KSÜ Mühendislik Bilimleri Dergisi, 27(4), 2024
Araştırma Makalesi

KSU J Eng Sci, 27(4), 2024
Research Article

Kahramanmaras Sutcu Imam University
Journal of Engineering Sciences

Geliş Tarihi : 29.04.2024
Kabul Tarihi : 25.07.2024

Received Date : 29.04.2024
Accepted Date : 25.07.2024

ENHANCING DEEP LEARNING PERFORMANCE THROUGH A GENETIC ALGORITHM-ENHANCED APPROACH: FOCUSING ON LSTM

GENETİK ALGORİTMA DESTEKLİ BİR YAKLAŞIM İLE DERİN ÖĞRENME PERFORMANSININ GELİŞTİRİLMESİ: LSTM ODAKLI

Tarık Üveys ŞEN[1] (ORCID: 0009-0000-0297-6064)
Gökhan BAKAL[1*] (ORCID: 0000-0003-2897-3894)

[1] Abdullah Gul University, Computer Engineering Department, Kayseri, Türkiye

*Sorumlu Yazar / Corresponding Author: Gökhan BAKAL, gokhan.bakal@agu.edu.tr

ABSTRACT

Deep learning has shown remarkable success in various applications, such as image classification, natural language processing, and speech recognition. However, training deep neural networks is challenging due to their complex architecture and the number of parameters required. Genetic algorithms have been proposed as an alternative optimization technique for deep learning, offering an efficient alternative way to find an optimal set of network parameters that minimize the objective function. In this paper, we propose a novel approach integrating genetic algorithms with deep learning, specifically LSTM models, to enhance performance. Our method optimizes crucial hyper-parameters including learning rate, batch size, neuron count per layer, and layer depth through genetic algorithms. Additionally, we conduct a comprehensive analysis of how genetic algorithm parameters influence the optimization process and illustrate their significant impact on improving LSTM model performance. Overall, the presented method provides a powerful mechanism for improving the performance of deep neural networks, and; thus, we believe that it has significant potential for future applications in the artificial intelligence discipline.

Keywords: genetic algorithm, hyper-parameter optimization, deep learning, lstm

ÖZET

Derin öğrenme, görüntü sınıflandırma, doğal dil işleme ve konuşma tanıma gibi çeşitli uygulamalarda dikkat çekici başarılar elde etmiştir. Ancak, derin sinir ağlarını eğitmek, karmaşık mimarileri ve gereken parametre sayısı nedeniyle zorlu bir süreçtir. Genetik algoritmalar, derin öğrenme için alternatif bir optimizasyon teknik olarak önerilmiştir ve optimal bir ağ parametre setini minimize eden bir amaç fonksiyonu bulmak için etkili bir alternatif yöntem sunar. Bu makalede, derin öğrenme ile genetik algoritmaları entegre eden, özellikle LSTM modellerini kullanarak performansı artırmayı amaçlayan yeni bir yaklaşım öneriyoruz. Yöntemimiz, genetik algoritmalar aracılığıyla öğrenme hızı, grup boyutu, katman başına nöron sayısı ve katman derinliği gibi kritik hiper-parametreleri optimize eder. Ayrıca, genetik algoritma parametrelerinin optimizasyon sürecini nasıl etkilediğine dair kapsamlı bir analiz yaparak, LSTM model performansını iyileştirmedeki önemli etkilerini gösteriyoruz. Genel olarak, sunulan yöntem, derin sinir ağlarının performansını artırmak için güçlü bir mekanizma sunmakta olup bu nedenle yapay zekâ disiplininde gelecekteki uygulamalar için önemli bir potansiyele sahip olduğuna inanıyoruz.

Anahtar Kelimeler: genetik algoritma, hiper-parametre optimizasyonu, derin öğrenme, lstm

ToCite: ŞEN, T. Ü., & BAKAL, G., (2024). ENHANCING DEEP LEARNING PERFORMANCE THROUGH A GENETIC ALGORITHM-ENHANCED APPROACH: FOCUSING ON LSTM. *Kahramanmaraş Sütçü İmam Üniversitesi Mühendislik Bilimleri Dergisi*, 27(4), 1350-1360.

