Journal of Machine Learning and Deep Learning (JMLDL)
Volume 1, Issue 1, Date: 31-December-2024
ISSN: 2978-4336, DOI: 10.64820/AEPJMLDL.11.1.9.122024

Optimization of Neural Network Weights with Nature Inspired Algorithm

Yusra Shereen
Department of Computer Science, FAST NUCES, Karachi, Pakistan
Email: yusrashereen@gmail.com

***Abstract*— Neural networks are machine learning algorithms inspired by the human brain regarding structure and function. The artificial neural network (ANN) performs well at tasks on which other conventional approaches fail. They play a crucial role in knowledge representation and learning. The strength of the connection between neurons is determined by weights. Weight optimization is critical to neural networks due to several reasons, more specifically it enables more accurate predictions and reduces loss. A variety of optimization procedures are used for weight optimization. Gradient-based algorithms are the widely used method for optimization of neural network weights, but they are unable to tackle non-differentiable functions. Moreover, gradient-based algorithms may trap in local minima for non-convex functions. Many significant real-world problems have non-convex characteristics, so utilizing gradient descent can cause algorithms to be stuck in local minima. In this paper, we proposed a novel gradient-free approach for optimizing neural network weights utilizing a genetic algorithm. The genetic algorithm is a meta-heuristic algorithm based on the natural evolution process. It can be used to solve both constrained and unconstrained optimization problems. Hence it can solve the problem of convergence for non-differentiable functions and can lead solutions towards global optima. Additionally, we proposed an algorithm to optimize neural network weights utilizing a genetic algorithm. We proved the correctness of our algorithm using the loop invariant technique. Moreover, computational cost analysis is presented for the proposed algorithm. Lastly, we utilized the MNIST dataset for demonstration of our proposed approach. Genetic algorithm capabilities of global search can overcome issues of local minima trapping of non-convex functions.**

Keywords*— *artificial neural network; optimization; weights; gradient free; Neural Network; Genetic Algorithm; neural networks; non convex functions; gradient descent

I. INTRODUCTION

Neural networks are a robust yet complex machine learning approach that has changed numerous fields. They simulate biological neuron and their interactions. They can learn to recognize intricate patterns in data, allowing them to excel at tasks like image recognition, speech processing, and natural language understanding. Neural networks can adapt and improve their performance through training on large datasets. This allows them to handle real-world complexities that rule-based systems struggle with. The activation function decides whether the neuron should be activated or not. The strength between connected neurons is determined and adjusted by weights. In neural networks, weight optimization is crucial for several reasons including improved performance, faster convergence, and efficiency. It helps in optimizing loss, which enables more precise predictions. There are various optimization algorithms that are utilized for weight optimization. The most well–known approach for optimizing NN weights is a gradient-based algorithm called as gradient descent. Gradient descent tries to optimize the loss function over a number of iterations utilizing local minima. It assumes that all functions are differentiable, hence it is unable to handle tasks that cannot be differentiated. The number of iterations required by gradient–based algorithms varies depending on the size of the task. Numerous significant issues are nonconvex, hence trying to differentiate them may lead the algorithm to trap in local optima.

Gradient based algorithms are worst in non-convex functions in which they may trap in local minima [1]. Additionally, they are slow as the number of iterations depends upon the problem scale [2]. Many researchers used different algorithms for gradient free weight optimization of neural networks. For neural network training, stochastic gradient descent (SGD) and its variations have gained popularity in recent years. SGD's optimization is more robust to noisy training data than adaptive gradient approaches. However, it has drawbacks related to vanishing gradients, extreme input sensitivity, and a lack of theoretical assurances [3].

In this study, we presented a novel approach to weight optimization of neural networks utilizing a genetic algorithm. The genetic algorithm is nature inspired algorithm and is based on the principle of natural selection, the mechanism that drives biological evolution, forms the basis of the genetic algorithm. It's a method for overcoming problems with limited and unconstrained optimization. The genetic algorithm is used to modify a population of individual solutions continuously.

A genetic algorithm is a metaheuristic optimization algorithm that can be used to solve both constrained and unconstrained optimization problems. Hence it can solve the problem of convergence for non-differentiable functions and can lead solutions towards global optima. We have utilized the power of genetic algorithm for optimization of neural network weights. As individual solutions are refined at each generation at the end our solution converges towards global

Received: 29-07-2024
Revised: 27-08-2024
Published: 31-12-2024

optima and we get optimal solutions, so it overcomes the problem of gradient-based optimization. The main contributions of this paper are as follows:

- We developed the neural network with gradient-free optimization utilizing a genetic algorithm.
- Proposed novel algorithm for gradient-free optimization-based neural network utilized for image classification.
- Demonstrated the use of gradient-free optimization-based neural network utilizing MNIST dataset.

This paper is divided into nine sections. In the next section relevant reviewed literature is presented. In the third section, the Genetic Algorithm is discussed along with different parent selection strategies. In the fourth section, our proposed approach is discussed. The correctness of the algorithm is proved in the fifth section utilizing the loop invariant method. The computational cost analysis is presented in the sixth section. The seventh section presents the methodology which is followed by results and findings presented in the eighth section. The final section concludes the study, emphasizing the significance of the research and its potential implications.

II. LITERATURE REVIEW

In neural networks, weight optimization is an integral part, as it reduces losses due to which more accurate predictions are possible. For optimization of weights, different optimization algorithms are used.

Gradient based algorithm (gradient descent) is the most recognized algorithm for the optimization of NN weights. Gradient based algorithm considers every function as differentiable and so it fails for non–differentiable functions. Gradient Descent is worst in non–convex functions in which it may stuck in local minima [1]. Gradient based algorithms are slow as the number of iterations depends upon the problem scale [2]. Stochastic Gradient Descent (SGD) and its variants have been popular in recent years for neural network training. Results show that SGD optimization is more resistant to noisy training data than its adaptive gradient techniques [5][12]. However, it suffers from limitations of vanishing gradients, excessive sensitivity to input, and a lack of theoretical guarantees [3]. To overcome these limitations, alternative algorithms for optimization of neural network weights have attracted fast-increasing attention.

