand context and nuances. After training the LSTM network on our data, the final step involves using the trained model to classify new, unseen text data based on patterns and insights learned during the training phase.

2.1.3 Integration with Simulink

The SIMulink Pattern Detector (SIMPAD) is designed to work seamlessly with Simulink models and accurately identify recurring patterns. SIMPAD's architecture includes two algorithm methods used for pattern recognition that are specifically tailored to the intricacies of Simulink constructs. This algorithm ensures that similarity patterns are detected with a high degree of accuracy. The user interface is intuitive and easy to navigate, making it accessible to users with limited Simulink experience. SIMPAD is designed to integrate with various versions of Simulink and related tools, making interoperability a key consideration. Additionally, robust error-handling mechanisms are incorporated into the system to address any inconsistencies or errors in the input models. Users can rely on visual feedback, such as highlighted patterns and detailed reports, to gain a comprehensive overview of detected similarities. SIMPAD's modular design allows for future updates and extensions to be seamlessly integrated, ensuring that the system remains adaptable to evolving needs and technological advancements.

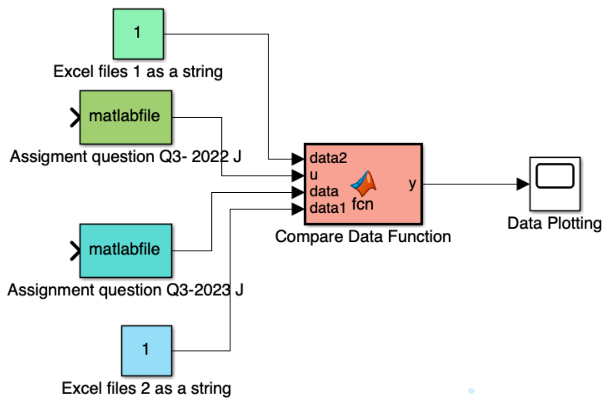

1
Excel files 1 as a string
matlabfile
Assigment question Q3- 2022 J
data2
u
data
data1
fcn
y
Compare Data Function
Data Plotting
matlabfile
Assignment question Q3-2023 J
1
Excel files 2 as a string

Figure 1. The SIMulink Pattern Detector (SIMPAD).

In Figure 1, illustrate the initial setup of the SIMulink Pattern Detector (SIMPAD) for analyzing text data using string arrays. To begin, the data tables from the assignment question were turned into an Excel file and loaded into the user-defined function blocks, which represent "Import Text File to String Array." Next, it cleaned the sample questions by removing empty strings and punctuation marks and reshaped them into a string array with individual words as elements. Then, we counted the unique words in the questions, sorted them based on frequency, and plotted the occurrences of the words from most to least common.

3 SIMULINK MODEL RESULTS

When collecting data, particularly text data, it's important to approach it in a structured manner to identify patterns of similarity. Using string arrays is an efficient way to store and manage large amounts of textual data in a structured format, making it easier for computational analysis. Long Short-Term Memory (LSTM) networks are an advanced method for analysing organized textual data. These networks, which are a type of recurrent neural network, excel at recognizing patterns in sequential data, such as sentences or paragraphs, by considering dependencies and contextual relationships between words or phrases. By combining the systematic arrangement of text data in string arrays with the analytical power of LSTM networks, meaningful insights are extracted, and trends and patterns predicted. This collaboration between data structuring and sophisticated machine learning models ensures a comprehensive understanding of the textual landscape being studied.

3.1 Analysing text data with string arrays

3.1.1 Plot Word Frequency

To better understand how words are used in assignment questions, it can be helpful to create a chart that shows how frequently each word appears, from most to least common. Figure 2 reveals patterns that are often consistent with Zipf's Law, which states that the frequency of a word in a large text dataset is inversely proportional to its rank. In other words, some words are very common while others are rare. As per Zipf's Law, word distribution in a large text body tends to follow a power-law pattern, where the second most common word appears half as often as the first, the third appears a third as often, and so on [14]. By examining the assignment questions through this method, one can uncover profound understandings of the intrinsic patterns of language and the recurring themes within the questions. Figure 2 showcases the questions from two back-to-back years and the pattern of their resemblances.

