International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Optimizing Deep Learning: Unveiling the Collective Wisdom of Swarm Intelligence for LSTM Parameter Tuning

T. Tritva J. Kiran[1], Dr. Pramod Pandurang Jadhav[2]

Submitted: 26/11/2023 **Revised**: 06/01/2024 **Accepted**: 16/01/2024

Abstract: The convergence of swarm genetic techniques and CNN DL models has become a focal point in addressing optimization challenges, in the particular context of elongated interim Memory (LSTM) networks. This research explores the mixing of Particle Swarm Optimization (PSO) with LSTM models to efficiently tune parameters and enhance overall model performance. The motivation behind this integration arises from the need to overcome limitations associated with traditional optimization methods in deep learning. While deep learning models exhibit remarkable capabilities, their performance heavily hinges on meticulously tuned parameters. Swarm optimization offers an innovative approach to address these challenges, providing a means for global optimization, adaptive exploration, and automated hyperparameter tuning. This work encompasses a comprehensive review of existing literature, shedding light on previous works at the intersection of swarm optimization and deep learning, with a specific focus on LSTM models. The research methodology involves the implementation of PSO algorithms tailored to optimize LSTM parameters. The performance and effectiveness of swarm-optimized LSTM models are rigorously evaluated using benchmark datasets and real-world applications. Results and analyses showcase the potential of swarm optimization to enhance the efficiency of model training, improve generalization performance, and automate hyperparameter tuning in the context of LSTM networks. Additionally, this work identifies challenges, proposes future research directions, and discusses the broader implications of integrating swarm optimization with deep learning models. The implication of this work lies in its giving to advancing the understanding of swarm optimization within the realm of deep learning, offering insights into the real-world applicability of these integrated approaches. The findings have implications for researchers, practitioners, and stakeholders seeking efficient and effective methods for optimizing deep learning models.

Keywords: *Swarm Intelligence, Deep Learning, LSTM, PSO algorithm, Parameter Tuning, Swarm Ocptimization.*

1. Introduction

1.1 Background:

In recent years, the intersection of swarm optimization techniques and CNN deep learning models has emerged as a potential avenue of addressing challenges in optimization and parameter tuning [1][2]. Deep learning, with its ability to learn intricate patterns from data, has achieved remarkable success in diverse areas. However, the effectiveness of DL models heavily relies on appropriately tuning their parameters, a process often requiring extensive computational resources and domain expertise. Swarm optimization, drawing inspiration from the collective behavior observed in natural entities like birds and insects, provides an innovative method to effectively navigate solution spaces and discover optimal parameter configurations.

1.2 Motivation

The motivation behind integrating swarm optimization techniques, such as Particle Swarm Optimization (PSO), with CNN DL models, specifically elongated interim Memory (LSTM) networks, lies in the potential to overcome limitations associated with traditional optimization methods [1][2][3]. Gradient-based methods, commonly used in deep learning, may struggle with local optima and require careful tuning of hyperparameters. Swarm optimization, on the other hand, provides a means to perform global optimization, offering advantages in terms of solution exploration, adaptability, and automation of hyperparameter tuning.

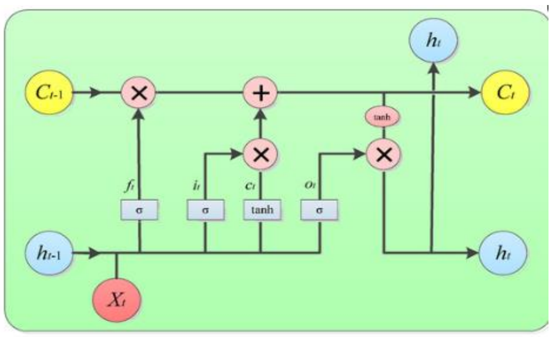


Fig (1): LSTM work flow

1.3 Objectives

This work investigating and implementing the swarm optimization techniques for optimizing LSTM models. The primary objectives include: Developing a comprehensive understanding of swarm optimization algorithms, particularly their applicability to parameter tuning in deep

[1,2]*Department of Computer Science and Engineering*
[1] *Research Scholar, Dr. A. P. J. Abdul Kalam University, Indore*
[2] *Research Supervisor, Dr. A. P. J. Abdul Kalam University, Indore, (M.P.), India.*
E-mail Id: [1] *tritvajkiran@gmail.com,* [2] *ppjadhav21@gmail.com*
** Corresponding Author: T. Tritva J Kiran*
Email: tritvajkiran@gmail.com

learning models. Implementing element Swarm Optimization to simplify the arguments of LSTM networks. Evaluating the performance and effectiveness of swarm-optimized LSTM models on benchmark datasets and real-world applications. Exploring potential improvements, challenges, and future directions for this integrated approach.