It is possible to use Local Search (LS) for a derivative-free, single–candidate optimization of neural networks. In the LS algorithm, portions of the search space are iteratively subjected to limited noise. According to the results stated by the author, LS was able to converge to a lower loss than SGD even if it was not competitive in terms of convergence speed [4]. Additionally, though with lesser performance, LS trained the convolutional neural network (CNN) using Accuracy rather than Loss as a learning signal. LS offers a workable substitute when SGD fails or is not appropriate. A Random Search Optimization (RSO) which is a gradient free Markov Chain Monte Carlo search–based approach can also be used. RSO investigates if adding a perturbation to a weight in a deep neural network reduces the loss on a mini-batch. If doing so lessens the loss, the weight is updated; otherwise, the current weight is kept. When comparing the number of weight updates, RSO converges orders of magnitude quicker than backpropagation. Even still, RSO's training time scales linearly with the number of parameters in the neural network because it computes the function for each weight update [9].

Evolutionary techniques can be used in place of randomization. In evolutionary algorithms (EAs), fundamental ideas from evolutionary biology, such as inheritance, random variation, and selection, are incorporated into algorithms that are used to solve challenging computer issues. Numerous challenging problems from a wide range of areas can be solved with EA approaches, and they can also create machine intelligence that is competitive with humans [10]. EAs also offer several significant advantages over common machine learning techniques, such as less reliance on the presence of a known or discoverable gradient within the search space, the ability to handle design problems where the goal is to create new entities entirely [11]; the ability to solve issues where human expertise is very restricted, support for interpretable solution representations, support for numerous objectives, and seamless integration of human expert knowledge. Grey Wolf Optimization (GWO) which is a gradient-free nature inspired algorithm can also be used for the optimization of neural network weights. Elman Neural Network (ENN) which can memorize past information can be used to solve the stock problem with GWO for the optimization of parameters [6].

The author showed empirically that ENN with GWO as an optimizer provides a more accurate prediction than benchmark models. The authors also stated that GWO surpasses the grasshopper optimization algorithm in adjusting the parameters of the neural network [7]. Hybrid Wolf–Bat algorithm which is a hybrid of two recently developed nature inspired algorithms, performs better than other bio-inspired algorithms in terms of accuracy and convergence speed [8]. Particle Swarm Optimization (PSO) is a metaheuristic algorithm that draws inspiration from the collective behavior of social organisms including ant colonies, fish schools, and bird flocks. This algorithm mimics how members communicate and share information. Numerous optimization problems have been solved using particle swarm optimization, both alone and in conjunction with other current algorithms. Through agents, sometimes known as particles, whose trajectories are modified by a stochastic and a deterministic component, this method searches for the best possible solution [13]. PSO can also be used for gradient free optimization of neural networks. PSO algorithms are skilled in both exploration and exploitation and can address simultaneous adaptation in each NN component [14][15]. PSO can quickly converge CNN architecture with performance [16][17]. In [18] researchers focus on the prediction of utilization of cloud resources using a Functional Link Neural Network (FLNN) with a hybrid Genetic Algorithm (GA) and Particle Swarm Optimization (PSO). They tested the proposed strategy on Google cluster trace data, and the results of their experiments revealed that it is more accurate than more conventional methods. Another nature inspired algorithm Ant Colony Optimization (ACO) can also be used for the optimization of neural network

weights. [19][20][21] uses ACO for the optimization of neural networks and showed that ACO optimized neural network predicts more accurately with the lowest error. A thorough review of the literature and software repositories indicated that many authors used partial gradient–based approaches in the name of gradient free approach for optimization of neural network weights. Some authors also used different evolutionary algorithms or nature inspired algorithms for optimization, which results in better accuracy.

III. ALGORITHM

To optimize neural network weights, this paper proposes a gradient-free approach based on a genetic algorithm. The genetic algorithm is based on natural selection the process that derives biological evolution. It is a technique for resolving both constrained and unconstrained optimization issues. A population of individual solutions is repeatedly modified by the genetic algorithm. The genetic algorithm chooses individuals of the present population as parents at each stage and employs them to produce the offspring that will make up the following generation. The population "evolves" toward the best solution over subsequent generations. The genetic algorithm can solve many optimization problems, including those where the objective function is discontinuous, non–differentiable, stochastic, or highly nonlinear and are not well suited for standard optimization algorithms. The genetic algorithm is presented below:

1) Initialize initial population
2) Compute the fitness of the population
3) do
   - Select fittest candidates as parents
   - Offspring generation using crossover and mutation
   - Compute the fitness of the population

   While! (Termination criteria fulfilled)
4) Stop

Termination criteria are the most important aspect of genetic algorithms. The three termination criteria used for GA are maxed number of generations, convergence-based, and predefined value based. Fig 1 is the flowchart of the standard genetic algorithm. Parent selection is a very crucial step in the convergence of genetic algorithm.

It is vital to maintain diversity in the population to avoid premature convergence. There are several techniques for parent selection in genetic algorithm. Fitness proportionate selection, tournament selection, and rank based selection are the most common.

A. Parent Selection Strategies

Parent selection is a very crucial step in the convergence of genetic algorithm. It is significant to maintain diversity in the population to avoid premature convergence. There are several techniques for parent selection in genetic algorithm. Fitness proportionate selection, tournament selection, and rank based selection are most common.

i. Fitness Proportionate Selection

Fitness proportionate selection is the most popular technique for parent selection. In this, every individual has the probability to become a parent which is proportional to its fitness. So, the fitter individuals have higher chances of mating, and their features will propagate to the next generation. Two strategies for its implementation are:

1) Roulette Wheel Selection

Roulette wheel selection is also known as fitness proportionate selection, in which the probability of selection is assigned to each individual based on their fitness level relative to the total fitness of the population. A circular is divided into n portions or pies, where each portion is the fitness of individuals and n is the number of individuals. The wheel is rotated to select individuals, with those having higher fitness occupying broader segments and thus having a higher probability of being chosen. The fittest individuals occupy a dominant portion of the roulette wheel, so they have more chances of selection.