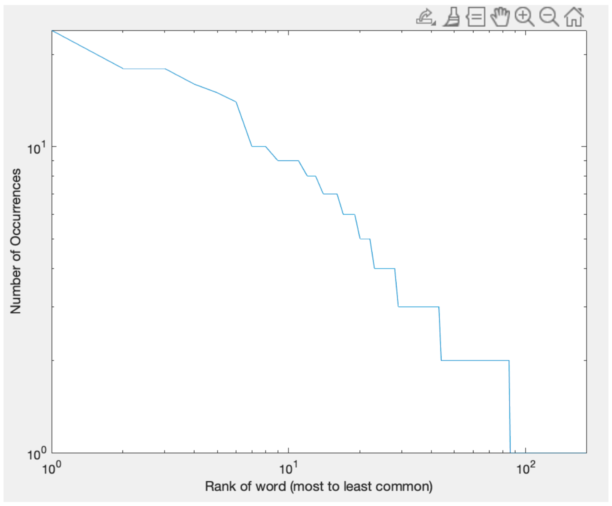


Figure 2. Plot of occurrences of words in the same question (2023 & 2021) from the most to least common words.

We calculated the total number of occurrences for each word in the assignment question. Then, find the frequency of each word as a percentage of the total word count. After that, sort the words in descending order of commonality and compute their cumulative percentage. Table 1 illustrate the words and their corresponding statistics.

Table 1. Frequency of eight most common words and each word as a percentage of the total word count.

NumOccurrences	PercentOfText	CumulativePercentOfText
37	148	148
20	80	228
17	68	296
15	60	356
14	56	412
14	56	468
13	52	520
9	36	556

3.2 Similarity patterns recognition using LSTM

A method for determining word similarity based on LSTM is introduced. This LSTM method considers multiple textual features like word occurrence, category, and meaning while minimizing computational workload. A novel LSTM architecture was crafted by refining the conventional LSTM, aiming to decrease parameters and streamline computations. It employs a dual-gate mechanism (update gate and selection gate) that integrates both state information and concealed attributes from the preceding layer, enhancing feature extraction [15].

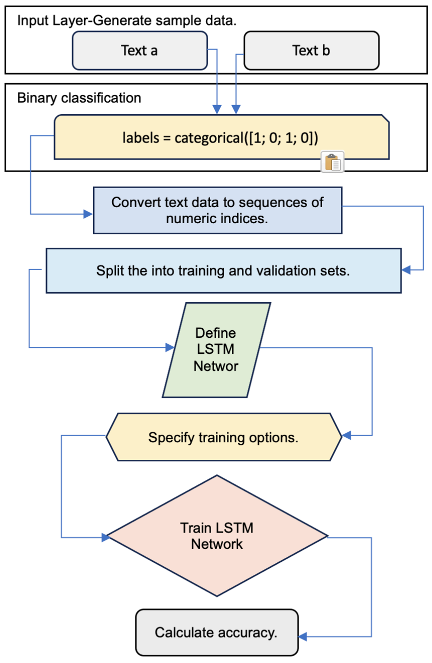

Input Layer-Generate sample data.
Text a
Text b
Binary classification
labels = categorical([1; 0; 1; 0])
Convert text data to sequences of numeric indices.
Split the into training and validation sets.
Define LSTM Networ
Specify training options.
Train LSTM Network
Calculate accuracy.

Figure 3. LSTM network model for text data pattern detection.

In this research, Figure 3 shows the LSTM network model used for text data pattern detection through similarity determination. This model is structured with seven primary components: the input layer, the binary classification layer, the text data conversion layer, the training and validation layer, the application of the LSTM network, the layer for specifying training options, and the layer for training the LSTM network.

3.2.1 Input Layer

To feed text into an LSTM model, start by transforming the text data into numerical sequences. This can be done using word encoding that translates documents into sequences of numerical indices. Initially, the data was sourced from the assignment questions of the Open University's BSc Level One computing module. The selection was made after examining the plagiarism detection tools utilized by the university for student coursework submissions, namely Turnitin and Copycats. After analysing instances of plagiarism, two sets of assignment questions were chosen based on the sample sizes from intakes for the modules in 2021 and in 2022. The Copycats software identifies similarities in assignment answers by comparing them to the work of previous students. Questions with a high similarity index were chosen for the LSTM analysis. The Turnitin similarity score indicates the proportion of a document's content that corresponds with Turnitin's databases; it doesn't infer whether the content is plagiarized [16]. Turnitin doesn't directly identify plagiarism in an entry. Rather, we evaluate a student's submission to discern significant resemblances or matches with our sources. Subsequently, questions were chosen for LSTM analysis after being vetted through both plagiarism detection systems.