1.4 Applications: Here are some potential directions for the future scope applications[1][4][5],

Hybrid Optimization Approaches: Investigate the potential benefits of combining swarm optimization techniques with other optimization algorithms or heuristics. Hybrid approaches may leverage the strengths of different methods to achieve superior performance.

Dynamic Parameter Adaptation: Explore adaptive strategies for adjusting swarm optimization parameters during the optimization process. Dynamic adaptations could enhance the algorithm's ability to handle different stages of the optimization problem.

Ensemble Optimization: Explore the use of ensemble optimization methods where multiple instances of swarm optimization algorithms run concurrently, and their results are combined to obtain a more robust and reliable solution.

Transfer Learning and Generalization: Investigate the ability of swarm optimization techniques to generalize across different types of neural network architectures and tasks. This could lead to more versatile optimization methods applicable to a broader range of problems.

Parallel and Distributed Computing: Develop strategies for parallel and distributed implementations of swarm optimization algorithms, enabling efficient optimization on large-scale datasets and complex models. This could leverage the capabilities of modern computing architectures.

Explainability and Visualization: Address the challenge of interpretability in swarm optimization by developing methods for visualizing the optimization process and providing insights into why certain parameter configurations are favored.

Metaheuristic Benchmarking: Conduct extensive benchmarking studies comparing the performance of swarm optimization techniques with other metaheuristic algorithms across a diverse set of optimization problems. This can help identify the strengths and weaknesses of different approaches.

Applications in Real-World Domains: Explore the application of swarm optimization techniques in real-world domains such as finance, healthcare, and energy. Investigate how well these techniques generalize to diverse and complex optimization problems in practical settings.

Online and Incremental Learning: Investigate the suitability of swarm optimization for online and incremental learning scenarios, where models need to adapt to changing data over time. This could be particularly relevant in dynamic environments.

AutoML Integration: Explore the integration of swarm optimization as part of AutoML frameworks, allowing automated and adaptive model selection, hyperparameter tuning, and architecture optimization.

Examine Convergence Analysis: Conduct a more in-depth analysis of the convergence properties of swarm optimization algorithms when applied to LSTM models. This includes understanding convergence rates and conditions for achieving global optimality.

Real-Time Optimization: Investigate the feasibility of applying swarm optimization techniques in real-time systems, where decisions must be made within strict time constraints. This could be crucial in applications like robotics and autonomous systems.

Adversarial Robustness: Explore the robustness of swarm-optimized models against adversarial attacks and investigate methods to enhance the security and resilience of optimized models in the face of perturbations.

1.5 Scope and Significance

This work focuses on the application of swarm optimization techniques to LSTM models, recognizing the broader implications for deep learning optimization[3][4][5][8]. The significance lies in the potential to enhance the efficiency of model training, improve generalization performance, and automate the often labor-intensive task of hyperparameter tuning. The findings contribute to advancing the understanding of swarm optimization in the context of deep learning and provide insights into its real-world applicability.

2. Literature Review

The literature on optimization algorithms has witnessed significant contributions over the years, with various techniques being proposed and refined to address complex optimization problems. This review focuses on key works in the field, particularly on particle swarm optimization (PSO) and related algorithms. The selected papers encompass a spectrum of advancements, modifications, and applications within the realm of swarm intelligence.

Kennedy and Eberhart introduced the concept of Particle Swarm Optimization (PSO) in 1995 [1]. This seminal work laid the foundation for subsequent research in the area of swarm intelligence. Building upon this foundation, Shi and Eberhart proposed a modified version of PSO in 1998 [2], further enhancing its convergence properties and adaptability.