INTRODUCTION

Machine learning (ML), as a subfield of artificial intelligence, has been boosted by deep learning approaches, which have enabled remarkable performance improvement in distinct application domains such as image recognition, natural language processing, and speech recognition. However, training deep neural networks is a complex process due to having intricate architectures and learning a variety of parameters (Şen and Bakal, 2023). One of the most challenging yet critical aspects here is selecting the optimal hyper-parameters, such as the learning rate, batch size, number of layers, and number of neurons in each layer. The learning rate controls how much to change the model in response to the estimated error each time the model weights are updated, while the batch size determines the number of samples to be used in each training iteration. The number of layers and neurons per layer defines the depth and width of the neural network, respectively, crucially influencing the model's capacity to learn complex patterns and achieve state-of-the-art results. Several optimization techniques, such as stochastic gradient descent and Adam optimizer, have been proposed for optimizing hyper-parameters in deep learning models. However, these techniques are often too sensitive to the choice of hyper-parameters themselves, and finding their optimal set can be a time-consuming task for a given problem. The primary reason for genetic algorithm usage instead of commonly used grid search tuning is to utilize the genetic algorithms' power for hyper-parameter tuning. This claim is because genetic algorithms can offer advantages over the grid search approach. These advantages include balanced exploration and exploitation, the capability to handle complex search spaces, adaptability to non-gradient scenarios, parallelizability, robustness to local optima, and flexibility to adapt to changing landscapes. Consequently, genetic algorithms can provide an efficient and effective strategy for optimizing configurations in hyper-parameter tuning tasks.

The proposed study investigated the changes in fitness levels across three generations within a particular population by addressing the challenge of fitness (Kramer and Kramer, 2017). The obtained results suggest that the optimized model varies significantly with the percentage of data used for the training process. For 5% of the dataset, there was an observed decrease in fitness of approximately 0.47%. In contrast, a 10% dataset was associated with an increase in fitness of 4.2%, and a 20% dataset resulted in a fitness increase of 3.45%. Our fitness score, accuracy, measures the model's predictive accuracy, with higher values indicating better performance. Since our problem is a maximizing problem, the goal is to maximize the accuracy score to achieve optimal model performance.

To effectively manage computational resources while still maintaining scientific rigor, a subset approach was adopted for data analysis. This approach involved utilizing varying percentages of the dataset (5%, 10%, and 20%) to evaluate its impact on observed outcomes, specifically focusing on fitness metrics. The rationale behind this strategy lies in its ability to provide insights into the relationship between dataset size and analytical outcomes, allowing for a nuanced understanding of the dataset's characteristics without overwhelming computational resources.

Results indicate that, despite the reduced dataset sizes, meaningful trends in fitness were still discernible. For instance, the analysis revealed that a 5% dataset was associated with a slight decrease in fitness of approximately 0.47%, suggesting potential limitations in capturing finer details due to the reduced sample size. Conversely, both the 10% and 20% datasets exhibited notable increases in fitness, showcasing the potential benefits of larger dataset size in enhancing analytical outcomes. These findings underscore the importance of carefully selecting dataset sizes relative to available computational resources, ensuring that analytical objectives are met while mitigating resource constraints.

Overall, this study highlights the critical role of genetic algorithms for hyperparameter selection to achieve an optimal deep learning performance and expresses the need for further research to address this challenge. The obtained outcomes also demonstrate valuable insights into the impact of dataset size on fitness levels in evolutionary dynamics.

In the subsequent sections, we first discuss the general background and some related work in Section 2. We provide details regarding the dataset used in our experiments in Section 3. Then, we present thorough explanations of the models we constructed in Section 4, including feature extraction, data preprocessing, and technical aspects of genetic algorithms, such as selection and mutation. Section 5 demonstrates the results and discussion from unique angles. Ultimately, we conclude with critical remarks in Section 6.

BACKGROUND

In this section, we provide a comprehensive overview of genetic algorithms (GAs). We begin by discussing the basic principles of GAs, including the representation of solutions, the fitness function, and the genetic operators. We then discuss the different types of GAs and the factors that affect their performance. Finally, we discuss related literature in the parallel domain.

Genetic Algorithms

Genetic algorithms (GAs) are a class of optimization algorithms inspired by the process of natural selection and genetics concepts. GAs are broadly employed to solve widespread optimization problems, including but not limited to engineering design, finance, production systems, and many others. The idea of using the genetics principles for optimization was first proposed by Holland (1992a, 1992b) in the 1960s. Since then, GAs have been developed and refined by many researchers and are now known as one of the most powerful optimization techniques available.

The basic idea behind GAs is to simulate the natural selection process for solving problems in computational studies. In a GA, a population of candidate solutions is evolved over many generations using three basic operations: selection, crossover, and mutation (Katoch, Chauhan, and Kumar, 2021; Kramer and Kramer, 2017; Lambora, Gupta, and Chopra, 2019). Selection is the process by which fitter individuals are selected from the current population to form the basis of the next generation. Crossover involves exchanging genetic information between two individuals to create new offspring. Mutation introduces random changes in the genetic makeup of individuals to help explore distinct regions of the solution space.

GAs offer several unique advantages compared to traditional methods of problem-solving. One of their key benefits is their ability to function without requiring imitative information, which is often not even available in real-life problems. In addition, genetic algorithms are more effective and efficient compared to primitive methods, possessing well-aligned capabilities that can optimize both continuous and distinct functions and multi-purpose problems. Another critical advantage is that genetic algorithms aim to provide not just one but the best possible solutions, ensuring a comprehensive exploration and resolution of the corresponding problem. Additionally, genetic algorithms continually improve the accuracy and refinement of their solutions over time, providing satisfactory answers to complex problems in various fields, such as engineering, finance, and computer science, particularly in cases of large search spaces with multiple parameters (Hajireza, Darabi, and Najafi Moghaddam, 2023; Lambora et al., 2019).