2) Stochastic Universal Sampling

Stochastic universal sampling is like roulette wheel selection but instead of one fixed point multiple fixed points are chosen. All the parents are selected in just one spin.

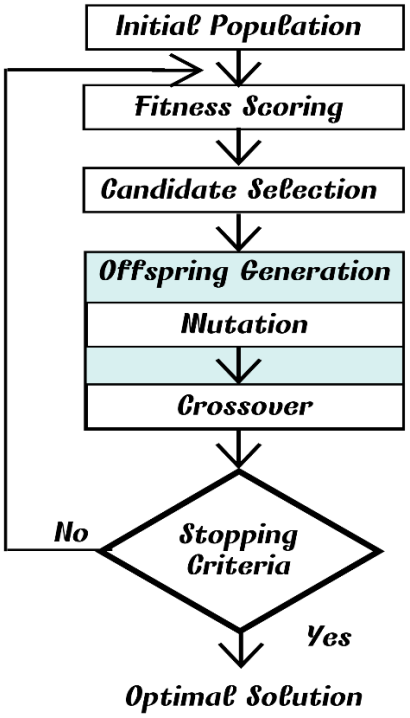

Fig. 1. Flowchart of genetic algorithm.

ii. Tournament Selection

In tournament selection, K individuals are randomly selected from the population for the tournament, and the best out of these are selected for mating. The process is iterated multiple times until all the fittest candidates are selected for mating.

iii. Rank Based Selection

Rank based selection works well when individuals have very close fitness values. Everyone has a portion of the pie as per their rank. All individuals are ranked as per their fitness scores, so every individual has an almost equal share of the pie.

iv. Truncation Selection

Truncation selection is the fundamental technique in this context, but its application is relatively limited in practice. It orders the candidate solutions and then selects n fittest individuals.

B. Offspring Generation

In every search algorithm, two main strategies are used to search, exploration and exploitation. In exploration, entirely new regions of search space are explored. While in exploitation, solutions that exist in already explored regions are searched. Two operators that are used for offspring generation in the genetic algorithm are:

1. Mutation

The mutation operator increases the structural diversity of the population. It randomly modifies one or more genes of chromosomes for reproduction of offspring. It helps in exploring new solutions, which increases genetic diversity and prevents premature convergence. It implements the exploration operator and widens the search space.

2. Crossover

The crossover operator combines the genes of parents for reproduction. One-point crossover and two-point crossover are widely used crossover operators. In a one-point crossover, randomly one point is selected less than the length of chromosomes, and genes are exchanged at this point. Two-point crossover is the same as a one-point crossover but instead of one point two points are selected randomly and then their genes are swapped. Most solutions generated from crossover exist in the exploitation zone, so the crossover operator implements exploitation.

IV. NEURAL NETWORK WEIGHTS OPTIMIZATION UTILIZING GENETIC ALGORITHM

Weight optimization is important in neural networks because it lowers losses, which makes it feasible to make more accurate predictions. Different optimization algorithms are employed for weight optimization. The most used approach for NN weight optimization is a gradient–based algorithm (gradient descent). The gradient-based technique considers every function to be differentiable, hence it fails for functions that are not differentiable. The number of iterations in gradient-based algorithms is scale–dependent and extremely sluggish. Numerous significant issues are non–convex, where an algorithm may become stuck in a local optimum. The worst non-convex functions for gradient-based algorithms are those where they may become stuck in local minima. In this paper, we proposed a gradient free weights optimization technique based on a genetic algorithm. The genetic algorithm is based on Darwin's theory of natural evolution, which is based on the notion that all species are connected and undergo slow evolution. The process of natural selection is what drives evolution, which is the change in a species' features over multiple generations. The population of the genetic algorithm is a set of all individuals. Each individual is represented in chromosome a bit string for genotypic representation, which is a depiction of the DNA structure of species. So, the population of individual solutions represents a population of species that evolves with time. Fig 2 illustrates the representation of genes, chromosomes, and population. Gene is the specific character of the chromosome, and the value of that character is allele. A chromosome is represented as a bit string array and a population is a set of chromosomes.

Fig. 2. Representation of gene, chromosome, and population.

Fig 3 is the architecture of a neural network that uses a genetic algorithm for weight optimization. This neural network is used for handwritten character recognition on the MINIST dataset. In genetic algorithm, fitness evaluation is dependent on the objective function of the problem. For optimization of neural network weights, two possible objectives are minimizing error and maximizing accuracy. Algorithm 1 outlines the pseudocode of the proposed optimization of neural network weights using a genetic algorithm. Steps 1 – 7 are related to the inputs and outputs of the system. In the next step, the neural network is initialized. Step 9 evaluates the fitness of the initialized neural network. The while loop in steps 10 – 14 optimizes neural network weights utilizing a genetic algorithm. After the execution of this algorithm, neural network weights and biases will be optimized.

Algorithm 1: Neural network weights optimization using GA

Input:

1. Training dataset
2. Chromosome length = 10
3. Population size = 10
4. Termination criteria = convergence based
5. Parent selection technique = binary tournament
6. Reproduction method = one point crossover and flip mutation

Output:

7. Neural network with optimized weights and biases.

8. Initialize neural network
9. Evaluate fitness of initial population
10. **While** !(termination_criteria_met) **do**
11. Select parents using fittest individuals
12. Generate offspring with selected parents using reproduction method
13. Calculate fitness of generated offspring
14. Append offspring candidates in population
end

V. ALGORITHM CORRECTNESS PROOF

The algorithm correctness will be proved using loop invariant. The three termination criteria used for GA are max number of generations, convergence based and predefined value based [3]. Therefore, the goal of termination criteria is to stop algorithm after reaching the maximum fitness. As there are three different termination criteria of GA, so termination and maintenance part of loop invariant will slightly differ in each criterion. So, correctness of GA using loop invariant is proved differently. As GA is optimization algorithm, it can be considered correct only if the solution of genetic algorithm is optimal.