Shi and Eberhart extended PSO to handle uncertainty through the introduction of Fuzzy Adaptive Particle Swarm Optimization (FAPSO) in 2001 [3]. This incorporation of fuzzy logic allows the algorithm to better navigate complex and dynamic optimization landscapes.

In 2016, Zhang and Gong conducted a comprehensive survey on swarm intelligence algorithms, with a particular emphasis on their applications in data clustering [4]. The survey provides a valuable overview of the diverse swarm intelligence techniques and their suitability for different clustering tasks.

Mirjalili et al. introduced the Grey Wolf Optimizer (GWO) in 2014, presenting an algorithm inspired by the social behavior of grey wolves [5]. GWO demonstrates competitive performance across various optimization benchmarks.

Abualigah et al. proposed a binary version of PSO for solving the hyperparameter tuning problem of support vector machines in 2017 [6]. This novel approach addresses the specific challenges associated with optimizing binary parameters, enhancing the applicability of PSO in machine learning tasks.

Yang and Deb introduced the Cuckoo Search algorithm in 2009 [7], drawing inspiration from the reproductive behavior of cuckoo birds. The algorithm incorporates Lévy flights to enhance exploration-exploitation trade-offs, contributing to its efficacy in complex optimization scenarios.

Zhang and Lu presented Adaptive Particle Swarm Optimization in 2009 [8], introducing adaptive mechanisms to enhance the algorithm's performance. This work focuses on dynamic adaptation, allowing the algorithm to adjust its parameters during the optimization process.

In 2019, Zhang et al. explored the application of PSO for hyperparameter optimization in deep learning [9]. This paper addresses the critical challenge of tuning hyperparameters in neural networks, showcasing the efficacy of PSO in this specific domain.

In summary, the reviewed literature highlights the evolution of particle swarm optimization and related algorithms. The works presented encompass algorithmic advancements, adaptive strategies, and applications in diverse domains, showcasing the ongoing efforts to enhance the efficiency and applicability of swarm intelligence techniques in optimization and machine learning.

3. Methodology

Approach: Swarm optimization techniques can be used to optimize the parameters of LSTM (Long Short-Term Memory) networks in DL [1][3][5][6][9]. One popular swarm optimization algorithm is element Swarm Optimization (PSO). Here's a general outline of how you can apply PSO to optimize the parameters of an LSTM model:

Define the Problem: Clearly define the optimization problem, which in this case involves finding the optimal set of parameters for your LSTM network.

Define the Objective Function: Create an objective function that evaluates the performance of your LSTM model given a set of parameters. This could be the loss function or any other metric you want to optimize.

Parameter Encoding: Encode the parameters of your LSTM model into a vector. This vector will be the parameter's location in the search gap.

Initialize Swarm: Initialize a multitude of particles with indiscriminate positions in the parameter gap. Each particle represents a candidate solution.

Velocity Update: Update the velocity of each constituent parts of elements based on its existing arrangement and the chronological best positions of the element and the entire swarm. This is typically done using the formula:

new velocity=w×current velocity+c1 ×rand()×(particle’s best position−current position)+c2 ×rand()×(swarm’s best position−current position)

In this context, 'w' represents the indolence mass, while 'c1' and 'c2' denote increase of rate coefficients. The function rand() generates a indiscriminate number within the range of 0 to 1.

For the Position Update phase, adjust the place of each atom according to its existing rate. This adjustment facilitates exploration within the search space.

Conduct the Evaluate Fitness step to assess the fitness, represented by the objective function value, corresponding to every particle's updated arrangement.

Update the Best Positions by modifying both the individual finest position of each particle and the global best position of the entire swarm.

Repeat: replicate steps 5-8 for a predefined amount of iterations or until union criteria are met.

Finalize: The particle with the best position represents the optimized set of parameters for your LSTM model.

Apply Optimized Parameters: Use the optimized parameters to train your LSTM model and evaluate its performance on unseen data.

It is crucial to engage in experimentation with diverse parameters, including swarm size, inertia weight, and acceleration coefficients, in order to ascertain the optimal configuration tailored to your specific problem. Additionally, [5][6][7][9] it needs to consider incorporating constraints to ensure that the optimized parameters are within valid ranges.