In summary, genetic algorithms are a subclass of stochastic optimization techniques extensively studied and applied in various fields (Haldurai, Madhubala, and Rajalakshmi, 2016). GAs are inspired by the natural selection process, in which individuals with better traits are more likely to survive and reproduce. GAs use this principle to search for optimal solutions to problems by iteratively generating and evaluating a population of candidate solutions. The fundamental steps of a GA are as follows:

1. Initialize a population of candidate resolutions.
2. Evaluate the fitness of each solution.
3. Select the best solutions to form a new population.
4. Generate new solutions by recombining and mutating the best solutions.
5. Repeat steps 2-4 until yielding a stopping criterion.

Since the utilization of GAs is practical, researchers intensively employ them in solving various optimization problems, including scheduling, routing, and machine learning (Peng et al., 2019; Zivkovic et al., 2021).

DATASET DETAILS

In data analysis studies, the accuracy and interpretability of the results heavily depend on the quality and relevance of the dataset. This section outlines the collection process and key statistics of the dataset. The data instances were collected and generated as a Drug Review Dataset by Kallumadi and Grer (2018) from the drugs.com website. Each data sample consists of a personal opinion about a drug taken for a condition, corresponding drug and disease names, and a rating score between 1 and 10. The original data set contains 215,063 data records and the distribution of the records is visually represented in Figure 1.

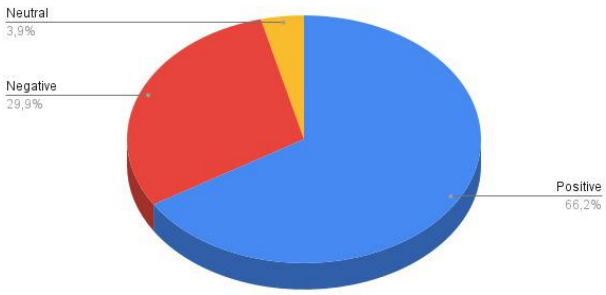

Figure 1. Class Distribution of Drug Review Examples in the Dataset

We categorized them into three groups based on their "rating" value, resulting in 142,306 annotated positive-class instances, 64,295 annotated negative-class instances, and 8,462 labeled neutral-class examples. The records were annotated as positive if the rating score was greater than or equal to 7. Conversely, the records were labeled as negative if the rating value was less than or equal to 5. The rest of the records sharing a rating value of 5 were annotated as neutral.

METHODOLOGY

This section provides a comprehensive overview of our proposed approach. In particular, we covered the principal details of data pre-processing and the structure of a deep learning model we built through the genetic algorithms' utilization in detail, as well as the fundamental concepts in genetic algorithms in the following subsections. This work adheres to the ethical standards and principles governing scientific research and publication.

Data Preprocessing and Partitioning

In our investigation, we chose not to perform any text cleaning or n-gram separation. Instead, we used the Tokenizer function available in the Keras library (Chollet et al., 2015) to convert the textual review column into an array. This strategy resulted in the average, maximum, and minimum lengths of the text instances being 89, 1992, and 0, respectively. Based on these length values, we set the maximum length for the padding sequence to 1000. As mentioned in Section 3, we utilized the rating scores to establish the class distribution for categorizing the unprocessed drug reviews in the dataset. Then, we exploited the label-encoder package from the Sklearn library (Pedregosa et al., 2011) to convert the resulting class distribution into multi-class digital data. Here, the obtained annotated data will be employed to train machine learning models that can predict the sentiment of the drug review instances in the test set. Lastly, we divided the data set into three separate collections, with 70% assigned to training, 20% to testing, and 10% to validation.

Selection in Genetic Algorithms

In genetic algorithms, the selection concept chooses individuals from a population to participate in reproduction processes. The main goal is to ensure that the best individuals in the population are more likely to be selected for reproduction so that their genes can be transferred to the next generation. In our experiments, we employ a selection operator based on fitness proportionate selection to determine the parents for the next generation, commonly referred to as roulette wheel selection (Jebari, 2013; Katoch et al., 2021; Lipowski and Lipowska, 2012; Rathore and Rathore, 2016). This approach gives individuals with higher fitness scores a greater likelihood of being selected as parents. Then, we normalize the fitness scores of all individuals such that they sum up to one for satisfying proportionality. Subsequently, we randomly selected two parents from the population based on their normalized fitness scores. Apart from the proportionate fitness selection, there are several selection methods available in genetic algorithms, such as those mentioned below:

- *Tournament selection*: In this method, a few individuals are randomly selected from the population, and chosen the best among them as a parent for the next generation. This process is repeated to select the second parent (Fang and Li, 2010; Greenstein, Elsey, and Hutchison, 2023; Jebari, 2013; Katoch et al., 2021; Rathore and Rathore, 2016).

