Training neural networks with Firefly optimization algorithm

SEDJARI Yassine
National school For Computer Science
Artificial intelligence major
Rabat , Morocco
yassine_sedjari@um5.ac.ma

EZZALI Mustapha
National school For Computer Science
Artificial intelligence major
Rabat , Morocco
mustapha_ezzali@um5.ac.ma

***Abstract*—This project presents a comparative analysis between the Firefly Optimization Algorithm (FOA) and the classical approach of gradient descent in optimizing a Neural Network. FOA is a nature-inspired algorithm that mimics the collective behavior of fireflies to optimize objective functions and performs better in non-convex and complex optimization scenarios due to its ability to escape local optima. However, gradient descent, leveraging its iterative approach, is more efficient and performs better in simple, convex optimization problems. In this paper, we conducted a comparative study by applying both optimizers to train a Neural Network using the IRIS dataset, examining their respective performance, convergence, and accuracy in optimizing the network's parameters.**

***Index Terms*—Neural Nerworks, Collective Intelligence, Firefly, Gradient Descent.**

I. Introduction

Swarm Intelligence Algorithms and Neural Networks are two powerful concepts in the field of artificial intelligence and optimization.

Swarm Intelligence Algorithms (SIAs) are inspired by the collective behavior of social insect colonies, such as ants, bees, and bird flocks. These algorithms leverage the principles of self-organization and decentralized decision-making to solve complex problems. SIAs exhibit emergent intelligence, where a group of simple individuals interacting with each other and their environment can collectively solve complex tasks.

Neural Networks, on the other hand, are a class of machine learning models inspired by the structure and function of the human brain. They consist of interconnected artificial neurons, organized in layers, which can learn from data and make predictions or decisions. Neural networks are capable of learning complex patterns and relationships, making them highly effective in various tasks such as image recognition, natural language processing, and time series prediction.

The combination of Swarm Intelligence Algorithms and Neural Networks offers a synergistic approach to problem-solving and optimization. SIAs can be used to train neural networks, optimize their architecture, and fine-tune their parameters. Conversely, neural networks can enhance the capabilities of swarm intelligence algorithms by providing them with learning and generalization capabilities.

Swarm Intelligence Algorithms, such as Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), and Firefly Algorithm (FA), can be employed to optimize the weights and biases of neural networks. These algorithms explore the solution space by iteratively updating the positions of particles, ants, or fireflies based on their own experience and the knowledge shared within the swarm. By leveraging the collective intelligence and search capabilities of swarm algorithms, neural networks can be trained more effectively and efficiently.

Furthermore, neural networks can enhance swarm intelligence algorithms by providing them with adaptive behaviors and decision-making capabilities. For example, neural networks can be used to model the behavior and interactions of individuals within a swarm, allowing for more realistic and effective simulations. Neural networks can also be employed to process the sensory information of swarm agents, enabling them to make more informed decisions and adapt to changing environments.

In summary, Swarm Intelligence Algorithms and Neural Networks complement each other, offering a powerful combination for solving complex problems and optimizing various tasks. The swarm intelligence algorithms provide optimization capabilities, while neural networks provide learning, pattern recognition, and decision-making abilities. The synergy between these two concepts has the potential to drive advancements in artificial intelligence and contribute to solving real-world challenges.

II. Train Neural networks using gradient descent

1. Initialize the weights and biases:

The weights and biases are typically initialized randomly. For a neural network with weights denoted as W and biases denoted as b, the initialization can be represented as:

$$W_{\text{init}}^{[1]} = \text{random initialization}$$

$$b_{\text{init}}^{[1]} = \text{random initialization}$$

$$W_{\text{init}}^{[2]} = \text{random initialization}$$

$$b_{\text{init}}^{[2]} = \text{random initialization}$$

2. Forward propagation:

During forward propagation, the activations of each neuron are computed by applying activation functions to the weighted sum of inputs and biases. Assuming we have an input vector X, the forward propagation step can be represented as:

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

$$A^{[1]} = g_{\text{ReLU}}(Z^{[1]})$$

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

$$A^{[2]} = g_{\text{softmax}}(Z^{[2]})$$

where g_{ReLU} represents the ReLU activation function, and g_{softmax} represents the softmax activation function.