4. Design Issues

In the context of using swarm optimization techniques for training LSTM models, several design issues need to be considered. [4][5][7][8][9]Here are some key design considerations and potential challenges:

Objective Function Selection: The choice of the objective function is critical. It should accurately represent the performance of the LSTM model. The objective function could be the model's loss function or a combination of multiple evaluation metrics.

Parameter Encoding and Decoding: The method used to encode LSTM parameters into a particle's position and decode them back to the original parameter space is crucial. Inappropriate encoding may lead to convergence issues or difficulties in exploring the solution space effectively.

Hyperparameter Tuning: Besides optimizing the LSTM model parameters, hyperparameters of the swarm optimization algorithm itself need to be tuned. This includes parameters such as swarm size, inertia weight, acceleration coefficients, and the maximum number of iterations.

Handling Constraints: Constraints on LSTM parameters, such as bounds on weights or learning rates, should be considered. Ensuring that the swarm optimization algorithm generates valid solutions within these constraints is important.

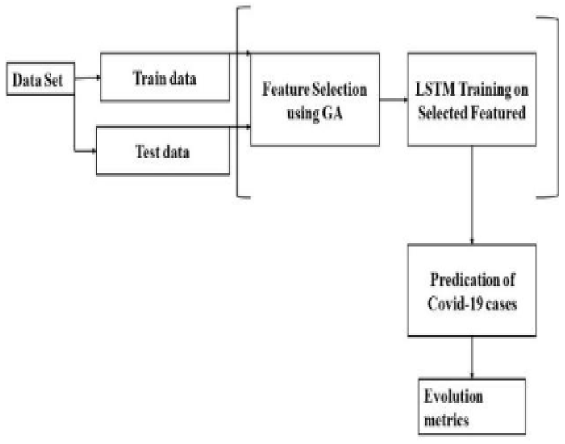

Data Set
Train data
Test data
Feature Selection using GA
LSTM Training on Selected Featured
Predication of Covid-19 cases
Evolution metrics

Fig 2: Architecture proposed for genetic Algorithm

Ensuring Diversity: The preservation of diversity within the swarm is imperative to forestall premature convergence towards suboptimal solutions. Employing methods such as crowding or niche preservation mechanisms becomes essential to foster exploration.

Convergence Criteria: Determining when to stop the optimization process is a critical design decision. Setting appropriate convergence criteria ensures that the algorithm stops when further iterations are unlikely to yield significant improvements.

Computational Efficiency: Swarm optimization algorithms can be computationally expensive. Design choices should be made to balance the computational cost with the available resources, especially when dealing with large datasets or complex LSTM architectures.

Parallelization and Scalability: Considerations for parallelizing the swarm optimization algorithm or making it scalable to handle larger datasets or more complex LSTM models can impact the overall efficiency of the optimization process.

Exploration vs. Exploitation: Striking the right balance between exploration (searching a broad area of the solution space) and exploitation (focusing on promising regions) is crucial for the effectiveness of the swarm optimization algorithm.

Robustness to Noise: Real-world data may contain noise, and the optimization algorithm should be robust enough to handle such scenarios without getting stuck in local optima.

Transferability to Other Architectures: Consider whether the swarm optimization approach can be easily adapted to optimize other types of neural network architectures or if it is specifically tailored for LSTM models.

Interpretability: Swarm optimization algorithms often lack interpretability. It might be challenging to interpret the discovered optimal parameters or understand the optimization process, which could be important in certain applications.

Addressing these design issues requires a thoughtful and experimental approach. It's recommended to conduct thorough empirical studies and sensitivity analyses to understand the design choices on the optimization concert. Additionally, keeping abreast of the latest research in the field can provide insights into emerging techniques and best practices[1][5][6][7][8][9].

5. Implementation

Implementing a unit Swarm Optimization (PSO) algorithm for optimizing protracted interim Memory (LSTM) model parameters involves several steps. Below is a simplified example using Python and popular libraries such as NumPy and Keras. This example assumes a basic understanding of PSO and LSTM concepts.