- *Rank-based selection*: This approach assigns ranks to individuals based on their fitness scores and selects parents based on their ranks rather than their fitness scores (Jebari, 2013; Katoch et al., 2021; Pencheva, Atanassov, and Shannon, 2009; Rathore and Rathore, 2016; Zheng and Wen, 2023).
- *Stochastic universal sampling*: This strategy assigns probabilities to candidate solutions based on fitness and selects them using a roulette wheel approach. This method promotes diversity while favoring fitter solutions (Jebari, 2013; Katoch et al., 2021; Pencheva et al., 2009).

Crossover in Genetic Algorithms

The crossover is a genetic operator that combines the genetic information of two parents to generate new offspring. It is one way to stochastically generate new solutions from an existing population and is naturally analogous to the crossover that happens during sexual reproduction in biology. In this study, we utilize a one-point crossover (Katoch et al., 2021; Pachuau, Roy, and Kumar Saha, 2021; Rathore and Rathore, 2016) operator that involves choosing a random crossover point and exchanging the values of the parameters before and after that point between two parents to generate two offspring. Specifically, we select a random crossover point for each parent pair and swap the values of the parameters before and after that point. We add the offspring produced by each parent pair to the next generation. We repeat this process until we obtain the desired number of offspring. Additionally, other crossover techniques used in genetic algorithms are listed below:

- *Multi-point crossover*: This approach involves selecting multi-crossover points instead of one and exchanging the parameter values between the parents within that segment (Katoch et al., 2021; Pachuau et al., 2021; Rathore and Rathore, 2016).
- *Uniform crossover*: In this method, each parameter value of the offspring is randomly selected from either parent with equal probability (Katoch et al., 2021; Pachuau et al., 2021; Rathore and Rathore, 2016).
- *Arithmetic crossover*: This operator considers taking a weighted average of the parameter values of the parents to generate the offspring (Kora and Yadlapalli, 2017; Pachuau et al., 2021; Rathore and Rathore, 2016).

Mutation in Genetic Algorithms

The mutation is a genetic operator that introduces random changes to an individual's genetic structure (Katoch et al., 2021; Rathore and Rathore, 2016). It is analogous to the usual mutation process that happens during biological reproduction. Technically, it is advantageous to introduce new genetic characteristics into the population, which can help to improve the population's diversity and prevent it from becoming stagnant. In this study, the mutation probability was 0.5, so each parameter is randomly mutated for each offspring in the current generation. If a parameter is selected for mutation, a new random value is generated for that parameter within its search space limits. By incorporating this technique, the population can explore new and unexplored search space areas beyond those generated by the crossover operator. This process can potentially lead to the identification of more optimal solutions and the avoidance of local optima. There are various mutation techniques applicable in genetic algorithms, including:

- *Gaussian mutation*: This method adds a random value drawn from a Gaussian distribution to the parameter value (Rathore and Rathore, 2016; Yan, 2023).
- *Boundary mutation*: This operator perturbs the parameter value by a fixed amount and ensures that the new value remains within the search space limits (Katoch et al., 2021; Rathore and Rathore, 2016).
- *Non-uniform mutation*: This approach involves applying a varying amount of perturbation to the parameter value depending on the generation number or the fitness score of the individual (Katoch et al., 2021; Rathore and Rathore, 2016).

Search Space and Optimization Parameters for Hyper-parameter Tuning

This section describes the search space and optimization parameters used in hyperparameter tuning. Hyper-parameters are parameters specified before model training and can significantly affect the model's performance. The hyper-parameter tuning identifies the optimal combination of hyper-parameters to optimize the model performance. The search space consists of the distinct value spectrums explored for each hyper-parameter during optimization. In this study, we address the following mentioned six hyper-parameters and their respective ranges:

- *Embedding size*: an integer value between 32 and 128.
- *Number of neurons in the first hidden layer*: an integer between 1 and 4.

- *Number of neurons in the second hidden layer*: an integer between 1 and 8.
- *Learning rate*: a float value between 0.0001 and 0.1.
- *Batch size*: an integer value between 32 and 256.
- *Number of epochs*: an integer value between 1 and 20.

These parameters are carefully chosen to explore a wide range of possibilities while optimizing the model's performance. For instance, an embedding size of 85 is selected from the range of 32 to 128, balancing computational efficiency with embedding dimensionality. Similarly, a learning rate of 0.0473, chosen from the float range of 0.0001 to 0.1, facilitates efficient gradient descent during model training. Each parameter's range ensures thorough exploration, aiming to discover the most effective configuration that enhances model accuracy and generalization. During the optimization, the hyper-parameters are randomly sampled from the corresponding sub-search space, and the model's performance is evaluated using these hyper-parameter values. Furthermore, we specify the population size and the number of generations used in the genetic algorithm optimization to discover the best combination of hyper-parameters. In our investigation, we set the population size of six and evolved the models for up to three generations. In conjunction with the decision to employ a population size of six and restrict the number of generations to three, our choice was also informed by the nature of our dataset, which exhibits characteristics conducive to rapid learning due to its size and inherent simplicity. Given the dataset's substantial volume and straightforward patterns, we anticipated that the genetic algorithm would efficiently navigate the hyper-parameter space and converge to optimal or near-optimal solutions within a relatively short span of generations. Therefore, our experimental design aligns with the advantageous properties of the dataset, enabling us to leverage its ease of learning to achieve effective optimization outcomes within the specified constraints. Finally, we employed a mutation rate of 0.5 to introduce additional variation in the population.

