Hybrid GD-SA Algorithm

Yun You

2024

[1]

1 Introduction

In this paper, we will discuss the following optimization problem,

$$\min f(x) \quad \text{where} \quad f : R^n \to R \tag{1}$$

Where min $f(x)$ is stand for the global minima, and we assumes that $f(x)$ is continuously differentiable.

In class, we have learned about both gradient descent-based models and simulated annealing-based models. Gradient descent is an optimization algorithm for finding a local minimum of a differentiable function. SA is a stochastic global search optimization algorithm, which has a specific acceptance rate to accept new

[1] https://github.com/IvyYY00/Hybrid-GDSA-763-GitHub Repository: Hybrid-GDSA-763

theta values, even if the new performance is worse than the original one. Both of these algorithms have been widely used in optimizing machine learning problems. When trying to solve optimization problems deterministically, the gradient descent method is among the most cost-effective deterministic methods. However, gradient descent may get stuck in local minima, which impedes further optimization, especially in high-dimensional non-convex optimization problems, like in deep learning. Instead of using the single gradient descent method, We will therefore mixing two single algorithms together to build custom optimizer to improve the performance. In other words, we will introduce a ensemble method GD-SA which stand for gradient descent improved by simulated annealing. This offers the model the ability to probabilistic overcome obstacles, enabling the model to potentially escape local minima and converge to an optimal state.

The main contributions of this work are to develop and assess a hybrid optimization method that combines gradient descent with simulated annealing, leveraging the strengths of both algorithms. This involves determining the optimal times to use gradient descent and simulated annealing, investigating the influence of parameters such as stopping threshold, temperature decreasing rate, and gradient learning rate. It also includes analyzing their convergence ability and evaluating performance. The aim is to enhance the performance of the hybrid algorithm in terms of convergence speed, accuracy, and ability to escape local optima compared to standard gradient descent. Then, we will evaluate the robustness and generalization ability of this hybrid approach across various types of optimization problems, ensuring its effectiveness in diverse scenarios.

This project is organized as follows: Section 2 reviews the basic concepts of gradient descent and simulated annealing, and introduces the hybrid GD-SA algorithm which is aimed at achieving better performance. In Section 3, we test their performance, make comparisons, and discuss the results and limitations.

2 Hybrid GD-SA

2.1 Setup and Notation

We consider the following general situation: Gradient descent follows the equation

$$\theta_{k+1} = \theta_k - \alpha_k \nabla f(\theta_k) \tag{2}$$

where $\nabla f(\theta_k)$ is the gradient vector of the function f at point θ_k, and α_k is the step size.

Simulated annealing is described by

$$\theta_{k+1} = \theta_{\text{ac}} + d_k \tag{3}$$

where θ_{ac} will be accepted if the random probability is less than or equal to $\exp\left(\frac{-\Delta E}{T}\right)$.

We let c and β denote control parameters smaller than 1. Let $\Delta\theta$ denote the Euclidean distance between changing weights, and use $\delta_{\text{threshold}}$ and ϵ to denote the conditions for stopping the gradient descent and the entire algorithm, respectively.

$$\Delta\theta = \|\theta_{\text{next}} - \theta\| \tag{4}$$

$$\Delta E = f(\theta_{\text{next}}) - f(\theta) \tag{5}$$

2.2 Algorithmic Description

2.2.1 Line Search

The gradient $\nabla f(\theta_k)$ always points in the direction of steepest increase in the loss function. The learning rate, α, defines the adjustment in the weights with respect to the loss gradient descent. It determines the convergence ability towards the optimal weights. The quality of gradient descent significantly depends on the choice of learning rate. A too high learning rate may cause the loss function to oscillate around the direction of steepest descent, leading to a zigzag pattern. This inefficiency can result in increased computational time and may even cause the algorithm to diverge. Conversely, a very low learning rate will cause slow convergence, making the loss function prone to getting stuck in local minima. To prevent this situation, we use an adaptive learning rate to ensure the convergence of the loss function.

First-order methods consist of using information from the first-order derivative to update the learning rate. Let us introduce a function that is useful in the following condition:

$$f(x - \alpha\nabla f(x)) \leq f(x) + c\alpha(-\nabla f(x)^T \nabla f(x)) \tag{6}$$

$$\alpha_{\text{new}} = \beta\alpha_{\text{old}} \tag{7}$$

The left side of the equation presents what the loss would be if we were to perform a gradient descent update with the given learning rate α. The right side of the equation indicates the original loss reduced by $c\alpha$ times the dot product of the gradient(The magnitude of gradient). This represents that under the step size α, the reduction in the function value is at least equal to or smaller than $c\alpha(-\nabla f(x)^T \nabla f(x))$, improving the reliability and efficiency of each step. If the condition is not satisfied, the step size α is reduced by multiplying it with β. This method ensures that the gradient descent does not overshoot the global minimum and converges at a reasonable rate.