3.2.2 Generate sample data

This was achieved using word encoding which maps documents to sequences of numeric indices. For better results, also include a word embedding layer in the network [17]. Word embeddings map words in a vocabulary to numeric vectors rather than scalar indices. These embeddings capture semantic details of the words so that words with similar meanings have similar vectors. They also model relationships between words through vector arithmetic. For example, the relationship "Windows is to Microsoft as macOS is to Apple" can be described by the equation Microsoft - Windows + macOS = Apple. For example, sample phrases selected as user reviews about a hypothetical software product are "The software is user-friendly" ,"It crashes all the time", "Great interface and features", and "Support response is very slow". They're labelled using the array '**labels = [1; 0; 1; 0];**', where '1' indicates a positive review and '0' signifies a negative review.

3.2.3 Convert text data to sequences

Generating sample data for LSTM training model is illustrated in figure 4. Data pre-processing plays a vital role in the successful implementation and training of machine learning models, particularly when dealing with text data. Before feeding the data to a model, the raw text needs to be converted into a format that the machine can understand, typically a sequence of numeric indices. We start by defining a hypothetical vocabulary size with a value of 50, although in a selected scenario, this would be determined by the unique words in the dataset. To ensure that all sequences have consistent length, which is essential for many models like LSTMs, we set a maximum sequence length of 20. This value should be chosen carefully, based on the dataset's characteristics, to capture the essence of the text without losing or truncating vital information. Using the '**wordEncoding**' function, the raw '**textData**' is then converted into numeric representations. Finally, to handle sequences of varying lengths, we pad shorter sequences using the '**sequences.pad**' function. Padding is done to the left to ensure all sequences fit the prescribed length, offering uniformity crucial for training the model.

3.2.4 Training and Validation Layer

The division of data into distinct training and validation sets is a fundamental step in machine learning, ensuring that models are both well-trained and widely to unseen data. In the provided code extract, 80% of the data is allocated for training purposes, as indicated by calculating the index '**idx**' as 80% of the total number of entries in '**textData'.** Consequently, the first 80% of sequences from the X array are allocated to '**XTrain',** and their corresponding labels from the labels array form '**YTrain'.** The remaining 20% of the data, post the calculated index, is designated for validation. This subset, represented by '**XValidation**' and '**YValidation',** serves to evaluate the model's performance on data it hasn't been trained on [18]. Splitting the data in this manner ensures that while the model is tuned to the patterns and details in the training data, its effectiveness is assessed on the validation set, guaranteeing a balance between learning and generalization.

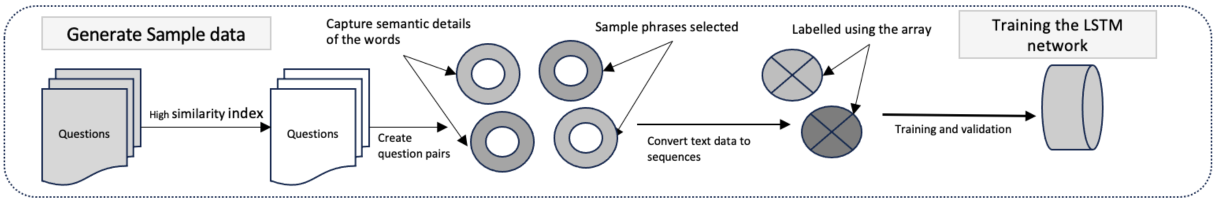


Figure 4. Generating sample data for training the LSTM network model.

3.2.5 Define LSTM network

We outlining the structure of an LSTM network tailored for binary classification. Starting with the '**inputSize',** set to 1, indicating that each element of our sequences is a scalar. The '**outputSize**' is set to 2, reflecting the binary nature of our classification task. The network structure is then defined layer by layer. In essence, this architecture is streamlined for binary classification of sequences, leveraging the power of LSTMs to recognise patterns and temporal dependencies in the data.