Experimental Model Configurations

In this section, we present the deep learning models conducted for our sentiment analysis experiments, which incorporate LSTM architecture and genetic algorithms.

Long Short-Term Memory (LSTM) Model

Long short-term memory (LSTM) is a typical recurrent neural network (RNN) used to process sequential data. RNNs are a class of neural networks designed to process data in a sequence form, such as text or speech (Bozkurt et al., 2024; Kolukisa et al., 2021; Sherstinsky, 2020). Nevertheless, LSTMs are specifically designed to address the vanishing gradient issue, which can occur in RNNs when they are trained on long data sequences.

In our study, we constructed our deep-learning model for LSTM algorithms using the Keras deep neural network library (Chollet et al., 2015). The model architecture comprised an input layer with a dimension of 1000 as the input array was padded using the pad sequence arrangement. We then added an embedding layer with the size of which we used a number chosen randomly from the search space using genetic algorithms. Following, we incorporated a bidirectional Bi-LSTM layer (which is a powerful component for understanding long-range dependencies in the data by learning from both the antecedent and subsequent parts of a sequence), also with a number randomly chosen from the search space by genetic algorithms and included a dropout layer for fixing over-fitting issues. Finally, we repeated this process for Bidirectional LSTM and dropout layers before adding a dense layer to serve as the output layer. The activation function was "Softmax", while the optimizer function was the "Adam" algorithm for the model. The loss function was determined as the sparse-categorical-cross-entropy for the LSTM model. In the training phase, the batch size was set to a number selected randomly from the search space using genetic algorithms. Then, we trained the entire LSTM network for several epochs randomly picked by the genetic algorithms while maintaining a dropout value of 0.5.

Mechanism of Genetic Algorithm-Driven Approach

In this experimental effort, we operate a population-based optimization approach to create the initial generation of candidate solutions. Specifically, we initialize the population by generating a list object called population. The population capacity is determined by the population size defined in Section 4.5. The whole process is graphically illustrated in Figure 2 with the subfigures. Technically, we iterate over each parameter space in the "search space" list using a loop mechanism to generate each candidate solution. We use the "random.uniform" function to generate a random value within the lower and upper bounds of the corresponding space for continuous variables. However, we round the relevant value to the nearest integer for discrete variables. Each parameter value is then appended to a list of parameters for that iteration. Finally, we append the list of parameters to the "population" list. This process is

repeated for each iteration of the population size, resulting in a population of randomly generated parameter values sampled from the search space explained in Section 4.5. After generating the initial population, the next step is to evaluate the fitness of each candidate solution, and thus, for assessment purposes, we use accuracy as our fitness measure. Once we have calculated the fitness scores, we normalize them to ensure they sum up to one. This operation is achieved by dividing each fitness score by the sum of all fitness scores. The resulting normalized fitness scores are then treated as probabilities for selecting parents for the reproduction process described in Section 4.2. To select parents, we use a stochastic method called roulette wheel selection, where the probability of picking a candidate solution as a parent is proportional to its normalized fitness score expressed in Section 4.2. Specifically, we randomly choose two parent candidates from the population for each offspring. Next, we make new offspring through the reproduction mechanism. In this work, we utilize a crossover operator to produce the offspring. In particular, we randomly choose a crossover point for each parent pair and swap the values of the parent parameters before and after the crossover point. This procedure is repeated until we have the desired number of offspring, as stated in Section 4.3. Finally, the newly generated offspring are added to the population by replacing the least fit members of the previous generation. This process is repeated for a fixed number of generations or until a solution that meets the desired criteria is discovered. Afterward, we implement a mutation operator in some of the offspring to promote additional diversity in the population. Specifically, we randomly select an offspring in the current generation and apply the mutation operator to one of its parameters with a probability score of 0.5. As applied for each selected parameter, we generate a new random value within its corresponding search space bounds using the "random.uniform" function.