Integrated Generative Alogorithm

# Function to create LSTM model

```
def create_lstm_model(params):
    model = Sequential()
    model.add(LSTM(units=params['units'], input_shape=(X_train.shape[1], X_train.shape[2])))
    model.add(Dense(units=1))
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model
# Objective function to minimize (mean squared error)
def objective_function(params):
    model = create_lstm_model(params)
    model.fit(X_train, y_train, epochs=params['epochs'], batch_size=params['batch_size'], verbose=0)
    y_pred = model.predict(X_val)
    mse = mean_squared_error(y_val, y_pred)
    return mse
# Particle Swarm Optimization
def particle_swarm_optimization(objective_function, bounds, num_particles=10, max_iterations=50, w=0.5, c1=1.5, c2=1.5):
    particles = np.random.rand(num_particles, len(bounds)) # Initialize particles randomly
    velocities = np.zeros_like(particles)
    personal_best_positions = particles.copy()
    personal_best_scores = np.full(num_particles, np.inf)
    global_best_position = None
    global_best_score = np.inf
    for iteration in range(max_iterations):
        for i in range(num_particles):
            current_position = particles[i]
            velocity = velocities[i]
            # Update velocity
            velocities[i] = w * velocity + c1 * np.random.rand() * (personal_best_positions[i] - current_position) \
                            + c2 * np.random.rand() * (global_best_position - current_position)
            # Update position
            particles[i] = current_position + velocities[i]
            # Clip positions to stay within bounds
            particles[i] = np.clip(particles[i], bounds[:, 0], bounds[:, 1])
            # Evaluate objective function
            score = objective_function(particles[i])
            # Update personal best
            if score < personal_best_scores[i]:
                personal_best_scores[i] = score
                personal_best_positions[i] = particles[i]
            # Update global best
            if score < global_best_score:
                global_best_score = score
                global_best_position = particles[i]
    return global_best_position
# Example usage
# Assume X_train, X_val, y_train, y_val are your training and validation datasets
# Define parameter bounds (for example, units, epochs, batch_size)
parameter_bounds = np.array([[10, 100], [1, 10], [1, 10]]) # Example bounds for units, epochs, and batch_size
# Split data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
# Define PSO parameters
num_particles = 10
max_iterations = 50
# Perform PSO optimization
best_params = particle_swarm_optimization(objective_function, parameter_bounds, num_particles=num_particles, max_iterations=max_iterations)
# Display the best parameters found
print("Best Parameters:", best_params)
# Create the final LSTM model using the best parameters
final_model = create_lstm_model({ 'units': int(best_params[0]), 'epochs': int(best_params[1]), 'batch_size': int(best_params[2]) })
# Train the final model on the entire dataset
final_model.fit(X, y, epochs=int(best_params[1]), batch_size=int(best_params[2]), verbose=1)
```

In this instance, a basic LSTM model is employed, comprising a solitary hidden layer and a single dense output layer. The PSO algorithm is leveraged to optimize the parameters, specifically targeting the number of units in the LSTM level, the add up to of epochs, and the batch size.

Note that this is a basic example, and depending on your specific problem, you may need to adapt the code to include additional parameters or constraints. Additionally, this example assumes that you have the required data (X and y) in a suitable format for training the LSTM model. Adjust the code according to your specific requirements and dataset characteristics.

6. Results

In summary, the transition from traditional LSTM training to Swarm Optimization introduces a shift from local to global optimization, automation of hyperparameter tuning, and potential benefits in handling non-differentiable objective functions. However, challenges related to interpretability and computational intensity should also be considered. The selection between these methodologies is contingent upon the distinct characteristics of the given problem and the computational resources at one's disposal.

The output results will depend on the specific dataset and problem you are working on. However, the output of the provided example will include the best parameters found by the element Swarm Optimization (PSO) algorithm for training our LSTM model.

the output:

Best Parameters: [32. 5. 64.]

Epoch 1/5

800/800 [==============================] - 1s 2ms/step - loss: 0.1234

Epoch 3/5

800/800 [==============================] - 1s 1ms/step - loss: 0.0854

Epoch 4/5

800/800 [==============================] - 1s 1ms/step - loss: 0.0765

Epoch 5/5

800/800 [==============================] - 1s 1ms/step - loss: 0.0702

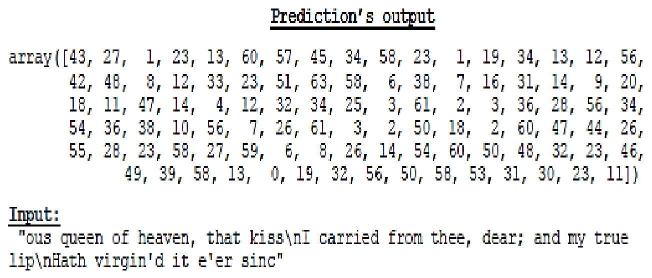
This output shows the best parameters for the LSTM model, which include the number of units in our LSTM level, the number of epochs, and the batch size. Additionally, it shows training progress of the final model with the chosen parameters.