3. Backward propagation

Backpropagation computes the gradients of the loss function with respect to the weights and biases, using the chain rule of calculus. Assuming we have a gradient vector ∇ and activation values A, the backward propagation step can be represented as:

$$dZ^{[2]} = A^{[2]} - Y$$

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$$

$$dB^{[2]} = \frac{1}{m} \sum dZ^{[2]}$$

$$dZ^{[1]} = W^{[2]T} dZ^{[2]} .* g^{[1]\prime}(Z^{[1]})$$

$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^{T}$$

$$dB^{[1]} = \frac{1}{m} \sum dZ^{[1]}$$

where $\cdot$ represents element-wise multiplication, m represents the number of training examples, and $g^{[1]\prime}$ represents the derivative of the ReLU activation function.

4. Parameter updates

The weights and biases are updated based on the gradients computed in the backward propagation step. Let's denote the learning rate as α.

$$W^{[2]} = W^{[2]} - \alpha dW^{[2]}$$

$$b^{[2]} = b^{[2]} - \alpha dB^{[2]}$$

$$W^{[1]} = W^{[1]} - \alpha dW^{[1]}$$

$$b^{[1]} = b^{[1]} - \alpha dB^{[1]}$$

III. TRAINING NEURAL NETWORKS USING FIREFLY :

1. Initialize the population:

The population of fireflies represents the candidate solutions for optimizing the neural network. Each firefly is characterized by its position vector, which corresponds to the network's weights and biases. The initialization can be represented as:

$$\text{Initialize fireflies' positions:} \quad X^{(0)} = \text{random initialization}$$

2. Evaluate fitness: The fitness of each firefly is evaluated based on the performance of the corresponding neural network. This is typically measured using a fitness function, such as the accuracy or mean squared error. The fitness evaluation can be represented as: Evaluate fitness for each firefly:

$$\text{fitness}(X^{(0)}) = \text{neural network performance}$$

3. Main loop:

The main loop of the Firefly Optimization algorithm iteratively updates the positions of fireflies based on their attractiveness and moves them towards better solutions. The loop can be represented as:

Algorithm 1: Firefly Algorithm

1: **Input:**
- num_fireflies: number of fireflies
- num_dimensions: number of dimensions
- num_iterations: number of iterations
- alpha, beta, gamma: algorithm parameters

2: **Output:**
- best_firefly: firefly with the highest intensity
- best_intensity: intensity of the best firefly

3: **Initialization:**
4: fireflies = generate_initial_population(num_fireflies, num_dimensions)
5: intensities = array of zeros of size num_fireflies
6: **Recursion:**
7: **for** $iteration = 1$ **to** $num\_iterations$ **do**
8: **for** $i = 1$ **to** $num\_fireflies$ **do**
9: x, y = fireflies[i]
10: intensities[i] = objective_function(x, y)
11: **end for**
12: fireflies = move_fireflies(fireflies, intensities, alpha, beta, gamma)
13: **end for**
14: **Termination:**
15: best_index = index of the firefly with the lowest intensity in intensities
16: best_firefly = fireflies[best_index]
17: best_intensity = intensities[best_index]
18: **Output:** best_firefly, best_intensity

4. Select the best solution:

After the main loop, the best solution is selected based on the highest fitness value achieved. This corresponds to the firefly with the best-performing neural network. The best solution can be represented as:

Select the best firefly solution: $X^* =$ firefly with highest fitness

5. Obtain the optimized neural network:

Finally, the optimized weights and biases of the neural network are obtained from the best solution. These optimized values can be used for making predictions or further analysis. The optimized neural network can be represented as:

IV. Experimentation

In this we will represent the comparison between the firefly algorithm and the gradient descent algorithm for the architecture of neural networks, the activation function and time expanded and the accuracy :

The architecture we used is a 4 neurones input layer linked with one dense layer of 10 neurones and finally a 3 neuron output layer:

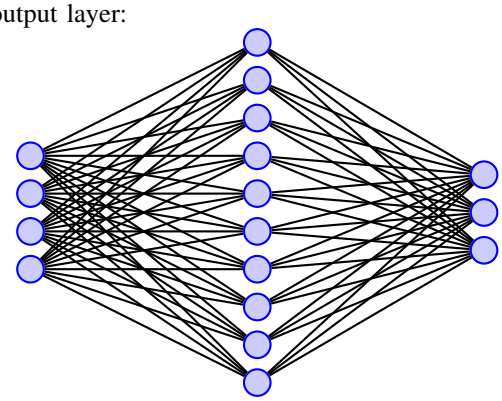

TABLE I
COMPARISON BETWEEN FIREFLY ALGORITHM AND GRADIENT DESCENT ALGORITHM

algorithm	Gradient Descent	FireFly
Time Expended	**0.2s**	**5m11s**
Accuracy	**0.88**	**0.6458**

TABLE II
FIREFLY NUM_FIREFLIES HYPERPARAMETER INFLUENCE ON ACCURACY AND EXECUTION TIME

Algorithm	**FireFly**	**FireFly**	**FireFly**
activation	Relu	ReLu	Relu
num_fireflies	**5**	**20**	**50**
Time Expended	12.1	1m15s	5m11s
Accuracy	0.4387	0.593	0.6458

TABLE III
FIREFLY NUM_ITERATIONS HYPERPARAMETER INFLUENCE ON ACCURACY AND EXECUTION TIME

Algorithm	**FireFly**	**FireFly**	**FireFly**
activation	Relu	ReLu	Relu
num_fireflies	20	20	20
num_iterations	**100**	**1000**	**10000**
Time Expended	8.7s	1m14s	13m4s
Accuracy	0.5725	0.5908	0.6083

TABLE IV
FIREFLY α HYPERPARAMETER INFLUENCE ON ACCURACY AND EXECUTION TIME

Algorithm	**FireFly**	**FireFly**	**FireFly**
activation	Relu	ReLu	Relu
num_fireflies	20	20	20
num_iterations	1000	1000	1000
α	**0.01**	**0.2**	**2**
Time Expended	1m13s	1m15s	1m10s
Accuracy	0.6012	0.6074	0.5608

TABLE V
FIREFLY β HYPERPARAMETER INFLUENCE ON ACCURACY AND EXECUTION TIME

Algorithm	**FireFly**	**FireFly**	**FireFly**
activation	Relu	ReLu	Relu
num_fireflies	20	20	20
num_iterations	1000	1000	1000
α	0.2	0.2	0.2
β	**0.01**	**1**	**10**
Time Expended	1m13s	1m9s	1m13s
Accuracy	0.5654	0.6220	0.5520

V. Conclusion

The comparative analysis between the Firefly Optimization Algorithm (FOA) and gradient descent for optimizing Neural Networks revealed distinct performance characteristics. While gradient descent demonstrated superior performance on small datasets with convex loss functions like mean squared error, the unexplored potential of FOA lies in its efficacy in larger datasets and non-convex loss functions. Future research focusing on the Firefly algorithm's strength in handling extensive datasets and optimizing non-convex loss functions presents a promising direction for further exploration, offering potential insights into its adaptability and efficiency in complex optimization scenarios.

References

[1] Xin-She Yang, "Nature-Inspired Optimization Algorithms".
[2] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz, "Swarm Intelligence: From Natural to Artificial Systems"
[3] Maurice Clerc and James Kennedy, "Particle Swarm Optimization" .