A. Convergence based

The algorithm is terminated after solution is converged or when there is no change in population for X generations.

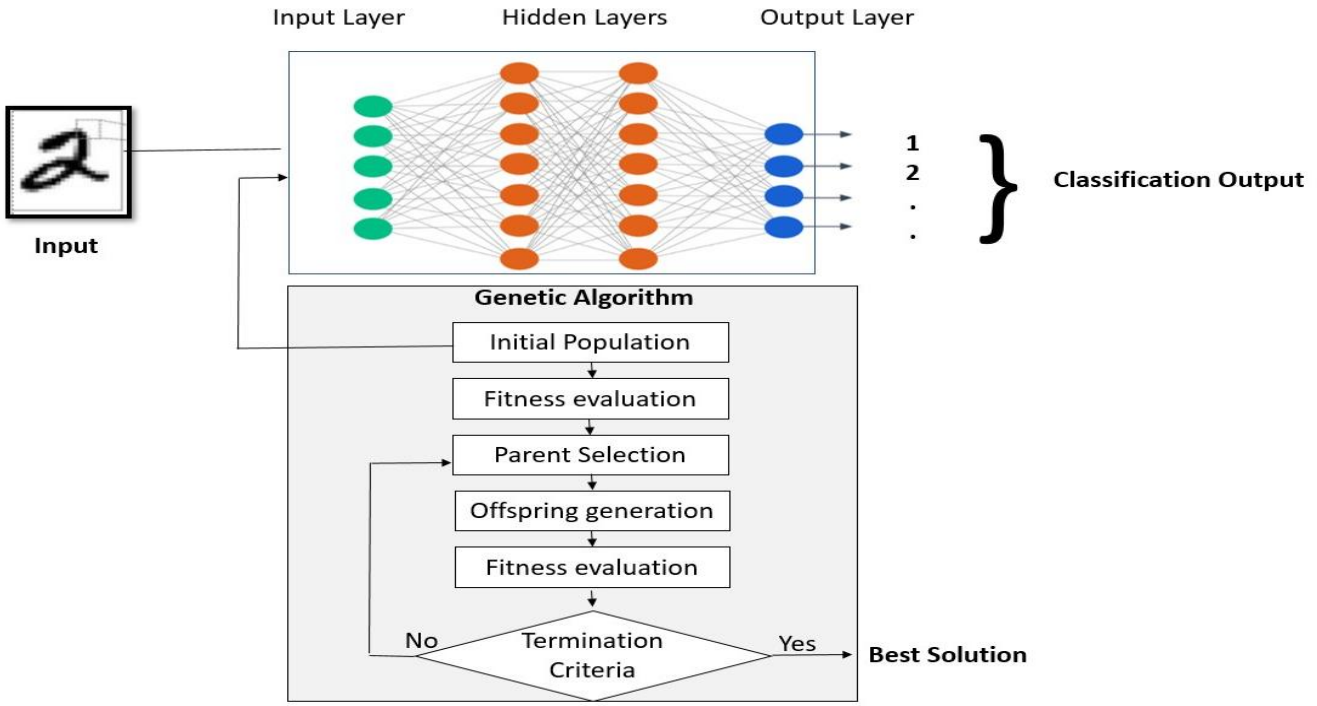

Fig. 2. Architecture of classification model with gradient free weight optimizer.

Counter variable c is used to count number of generations in which population does not change.

1) Loop Invariant

At the start of iteration j of the loop, the population only consists fittest individuals (solutions) which survived from 1st generation to jth generation.

2) Initialization

At start of loop, the initial population contains randomly generated solutions, which are then evaluated using fitness function. As this is 1st generation so all the individuals are fittest. Counter variable c is initialized with zero.

3) Maintenance

Assume that loop invariant holds at start of iteration j. Then the current generation must have fittest individuals (solutions), which survived from 1st generation to jth generation. In loop body of jth iteration, parents are selected from population using selection criteria. Offsprings are generated with parent chromosome using crossover and mutation. Fitness of offsprings and jth generation is evaluated, individuals who passes this evaluation proceed to next generation. If offsprings did not survive after evaluation increment counter variable c. Thus, at start of j+1 iteration the population only fittest individuals which survived from 1st generation (initial population) to j+1 iteration, which is what we are proving that at the end of each generation, only optimal individuals (solutions) survive.

4) Termination

The loop terminates when solution converges or c is equal to X. At the end of n generations, the loop invariant states: The solution is most optimal solution that survived from 1st to nth generation and the population remains same for X iterations. Because if any other optimal solution exists then it should survive and should be a part of population which remains same for at-least X generations so, the current best solution is optimal solution, which is what we are proving that at the end of X generations, the solution is optimal solution. Therefore, the algorithm is correct.

B. Maximum Number of Generations

The algorithm is terminated after reaching n number of generations.

1) Loop Invariant

At the start of iteration j of the loop, the population only consists fittest individuals (solutions) which survived from 1st generation to jth generation.

2) Initialization

At start of loop, the initial population contains randomly generated solutions, which are then evaluated using fitness function. As this is 1st generation so all the individuals are fittest.

3) Maintenance

Assume that loop invariant holds at start of iteration j. Then the current generation must have fittest individuals (solutions), which survived from 1st generation to jth generation. In loop body of jth iteration, parents are selected from population using selection criteria. Offsprings are generated with parent chromosome using crossover and mutation. Fitness of offsprings and jth generation is evaluated, individuals who passes this evaluation proceed to next generation. Thus, at start of j+1 iteration the population only fittest individuals which survived from 1st generation (initial population) to j+1 iteration, which is what we are proving that at the end of each generation, only optimal individuals (solutions) survive.