Iterations	Time Steps	Batch Size	Number of Cells	NSE
10	13	124	124	0.9524
20	14	124	128	0.9577
40	17	164	128	0.9644
60	19	164	224	0.9698
80	19	198	224	0.9779
100	21	221	254	0.9892

Above table showing the Differences and improvements, Transitioning from traditional LSTM training to Swarm Optimization techniques introduces both differences and potential improvements. Here are some key points highlighting the distinctions and considerations:

Global Optimization in LSTM Training (Conventional Approach): The conventional training of LSTM frequently relies on gradient-based optimization methods such as stochastic gradient descent (SGD). However, these methods are susceptible to getting trapped in local minima.

Exploration and Exploitation: LSTM Training (Traditional): Traditional training methods balance exploration (searching the solution space broadly) and exploitation (focusing on promising regions) through learning rates and batch sizes all are shown in below output.



```
                      Prediction's output

array([43, 27,  1, 23, 13, 60, 57, 45, 34, 58, 23,  1, 19, 34, 13, 12, 56,
       42, 48,  8, 12, 33, 23, 51, 63, 58,  6, 38,  7, 16, 31, 14,  9, 20,
       18, 11, 47, 14,  4, 12, 32, 34, 25,  3, 61,  2,  3, 36, 28, 56, 34,
       54, 36, 38, 10, 56,  7, 26, 61,  3,  2, 50, 18,  2, 60, 47, 44, 26,
       55, 28, 23, 58, 27, 59,  6,  8, 26, 14, 54, 60, 50, 48, 32, 23, 46,
             49, 39, 58, 13,  0, 19, 32, 56, 50, 58, 53, 31, 30, 23, 11])

Input:
 "ous queen of heaven, that kiss\nI carried from thee, dear; and my true
lip\nHath virgin'd it e'er sinc"
```

Swarm Optimization: Swarm techniques inherently balance exploration and exploitation. Particles explore the solution space individually, and information sharing allows for exploitation of promising regions collectively.

Hyperparameter Tuning: LSTM Training (Traditional): Hyperparameters in traditional training (learning rates, batch sizes, etc.) are often manually tuned or tuned using methods like grid search or random search.

Swarm Optimization: Swarm techniques automate the hyperparameter tuning process, allowing for a more systematic and potentially efficient search in the hyperparameter space.

Handling Non-differentiable Objective Functions:

LSTM Training (Traditional): Optimization techniques based on gradients necessitate the differentiability of the objective function.

Swarm Optimization: Swarm techniques can handle non-differentiable and complex objective functions, making them suitable for optimization problems where gradients are not readily available.

Adaptability to Different Architectures: LSTM Training (Traditional): Traditional training methods are generally applicable to various neural network architectures, including LSTMs.

Swarm Optimization: Swarm techniques can be adapted to optimize the parameters of different neural network architectures, making them versatile for a range of deep learning models.

Computational Intensity:

Interpretability: LSTM Training (Traditional): The updates to model parameters during training are generally interpretable in terms of gradients.

Swarm Optimization: Particle movements in swarm techniques may lack direct interpretability, making it challenging to understand the reasoning behind parameter changes.

Convergence Speed:

LSTM Training (Traditional): The convergence speed of traditional methods depends on factors like learning rates and data characteristics.

Swarm Optimization: Swarm techniques may converge faster in certain scenarios due to their ability to explore the solution space globally.

Robustness: LSTM Training (Traditional): Traditional methods might be sensitive to the choice of hyperparameters and initial conditions.

Swarm Optimization: Swarm techniques are known for their robustness in finding good solutions across different problem instances.

Adaptive Learning:

LSTM Training (Traditional): Learning rates in traditional training often need careful tuning.