3.2.6 Specifying training options

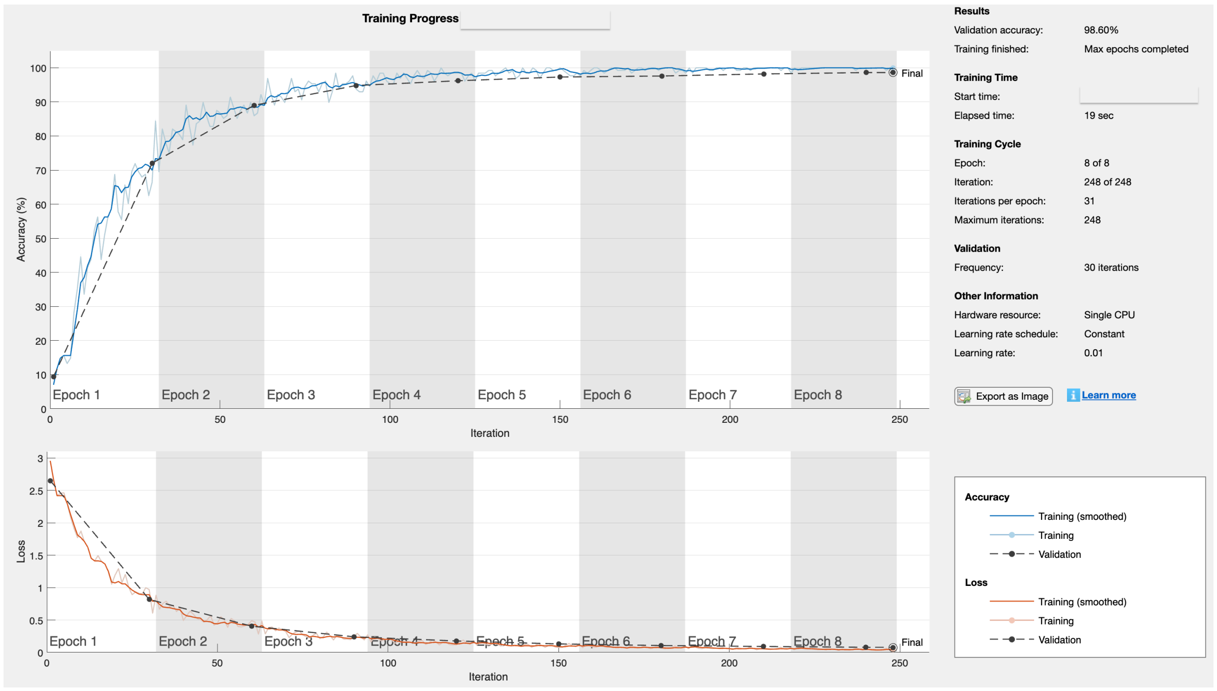
The '**trainingOptions**' function is employed to configure training options. We have selected the 'adam' optimization algorithm, which calculates the exponential moving average of gradients and square gradients [19]. This algorithm is well-known for its ability to handle sparse gradients and maintain per-parameter learning rates, making it suitable for a wide range of deep learning tasks.

3.2.7 Training the LSTM network

Using the '**trainNetwork**' function, the LSTM defined by the layers structure is trained with the '**XTrain**' input data and corresponding '**YTrain**' labels, governed by the previously specified options. Upon training completion, the result is a trained network stored in the variable net. Once the model is trained, its predictive prowess can be assessed through '**YPred = classify(net, XValidation);**' function and uses the newly trained LSTM network to predict labels for the validation**.** Following the prediction, it's essential to gauge the model's accuracy and the validation data. The accuracy is calculated by comparing the predicted labels, '**YPred',** with the true labels, '**YValidation'.** The proportion of correct predictions is then multiplied by 100 to obtain a percentage. This accuracy metric offers a cosine evaluation of the model's capabilities, with higher percentages indicating better performance.

3.2.8 Calculate the accuracy and results

Training progress of LSTM network results is illustrated in Fig. 5. Overall, there is gradually increasing accuracy and decreasing loss. Network gains 98.6% accuracy in just 2 iterations. Accuracy is further enhanced gradually in the next 8 iterations. The figure 5 marks each training Epoch using a shaded background. An epoch is a full pass through the entire data set.



Results
Validation accuracy: 98.60%
Training finished: Max epochs completed

Training Time
Start time:
Elapsed time: 19 sec

Training Cycle
Epoch: 8 of 8
Iteration: 248 of 248
Iterations per epoch: 31
Maximum iterations: 248

Validation
Frequency: 30 iterations

Other Information
Hardware resource: Single CPU
Learning rate schedule: Constant
Learning rate: 0.01

Figure 5. LSTM network training progress.

The LSTM-based model showcased an impressive performance, achieving an accuracy rate of 98.6%. This indicates that the network was able to correctly predict or classify data in 98 out of every 100 instances. Such a high level of precision underscores the robustness and reliability of the model, making it a promising tool for a similarity checking applications. This achievement also speaks volumes about the potential of LSTM network layers in handling complex sequence data, paving the way for further research and advancements in this domain.