4) Termination

The loop terminates after n generations. At the end of n generations, the loop invariant states: The best solution is optimal solution from 1st to n generations otherwise it would not survive till nth generation, which is what we are proving that at the end of n generations, the solution is optimal solution of n generations. Therefore, the algorithm is correct.

C. Predefined Value Based

The algorithm is terminated after solution is either lesser than or equal to predefined value for minimization problem or greater than or equals to predefined value for maximization problem

1) Loop Invariant

At the start of iteration j of the loop, the population only consists fittest individuals (solutions) which survived from 1st generation to jth generation.

2) Initialization

At start of loop, the initial population contains randomly generated solutions, which are then evaluated using fitness function. As this is 1st generation so all the individuals are fittest.

3) Maintenance

Assume that loop invariant holds at start of iteration j. Then the current generation must have fittest individuals (solutions), which survived from 1st generation to jth generation. In loop body of jth iteration, parents are selected from population using selection criteria. Offsprings are generated with parent chromosome using crossover and mutation. Fitness of offsprings and jth generation is evaluated, individuals who passes this evaluation proceed to next generation. Fitness of each solution is compared with predefined value, which does not fulfill the criteria. Thus, at start of j+1 iteration the population only fittest individuals which survived from 1st generation (initial population) to j+1 iteration, which is what we are proving that at the end of each generation, only optimal individuals (solutions) survive.

4) Termination

At termination of loop the best solution fitness is either lesser than or equal to predefined value for minimization problem or greater than or equals to predefined value for maximization problem. So, the best solution is optimal solution as it terminates after reaching the lower bound for minimization or upper bound for maximization of optimization problem. Therefore, the algorithm is correct.

VI. COMPUTATIONAL COST ANALYSIS

A. Theoretical Analysis

The cost of genetic algorithm is dependent on number of generations, length of chromosomes, and population size. Because genetic algorithm is optimization algorithm, and optimized solution is searched using fitness scores of individuals in population. When convergence–based, and pre–defined value based termination criteria are used the number of generations population evolved is based on objective function. If predefined value-based termination criterion is used and initial population fitness satisfies the pre–defined value constraint, then algorithm will terminate just after 1st generation. It can be called as best case, but it's rare case. As initial population is randomly populated, so getting optimized solution from random solution have rare chances. Even when convergence-based termination criterion is used, and solution did not change in 1st x generations then it is premature convergence. Parent selection technique also effects the computational cost of genetic algorithm. In roulette wheel selection the number of spins is equal to size of initial population or number of parents to be selected while in universal sampling all parents are selected in just one spin. In truncation or elitism selection the cost of sorting solutions will be added. The length of chromosome also effects the cost of genetic algorithm, as chromosomes are represented as bit array. In fitness evaluation and offspring generation operations are performed on chromosomes which is a bit array. Array traversal takes linear time, so the length of chromosomes affects the performance. Another key factor that impacts cost of genetic algorithm is population size.

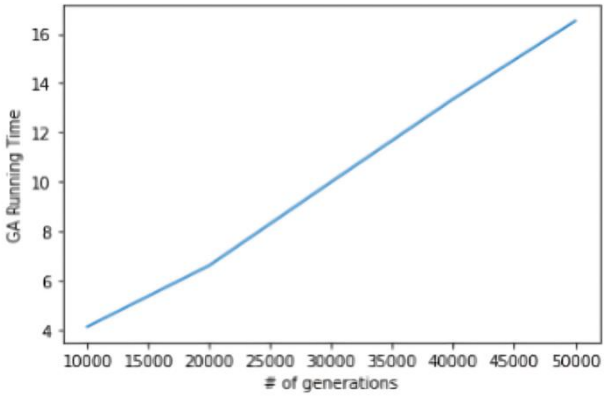

Fig 4. Runtime of a genetic algorithm as a function of the number of generations

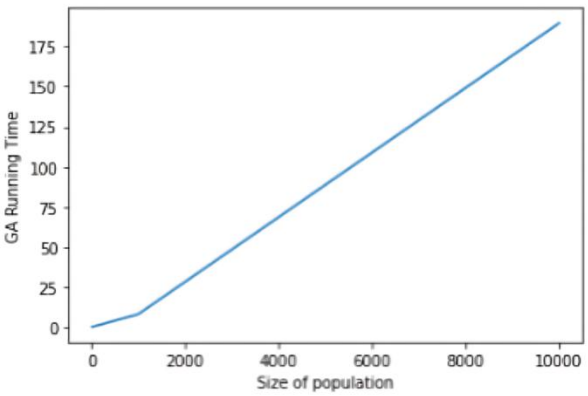

Fig 5. Runtime of a genetic algorithm as a function of the population size

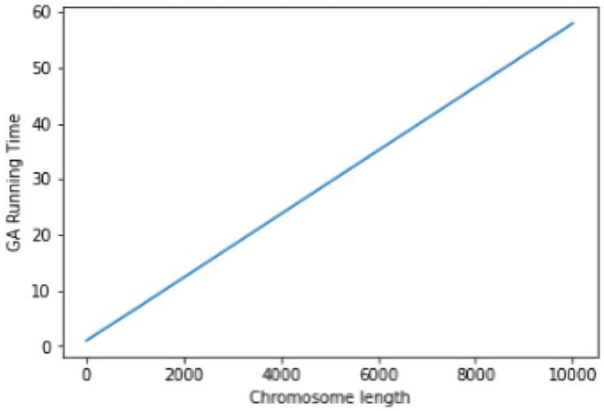

Fig. 6. Runtime of a genetic algorithm as a function of a chromosome length

Population is a set of all individuals or all solutions. Programmatically, population is array of all chromosomes, where each chromosome is a bit array. If chromosome length equals population size, the running time will be quadratic, usually the length of chromosomes do not equal population size. But population size effects the cost of GA. For same objective function, the cost of GA varies as per parent selection technique, operators for offspring generation,

number of generations, population size, and length of chromosomes. For max number of generations termination criterion change in population size and length of chromosomes will change the computational cost. For usual choices one gene mutation, one point crossover and parent selection using roulette wheel the complexity of GA is

$$O(g(nm + nm + n)) + O(f)$$

Which can be simplified as,

$$O(gnm) + O(f)$$

where g is number of generations, n is population size, m is length of chromosomes, and O(f) is fitness evaluating function cost.