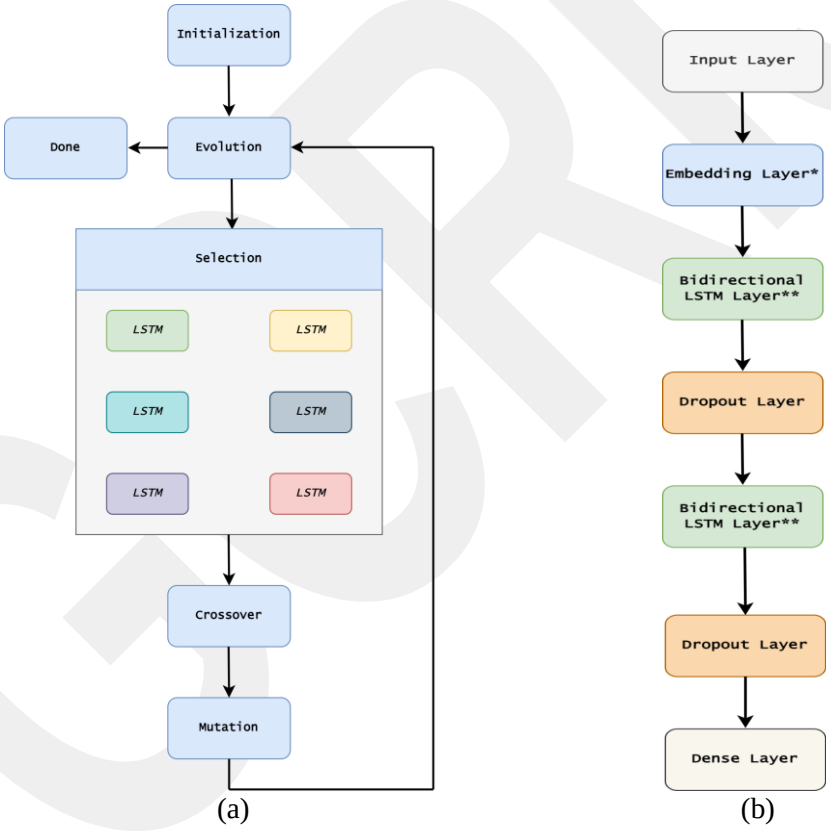

Figure 2. The sub-figure (a) represents the overall flow diagram of the proposed approach with LSTM integration, while the sub-figure (b) shows the internal LSTM architecture. The Genetic Algorithm selects *Embedding size and **Number of neurons in the corresponding layers.

RESULTS AND DISCUSSION

We conducted an experimental study to investigate the effectiveness of using a genetic algorithm-driven approach for hyper-parameter optimization in deep learning models. We specifically built LSTM models for sentiment analysis and found that the genetic algorithm approach was highly successful in tuning the hyper-parameters of the models, resulting in improved performance. Our results suggest that the genetic algorithm approach is a promising

technique for hyper-parameter optimization in deep learning, especially when an adequate amount of data is available.

As seen in Table 1 and inspected from Figures 3a, 3b, and, 3c, the fitness results indicate that exploiting genetic algorithms in deep learning can yield improved performance (the fitness score represents the accuracy achieved by the LSTM model using the hyperparameters determined by the genetic algorithm at each generation), but the amount of data used is a critical factor. When using only 5% of the data, a decrease of approximately 0.47% is observed. Nevertheless, we noticed increases in performance of 4.2% and 3.45%, respectively. These outcomes illustrate the potential value of genetic algorithms in deep learning and underscore the importance of considering the quantity of data when evaluating their efficacy.

Table 1. Fitness Scores of Models

	Generation Steps		
Percentage	1	2	3
5%	0.716690	0.706872	0.712014
10%	0.696820	0.738896	0.733286
20%	0.733021	0.726826	0.767504

These results have important implications for the deep learning models. Hyperparameter tuning is a crucial aspect of deep learning, as it can significantly impact the model's performance. Traditional methods for hyper-parameter tuning, such as grid search and random search, can be computationally expensive and time-consuming, specifically when dealing with large and complex models. However, genetic algorithms offer a more efficient and automated approach to hyper-parameter tuning, potentially saving significant time and resources. Still, it is important to note that the effectiveness of genetic algorithms for hyper-parameter tuning may vary depending on the specific characteristics of the data and model being used. For instance, the optimal hyperparameters for a given model may differ depending on the data set being used, and the effectiveness of the genetic algorithm may depend on the model's complexity and the hyper-parameter space's size. Therefore, it is important for future research to explore the effectiveness of genetic algorithms across a range of data sets and model types
.