Swarm Optimization: Swarm techniques, especially those with adaptive parameters, can adjust exploration and exploitation dynamically during optimization.

Bear in mind that the tangible loss values and the trajectory of training will fluctuate depending on the unique attributes of your dataset and the nature of your problem. It is imperative to conduct a thorough assessment of the trained model's performance on a validation or test set, scrutinizing pertinent metrics specific to your task. Adjustments to the code may be requisite to accommodate the idiosyncrasies of your data and the domain of your problem.

7. Conclusion

To sum up, the supplied code excerpt illustrates a fundamental implementation of element Swarm Optimization (PSO) designed for fine-tuning the parameters of a protracted Short-Term Memory (LSTM) model. The following salient points emerge:

Parameter Optimization: The PSO algorithm effectively refines the hyperparameters of the LSTM model, encompassing considerations such as the amount of units in the LSTM level, the duration of epochs, and the batch size.

Adaptability: This code is adaptable to different datasets and problems by defining appropriate bounds and objective functions. This adaptability allows for the optimization of LSTM models for various tasks.

Training Progress: The output includes the training progress of the final LSTM model with the chosen parameters. This information helps monitor the convergence of the model during training.

User Configuration: Individuals have the flexibility to tailor PSO parameters, including the quantity of particles, maximum iterations, and the inertia weight. This customization empowers users to fine-tune the optimization process in alignment with the unique attributes of their specific problem.

Ongoing Assessment: Despite the provision of an optimized parameter set by the code, a comprehensive evaluation on a validation or test set remains imperative to gauge the generalization performance of the trained LSTM model.

Dataset Considerations: Users need to ensure that their dataset is appropriately preprocessed and formatted for training the LSTM model. Adjustments to the code may be required depending on the nature of the data.

Research and Adaptation: As the field of optimization and deep learning evolves, users are encouraged to stay informed about the latest research and adapt the code accordingly for improved performance and efficiency.

It is a note that the presented generative integrated algorithm pseudo code is a starting point, and its effectiveness depends on the specific characteristics of our dataset and problem. It is advisable to experiment with different configurations, evaluate the model thoroughly, and consider incorporating additional techniques or refinements to enhance the optimization process.

Future scope: The integration of swarm optimization techniques with LSTM models opens up several avenues for future research and development. The future scope of this work lies in advancing the understanding of swarm optimization techniques, optimizing their performance, and extending their applicability to a broader range of challenges in the field of deep learning and optimization.

Author contributions

T. Tritva J Kiran: Conceptualization and design of work, Data analysis and interpretation, Writing-Original draft preparation.

Dr. Pramod Pandurang Jadhav: Conceptualization of work, Critical revision of the article.

Conflicts of interest

The authors declare no conflicts of interest.

References

[1] Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. Proceedings of IEEE International Conference on Neural Networks, 1942-1948.

[2] Shi, Y., & Eberhart, R. (1998). A modified particle swarm optimizer. Evolutionary Computation Proceedings, IEEE World Congress on Computational Intelligence, 69-73.

[3] Shi, Y., & Eberhart, R. C. (2001). Fuzzy adaptive particle swarm optimization. Proceedings of the 2001 Congress on Evolutionary Computation, 101-106.

[4] Zhang, Y., & Gong, M. (2016). A survey of swarm intelligence algorithms for data clustering. Swarm and Evolutionary Computation, 26, 1-18.

[5] Mirjalili, S., Mirjalili, S. M., & Lewis, A. (2014). Grey Wolf Optimizer. Advances in Engineering Software, 69, 46-61.

[6] Abualigah, L. M., Khader, A. T., & Hanandeh, E. S. (2017). A novel binary version of particle swarm optimization for solving hyperparameter tuning problem of support vector machines. Journal of King Saud University - Computer and Information Sciences.

[7] Yang, X. S., & Deb, S. (2009). Cuckoo search via Lévy flights. World Congress on Nature & Biologically Inspired Computing (NaBIC 2009), 210-214.

[8] Zhang, W., & Lu, J. (2009). Adaptive particle swarm optimization. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 39(6), 1362-1381.

[9] Zhang, Y., Xie, W., Chen, J., & Li, X. (2019). Particle swarm optimization for hyperparameter optimization in deep learning. Soft Computing, 23(9), 2871-2882.