B. Empirical Analysis

In the context of this research, we considered using empirical analysis as a preferred method for evaluating performance of stochastic genetic algorithm. For parent selection binary tournament selection is used. For offspring generation 2–point crossover operator, and flip mutation with 20% probability is used. Cost of fitness evaluation function used in this empirical analysis is constant (O(1)). Fig 4 is graph of genetic algorithm running time with respect to number of generations. For different number of generations same population size and chromosome length is used. GA is executed 5 times using 10000, 20000, 30000, 40000, 50000 as number of generations, the chromosome length used is 10 also the population size is 10. For parent selection and offspring generation above specifications are used. Empirical results showed that by increasing number of generations running time increases although chromosome length, population size, parent selection technique, and offspring generation operator remains same. Fig 5 is graph of genetic algorithm running time with respect to size of population. For varied sizes of population same number of generations and chromosome length is used. GA is executed 4 times using 10, 100, 1000, 10000 as population sizes, the chromosome length used is 10 and GA is executed for 1000 generations each time. For parent selection and offspring generation above specifications are used. Empirical results showed that by increasing population size running time increases although chromosome length, number of generations, parent selection technique, and offspring generation operator remains same. Fig 6 is graph of genetic algorithm running time with respect to length of chromosomes. For different chromosome lengths same number of generations and population size is used. GA is executed 4 times using 10, 100, 1000, 10000 as chromosome length, the population size used is 100 and GA is executed for 1000 generations each time. For parent selection and offspring generation above specifications are used.

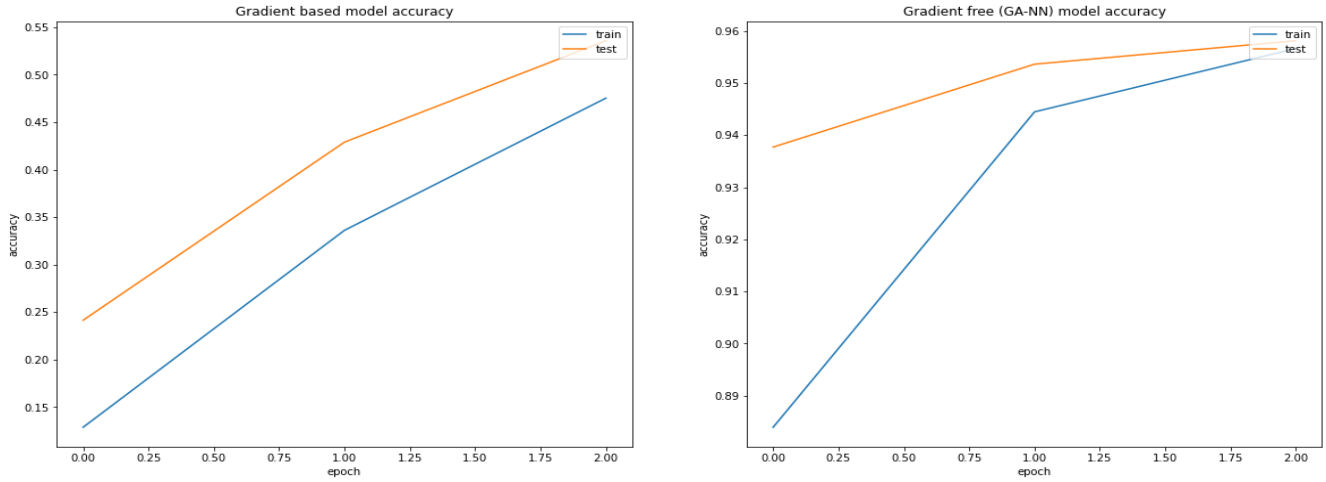

Fig. 7. Training and test accuracies for gradient-based weight optimization and gradient-free weight optimization.

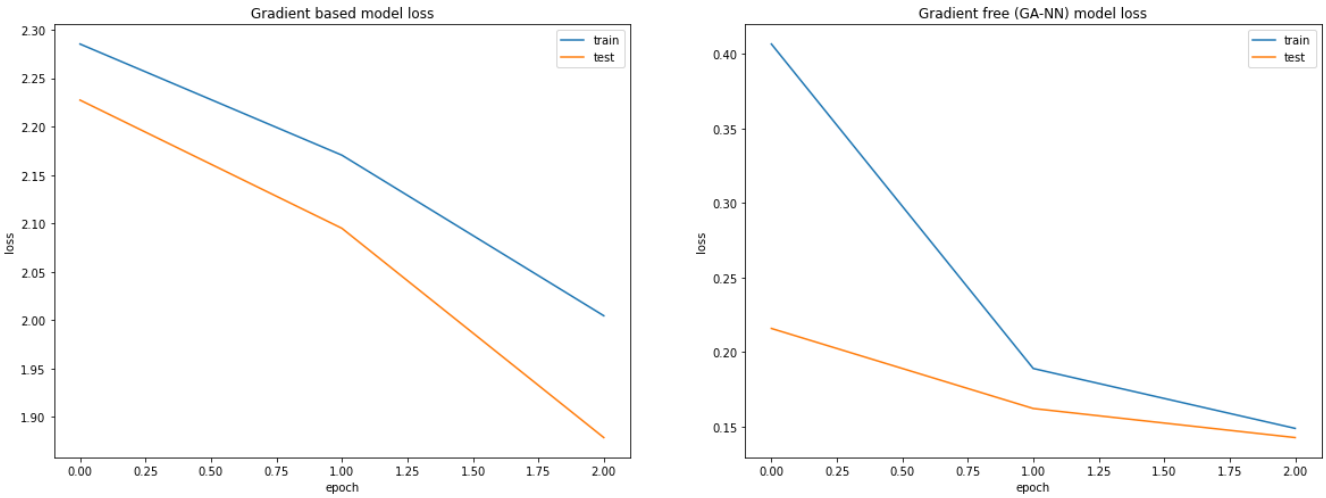

Fig. 8. Training and test losses for gradient-based weight optimization and gradient-free weight optimization.