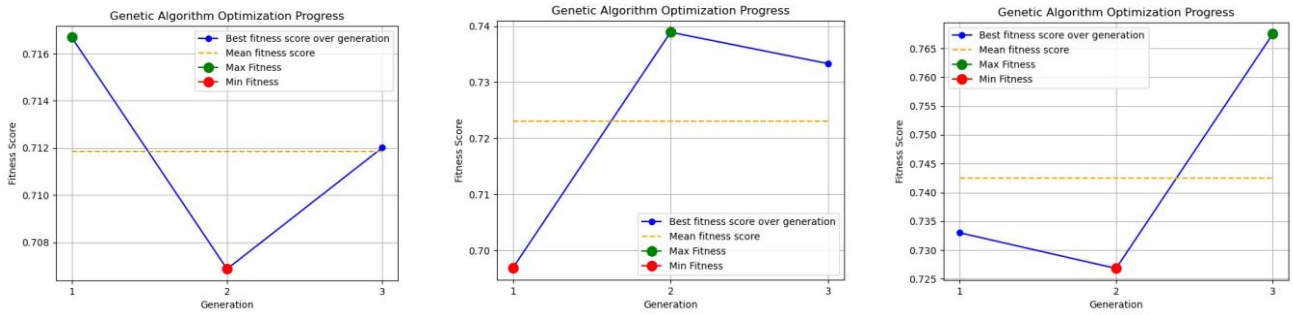

Figure 3. Overall Graphical Representation of the Fitness Scores with Unique Dataset Usage Rates. From Left to Right Images, **a.** 5% Fitness over Generation, **b.** 10% Fitness over Generation, **c.** 20% Fitness over Generation, Respectively.

In addition, it may be valuable to investigate the impact of other factors on the efficacy of genetic algorithms for hyper-parameter tuning. For example, the population size and selection criteria used in the genetic algorithm can significantly impact its performance. Similarly, the choice of hyper-parameter search space and encoding method can also play a role in determining the algorithm's effectiveness. By exploring these factors, researchers can gain a better understanding of the conditions under which genetic algorithms are most effective for hyper-parameter tuning. In addition to the merits of the genetic approach used, we presented classification results based on the cumulative use of the dataset, as shown in Figure 4, by confusion matrices. The average correct classification

percentage increased by 4% when the dataset size was increased from 5% to 10%. A further increase in the dataset size from 10% to 20% resulted in a 7% improvement. These findings underscore the direct correlation between dataset size and classification accuracy, reinforcing the importance of data volume in enhancing the efficacy of our genetic approach. Plus, these results suggest that the genetic method is a promising method for classification tasks, even with a small amount of data, and offers valuable insights into the potential scalability and robustness of our methodology across diverse dataset scales. Ultimately, they highlight the promising applicability of our approach to various real-world scenarios characterized by varying data availability.

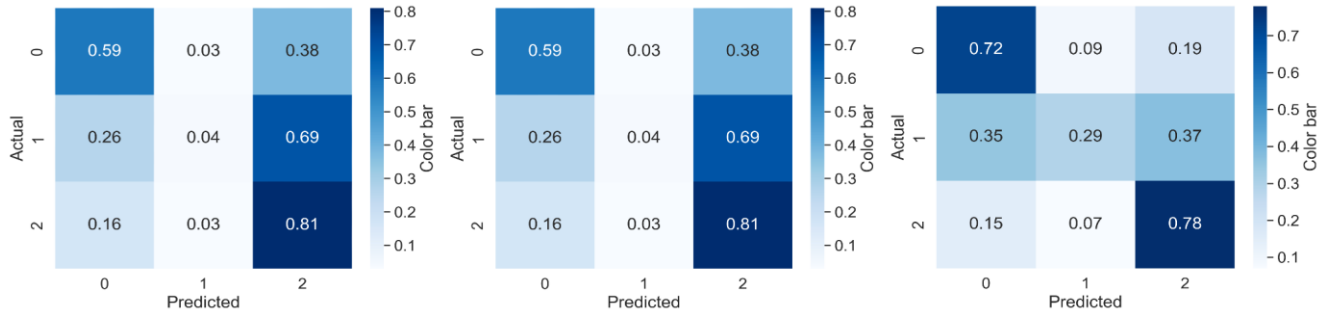

Figure 4. Confusion Matrix Representation of the Models Constructed by 5, 10, and 20 Percentages of the Dataset (0: negative, 1: neutral, 2: positive).

Overall, the findings of this study demonstrate the potential of genetic algorithms as a tool for hyper-parameter tuning in deep learning, mainly when large amounts of data are available. By further exploring the factors that impact the effectiveness of genetic algorithms, researchers can continue to refine and improve this approach, potentially leading to even more significant advancements in deep learning.

CONCLUSION

In conclusion, this study highlights the effectiveness of genetic algorithms for hyperparameter tuning in deep learning, specifically when ample amounts of data are available. The results show that increasing the data amount used for training can lead to performance improvements, indicating the potential of genetic algorithms to explore the hyperparameter space and identify better configurations. The findings have important implications for deep learning applications, as hyperparameter tuning is crucial for optimizing model performance. Genetic algorithms offer a more efficient and automated approach to hyper-parameter tuning, potentially saving time and resources compared to traditional methods. Nevertheless, the effectiveness of genetic algorithms may depend on the specific characteristics of the data and model chosen, and further research is needed to explore the impact of other factors, such as population size and selection criteria. Overall, this study suggests that genetic algorithms have great potential as a tool for hyper-parameter tuning in deep learning, and continued research in this area could lead to even better advancements.