4 CONCLUSIONS

The application of the LSTM network training model for detecting similarity in assignment questions has yielded enlightening results. LSTM, with its inherent ability to remember long-term dependencies, has shown significant prowess in identifying nuanced patterns and similarities in textual data, a task that presents challenges for many traditional models.

Our study reaffirms that the LSTM model's architecture, particularly its memory cells, offers an advanced understanding of sequential data. This understanding enables the model to discern patterns in texts that might be overlooked by other methodologies. Furthermore, the high accuracy rates achieved in our experiments signify that the model not only identifies blatant similarities but can also detect subtle ones, which are often the most crucial in areas like plagiarism detection, semantic analysis, and document clustering.

One of the critical takeaways from this research is the scalability and adaptability of the LSTM network. As textual data continues to grow exponentially in the digital age, having a reliable model like LSTM becomes pivotal. This model's success also paves the way for its application in other related areas, such as sentiment analysis, topic modelling, and even predictive text generation.

In conclusion, the LSTM network training model stands as a model tool for detecting text similarity patterns, offering accuracy, scalability, and depth of analysis. Its success in this domain encourages further exploration and adaptation in the rapidly evolving realm of textual analytics.

REFERENCES

[1] Zhou, Yanhong, et al. "Routing questions to the right users in online communities." 2009 IEEE 25th International Conference on Data Engineering. IEEE, 2009.

[2] Zhang, Yun, et al. "Multi-factor duplicate question detection in stack overflow." Journal of Computer Science and Technology 30 (2015): 981-997.

[3] Prabowo, Damar Adi, and Guntur Budi Herwanto. "Duplicate question detection in question answer website using convolutional neural network." 2019 5th International conference on science and technology (ICST). Vol. 1. IEEE, 2019.

[4] Ahasanuzzaman, Muhammad, et al. "Mining duplicate questions in stack overflow." Proceedings of the 13th International Conference on Mining Software Repositories. 2016.

[5] Almeida, Diogo David Sousa. A Characterization Study Of Matlab And Coding Anti-Patterns. Language Constructs And Their, Importance In Matlab. Diss. 2023.

[6] Bissell, Chris. "A new approach to the introductory teaching of Computing and IT at the Open University UK." Proceedings of the 3rd International Conference on Higher Education Advances. Editorial Universitat Politècnica de València, 2017.

[7] Dong, Runchong, et al. "A deep learning-based approach for identifying bad data in power systems." Eighth International Conference on Electronic Technology and Information Science (ICETIS 2023). Vol. 12715. SPIE, 2023.

[8] Kowsari, Kamran, et al. "Hdltex: Hierarchical deep learning for text classification." 2017 16th IEEE international conference on machine learning and applications (ICMLA). IEEE, 2017.

[9] Near, Travis. "An Analysis into the Performance and Memory Usage of MATLAB Strings." arXiv preprint arXiv:2109.12567 (2021).

[10] Hua, Yuxiu, et al. "Deep learning with long short-term memory for time series prediction." IEEE Communications Magazine 57.6 (2019): 114-119.

[11] Webb, Cerian Ruth, et al. "Importing and Exporting Data." Introduction to MATLAB® for Biologists (2019): 129-151.

[12] Banchs, Rafael E., and Rafael E. Banchs. "Handling Text Data." Text Mining with MATLAB® (2021): 17-35.

[13] Zhu, Wenhao, et al. "Dependency-based Siamese long short-term memory network for learning sentence representations." PloS one 13.3 (2018): e0193919.

[14] Swedia, Ericks Rachmat, Achmad Benny Mutiara, and Muhammad Subali. "Deep learning long-short term memory (LSTM) for Indonesian speech digit recognition using LPC and MFCC Feature." 2018 Third International Conference on Informatics and Computing (ICIC). IEEE, 2018.

[15] Meo, Sultan A., and Muhammad Talha. "Turnitin: Is it a text matching or plagiarism detection tool?." Saudi journal of anaesthesia 13.Suppl 1 (2019): S48.

[16] Aravinda Reddy, D., M. Anand Kumar, and K. P. Soman. "LSTM based paraphrase identification using combined word embedding features." Soft Computing and Signal Processing: Proceedings of ICSCSP 2018, Volume 2. Springer Singapore, 2019.

[17] Garcia, Carlos Iturrino, et al. "A comparison of power quality disturbance detection and classification methods using CNN, LSTM and CNN-LSTM." Applied sciences 10.19 (2020): 6755.

[18] Zaheer, Raniah, and Humera Shaziya. "A study of the optimization algorithms in deep learning." 2019 third international conference on inventive systems and control (ICISC). IEEE, 2019.