Empirical results showed that by increasing length of chromosomes running time increases although population size, number of generations, parent selection technique, and offspring generation operator remains same. It is observed during empirical analysis that increase in population size takes more time as compared to increase in generations or increase in chromosome length.

VII. METHODOLOGY

The methodology of proposed neural network weights optimization approach is discussed below:

A. Dataset

In order to assess efficiency of our proposed gradient free approach for optimization of neural network, we utilized MNIST dataset. MINIST dataset is database of handwritten digits has a training set of 60,000 samples, and a test set of 10,000 samples. This publicly available dataset has a great value in deep learning community.

B. Deep Learning Model

The neural network architecture utilized in this study is feed-forward network. The model consists of input layer, single hidden layer, and output layer. Input layer has 32 neurons with ReLU (Rectified Linear Unit) activation function, hidden layer has 64 neurons with ReLU activation function, and output layer has 10 neurons with Softmax activation function.

C. Genetic Algorithm

The genetic algorithm presented in section IV is used for optimization of neural network weights. The initial population consists of 10 individuals, while the length of chromosome is 10. Neural network weights are randomly initialized, and fitness of individuals is evaluated on validation set. Binary tournament is used for parent selection in each generation. One-point crossover and flip mutation is used for offspring generation. The genetic algorithm is terminated when validation accuracy converges.

The model is trained on optimized weights and biases obtained from genetic algorithm. Adam optimizer is used for training of neural network. The performance of proposed approach is evaluated on validation set.

VIII. RESULTS & DISCUSSION

In this study, MINIST dataset is used to classify images with gradient free Neural Network using genetic algorithm and gradient based neural network to assess the efficacy of our proposed approach. MINIST dataset is database of handwritten digits. The feed-forward network is used as neural network architecture in this study. To assess efficacy of our proposed approach we compared results of training same dataset on same model architecture, using gradient based and gradient free approaches.

The base model used in both approaches created with same number of layers, and neurons. The model consists of input layer, single hidden layer, and output layer. Both models are trained for same number of epochs to analyze their efficacy after training for same number of epochs. Both models are trained for 3 epochs with batch size is 100. The genetic algorithm successfully optimized the neural network weights leading to an improvement in validation accuracy in just 3 epochs. Figure 7 shows the training and test accuracies of gradient based and gradient free NN models. It is shown that gradient free optimization based neural network converges faster than neural network that optimizes weights using gradient based approach. The accuracies of both approaches are summarized in TABLE I. The training accuracy of MLP is 52%, and of GA–NN 96% after training both models for 3 epochs. The validation accuracy of GA–NN is 95% and of MLP is 52%. It is shown that gradient free approach outperforms the gradient based approach when trained for same number of epochs. This indicates that the genetic algorithm effectively enhanced the model's performance.

TABLE I. TRAINING & VALIDATION ACCURACIES

	Gradient based optimization	**Gradient free optimization**
Training Accuracy	52%	96%
Validation Accuracy	52%	95%

Figure 8 shows the training and test loss of gradient based and gradient free NN models. It is shown that training and is greater than gradient free GA–NN. The training and validation loss curves of gradient free optimization indicate faster convergence when using optimized weights. Although processing time of gradient free GA-NN is greater than processing time of gradient based MLP, but the performance of GA–NN is much better than gradient based MLP. GA–NN predicts with 95% accuracy after training for 3 epochs. MLP is also trained on same NN architecture with same number of epochs, but its accuracy is 52%. Neural networks performance trained on same dataset is usually evaluated using model's accuracy and its loss during training and testing. Whereas it is desired that accuracy should be highest, and loss should be lowest. To evaluate weights optimization with both approaches, the performance of optimization technique can be evaluated in terms of high accuracy and minimized loss, provided that both models have same neural network architecture and hyperparameters.

Neural network with genetic algorithm as weights optimizer outperforms gradient based neural network. But for some cases when problem is simple it may take more computation time, or it may be slower than gradient based neural network.

IX. CONCLUSION & FUTURE WORK

In this study, we explored the use genetic algorithm for optimization of neural network weights, on MNIST dataset. The results that genetic algorithm can effectively improve the accuracy compared to conventional gradient based approach in lesser number of epochs. Feed-forward neural network with genetic algorithm as weight optimizer can predict with 95% accuracy after training for 3 epochs. Another feed-forward network with same architecture and gradient descent as weights optimizer is trained for same number of epochs but its accuracy is 52%. Cost analysis of genetic algorithm is also performed, and it shows that cost of stochastic genetic algorithm is mainly based on number of generations, chromosome length, population size, and fitness function even if same methods are used for parent selection and offspring generation. Neural network with genetic algorithm

as weights optimizer outperforms gradient based neural network. But for simple use cases it may be slower than gradient based neural network. Many important problems have non–convex nature, in which gradient–based algorithm may trap in local minima. The findings suggest that genetic algorithm can serve as a powerful tool for optimizing neural network weights, particularly in scenarios where traditional optimization techniques may struggle to escape local minima. The observed improvements in model performance highlight the potential of evolutionary algorithms in machine learning applications. In future research, researchers may investigate the potential of other nature inspired algorithms for optimization of neural network weights. They can also analyze the performance of our proposed algorithm on complex neural network architectures such as transformers or recurrent neural networks (RNN) to assess the scalability and effectiveness of this approach. Future research may also explore the methods for optimization of genetic algorithm computational time and making this approach more feasible for large scale applications.