FUTURE WORK

In the context of future research directions, this study suggests exploring the efficacy of various hyperparameter optimization techniques, including genetic algorithms, grid search, Bayesian optimization, and random search. By conducting a comprehensive comparison of these optimization approaches in terms of their time efficiency and performance, researchers can gain valuable insights into their respective strengths and limitations for addressing the problem at hand. Moreover, considering ensemble-based methods and meta-learning techniques for hyperparameter optimization could also be promising avenues for future investigation. By systematically investigating and integrating these diverse approaches, researchers can make significant strides in improving the optimization process and furthering advancements in the field.

REFERENCES

Bozkurt, B., Coskun, K., & Bakal, G. (2024). Building a challenging medical dataset for comparative evaluation of classifier capabilities. *Computers in Biology and Medicine*, 178, 108721.

Chollet, F., et al. (2015). Keras. https://keras.io.

Fang, Y., & Li, J. (2010). A review of tournament selection in genetic programming. Advances in computation and intelligence: 5th international symposium, isica 2010, wuhan, china, october 22-24, 2010. proceedings 5 (pp. 181–192).

Greenstein, B.L., Elsey, D.C., Hutchison, G.R. (2023). Determining best practices for using genetic algorithms in molecular discovery. *The Journal of Chemical Physics*, 159 (9).

Hajireza, M., Darabi, R., Najafi Moghaddam, A. (2023). The impact of accruals and free cash flow on financial stability using genetic algorithm. Accounting and Auditing Studies.

Haldurai, L., Madhubala, T., Rajalakshmi, R. (2016). A study on genetic algorithm and its applications. *Int. J. Comput. Sci. Eng*, 4 (10), 139–143.

Holland, J.H. (1992a). Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT Press.

Holland, J.H. (1992b). Genetic algorithms. Scientific American, 267 (1), 66–73, Retrieved 2023-05-07, from http://www.jstor.org/stable/24939139.

Jebari, K. (2013, 12). Selection methods for genetic algorithms. *International Journal of Emerging Sciences*, 3, 333-344.

Kallumadi, S., & Grer, F. (2018). Drug Review Dataset (Drugs.com). UCI Machine Learning Repository. (DOI: https://doi.org/10.24432/C5SK5S).

Katoch, S., Chauhan, S.S., Kumar, V. (2021). A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*, 80 , 8091–8126.

Kolukisa, B., Dedeturk, B.K., Dedeturk, B.A., Gulsen, A., Bakal, G. (2021). A comparative analysis on medical article classification using text mining & machine learning algorithms. 2021 6th International Conference on Computer Science and engineering (UBMK) (p. 360-365).

Kora, P., & Yadlapalli, P. (2017, 03). Crossover operators in genetic algorithms: A review. *International Journal of Computer Applications*, 162, 34-36, https:// doi.org/10.5120/ijca2017913370.

Kramer, O., & Kramer, O. (2017). Genetic algorithms. Springer.

Lambora, A., Gupta, K., Chopra, K. (2019). Genetic algorithm-a literature review. 2019 international conference on machine learning, big data, cloud and parallel computing (comitcon) (pp. 380–384).

Lipowski, A., & Lipowska, D. (2012). Roulette-wheel selection via stochastic acceptance. *Physica A: Statistical Mechanics and its Applications*, 391 (6), 2193–2196.

Pachuau, J.L., Roy, A., Kumar Saha, A. (2021). An overview of crossover techniques in genetic algorithm. Modeling, Simulation and Optimization: Proceedings of CoMSO 2020, 581–598.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . .others (2011). Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12, 2825–2830.

Pencheva, T., Atanassov, K., Shannon, A. (2009). Modelling of a stochastic universal sampling selection operator in genetic algorithms using generalized nets. Proceedings of the tenth international workshop on generalized nets, sofia (pp. 1–7).

Peng, K., Du, J., Lu, F., Sun, Q., Dong, Y., Zhou, P., Hu, M. (2019). A hybrid genetic algorithm on routing and scheduling for vehicle-assisted multi-drone parcel delivery. *IEEE Access*, 7, 49191–49200.

Rathore, H., & Rathore, H. (2016). Genetic algorithms. Mapping Biological Systems to Network Systems, 97–106.

Sherstinsky, A. (2020). Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404, 132306.

Yan, J. (2023). Adaptive scheduling of agricultural machinery equipment production lines for intelligent manufacturing. *International Journal of Manufacturing Technology and Management*, 37 (3/4), 349–361.

Zheng, L., & Wen, Y. (2023). A multi-strategy differential evolution algorithm with adaptive similarity selection rule. *Symmetry*, 15 (9), 1697.

Zivkovic, M., K, V., Bacanin, N., Djordjevic, A., Antonijevic, M., Strumberger, I., Rashid, T.A. (2021). Hybrid genetic algorithm and machine learning method for covid-19 cases prediction. Proceedings of international conference on sustainable expert systems: Icses 2020 (pp. 169–184).

Şen, T. Ü., & Bakal, G. (2023). A transfer learning application on the reliability of psychological drugs' comments. 2023 international conference on smart applications, communications and networking (smartnets) (p. 1-6).