REFERENCES

[1] Datacamp, (March 2022), "Gradient Descent Tutorial" https://www.datacamp.com/tutorial/tutorialgradient-descent

[2] Datajobs, "Gradient Descent", https://datajobs.com/datasciencerepo/Gradient-Descent-[RIT].pdf

[3] Junxiang Wang, Hongyi Li, Liang Zhao, "Accelerated Gradient-free Neural Network Training by Multi-convex Alternating Optimization", Neurocomputing, Volume 487, 2022, Pages 130-143, ISSN 0925-2312, https://doi.org/10.1016/j.neucom.2022.02.03 9

[4] A. Aly, G. Guadagni and J. B. Dugan, "Derivative-Free Optimization of Neural Networks using Local Search," 2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), 2019, pp. 0293- 0299, doi: 10.1109/UEMCON47517.2019.8993007

[5] S. Chaudhury and T. Yamasaki, "Robustness of Adaptive Neural Network Optimization Under Training Noise," in IEEE Access, vol. 9, pp. 37039-37053, 2021, doi: 10.1109/ACCESS.2021.3062990

[6] Kumar Chandar, S. "Grey Wolf optimization-Elman neural network model for stock price prediction," in Soft Computing 25, pp. 649–658, 2021, doi: https://doi.org/10.1007/s00500-020-05174-2

[7] Moayedi, Hossein & Nguyen, Hoang & Foong, Loke. "Nonlinear evolutionary swarm intelligence of grasshopper optimization algorithm and gray wolf optimization for weight adjustment of neural network." Engineering with Computers. 2021, 37. 10.1007/s00366-019-00882-2.

[8] Utkarsh Agrawal, Jatin Arora, Rahul Singh, Deepak Gupta, Ashish Khanna, and Aditya Khamparia. 2020. Hybrid Wolf-Bat Algorithm for Optimization of Connection Weights in Multi-layer Perceptron. ACM Trans. Multimedia Comput. Commun. Appl. 16, 1s, Article 37 (January 2020), 20 pages. https://doi.org/10.1145/3350532

[9] Tripathi, R., & Singh, B. (2020). "RSO: A Gradient Free Sampling Based Approach For Training Deep Neural Networks." arXiv. https://doi.org/10.48550/arXiv.2005.05955

[10] Kannappan, K.; Spector, L.; Sipper, M.; Helmuth, T.; La Cava, W.; Wisdom, J.; Bernstein, O. Analyzing a decade of humancompetitive ("HUMIE") winners: What can we learn? In Genetic Programming Theory and Practice XII; Riolo, R., Worzel, W.P., Kotanchek, M., Eds.; Springer International Publishing: Cham, Switzerland, 2015; pp. 149–166

[11] Sipper, M.; Olson, R.S.; Moore, J.H. Evolutionary computation: The next major transition of artificial intelligence? BioData Min. 2017, 10, 26.

[12] Towards data science, "Various optimization algorithms for neural networks" https://towardsdatascience.com/optimizersfor-training-neuralnetwork-59450d71caf6

[13] Clara Marina Mart´ınez, Dongpu Cao, "Integrated energy management for electrified vehicles," in Ihorizon-Enabled Energy Management for Electrified Vehicles, Butterworth-Heinemann, 2019, pp 15-75, ISBN 9780128150108, https://doi.org/10.1016/B978-0-12-815010- 8.00002-8

[14] Mazaheri, P., Rahnamayan, S., & Bidgoli, A. A. (2022). Designing Artificial Neural Network Using Particle Swarm Optimization: A Survey. In (Ed.), Swarm Intelligence - Recent Advances and Current Applications [Working Title]. IntechOpen. https://doi.org/10.5772/intechopen.106139

[15] Lin, Cheng-Jian, Garro, Beatriz A., Vazquez, Roberto A. (2015), ´ "Designing Artificial Neural Networks Using Particle Swarm Optimization Algorithms", in Computational Intelligence and Neuroscience, Hindawi Publishing Corporation, ISSN - 1687-5265, https://doi.org/10.1155/2015/369298

[16] Francisco Erivaldo Fernandes Junior, Gary G. Yen, "Particle swarm optimization of deep neural networks architectures for image classification," in Swarm and Evolutionary Computation, Volume 49, 2019, pp 62-74, ISSN 2210-6502, https://doi.org/10.1016/j.swevo.2019.05.010

[17] Lapid, R., Haramaty, Z., & Sipper, M. (2022). An Evolutionary, Gradient-Free, Query-Efficient, Black-Box Algorithm for Generating Adversarial Instances in Deep Convolutional Neural Networks. Algorithms, 15(11), 407. https://doi.org/10.3390/a15110407

[18] Malik, S., Tahir, M., Sardaraz, M., & Alourani, A. (2022). A Resource Utilization Prediction Model for Cloud Data Centers Using Evolutionary Algorithms and Machine Learning Techniques. Applied Sciences, 12(4), 2160. https://doi.org/10.3390/app12042160

[19] Hong Zhang, Hoang Nguyen, Xuan-Nam Bui, Trung Nguyen-Thoi, Thu-Thuy Bui, Nga Nguyen, Diep-Anh Vu, Vinyas Mahesh, Hossein Moayedi, Developing a novel artificial intelligence model to estimate the capital cost of mining projects using deep neural network-based ant colony optimization algorithm, Volume 66, 2020, ISSN 0301-4207, https://doi.org/10.1016/j.resourpol.2020.101 604.

[20] Solomon Netsanet, Dehua Zheng, Wei Zhang, Girmaw Teshager, Shortterm PV power forecasting using variational mode decomposition integrated with Ant colony optimization and neural network, Energy Reports, Volume 8, 2022, Pages 2022-2035, ISSN 2352-4847, https://doi.org/10.1016/j.egyr.2022.01.120

[21] Xiaobo Zhao, Dongji Xuan, Kaiye Zhao, Zhenzhe Li, Elman neural network using ant colony optimization algorithm for estimating of state of charge of lithium-ion battery, Journal of Energy Storage, Volume 32, 2020, 101789, ISSN 2352-152X, https://doi.org/10.1016/j.est.2020.101789