2.2.2 Hybrid GD-SA

In this part, we will use the line search method to adjust the learning rate in gradient descent calculations and combine this with the simulated annealing method. The algorithm should first use the gradient descent approach to update θ, and calculate the absolute value of the changing loss. If the changing loss is smaller than $\delta_{\text{threshold}}$, we assume that the loss function is approaching a local minimum, and we should use the simulated annealing idea to give θ some perturbation. Unlike the regular simulated annealing, we do not consider the time when the perturbation causes less loss since we believe the loss has been sufficiently decreased during the gradient descent processing. We use P_{accept} calculated by $\exp\left(-\frac{|\Delta E|}{T}\right)$

to determine if the algorithm transitions into simulated annealing processing. T is decreasing throughout the iterations, so at the first several iterations, the algorithm should be more likely to perform the simulated annealing process. Inside the simulated annealing process, we don't check the change in loss; instead, we directly update θ according to $\alpha \times$ gradient. The probability of θ climbing up or down the hill is fifty-fifty.It is determined by the sign of a random variable from 0 to 1 minus 0.5. In other words, If the loss change, ΔE, is $\leq \delta_{\text{threshold}}$ and passes the acceptance probability based on the adjusted Metropolis-Hastings criteria, we transition to the simulated annealing process. A random perturbation is then added to the current θ. The updating equation is as follows:

$$\theta \leftarrow \begin{cases} \theta - \alpha \nabla f(\theta) & \text{if GD} \\ \theta + \text{sign}(\text{rand}(\text{size}(\theta)) - 0.5) \times \alpha \nabla f(\theta) & \text{if SA} \end{cases}$$

After transitioning to simulated annealing processing, we update the temperature as the iterations progress, reducing the likelihood of adding random perturbation as follows:

$$T \leftarrow \max(T \times 0.99, 0.001) \tag{8}$$

The process terminates when the change in θ is smaller than ϵ, a very small threshold, indicating potential convergence to an optimal solution:

$$\text{If } \Delta\theta < \epsilon \text{ and iterations} > 20 : \text{ terminate} \tag{9}$$

Pseudocode

Algorithm 1 Backtracking Line Search

Require: f, gradient, $x, \alpha = 0.9, \beta = 0.5, c = 1e-2$
Ensure: α (optimized step size)

$f_x \leftarrow f(x)$
while $f(x - \alpha \cdot \text{gradient}) > f_x + c \cdot \alpha \cdot (-\text{gradient} \cdot \text{gradient})$ **do**
    $\alpha \leftarrow \alpha \cdot \beta$
    if $\alpha < 1e-4$ **then**
        break
    end if
end while
return α

3 Application

3.1 Application I: Rastrigin Function

We apply the GD-SA algorithm to the Rastrigin function to evaluate its performance in dealing with multiple local minima. The Rastrigin function is the sum of a series of terms, each comprising a parabola and a negated cosine wave. It has an optimal θ at [0,0] corresponding to an optimal loss of 0. Based on this, it's a good choice to help evaluate the algorithm's performance in non-convex problems and its convergence performance. The function's definition is:

$$f(x) = A \cdot n + \sum_{i=1}^{n} \left[x_i^2 - A \cdot \cos(2\pi x_i) \right] \tag{10}$$

Here, $f(x)$ represents the value of the Rastrigin function at point x, A is a

Algorithm 2 Gradient Descent Simulated Annealing with Line Search

Require: f, grad_f, init_theta, $\alpha, \beta, T0$, max_iter, $\epsilon = 1e-4$, $\delta_{\text{threshold}} = 0.1$
Ensure: θ (optimized parameters), theta_updates (history of updates)

$\theta \leftarrow$ init_theta
theta_updates $\leftarrow [\theta.copy()]$
$T \leftarrow T0$
previous_loss $\leftarrow f(\theta)$
no_improvement_count $\leftarrow 0$
for $i \leftarrow 1$ to max_iter **do**
    gradient $\leftarrow$ grad_f(θ)
    $\alpha \leftarrow$ line_search2$(f,$ gradient$, \theta)$
    $\theta_{\text{next}} \leftarrow \theta - \alpha \cdot$ gradient
    $\theta_{\text{change}} \leftarrow \|\theta_{\text{next}} - \theta\|$
    current_loss $\leftarrow f(\theta_{\text{next}})$
    $\delta_E \leftarrow$ current_loss $-$ previous_loss
    if $|\delta_E| > \Delta_{\text{threshold}}$ **then**
        $\theta \leftarrow \theta_{\text{next}}$
        previous_loss $\leftarrow$ current_loss
    else
        $P_{\text{accept}} \leftarrow \exp(-|\Delta_E|/T)$
        rand $\leftarrow$ random()
        if rand $< P_{\text{accept}}$ **then**
            $\theta \leftarrow \theta +$ sign(random() $- 0.5) \cdot \alpha \cdot$ gradient
        end if
    end if
    $T \leftarrow \max(T \cdot 0.99, 0.001)$
    theta_updates.$append(\theta.copy())$
    no_improvement_count $\leftarrow$ no_improvement_count $+ 1$
    if $\theta_{\text{change}} < \epsilon$ and no_improvement_count > 20 **then**
        break
    end if
end for
return θ, theta_updates

constant set to 10, n is the number of dimensions of the input vector x, and x_i is the i-th component of the vector x. Figure 1 illustrates the function, showing both the three-dimensional and one-dimensional versions.

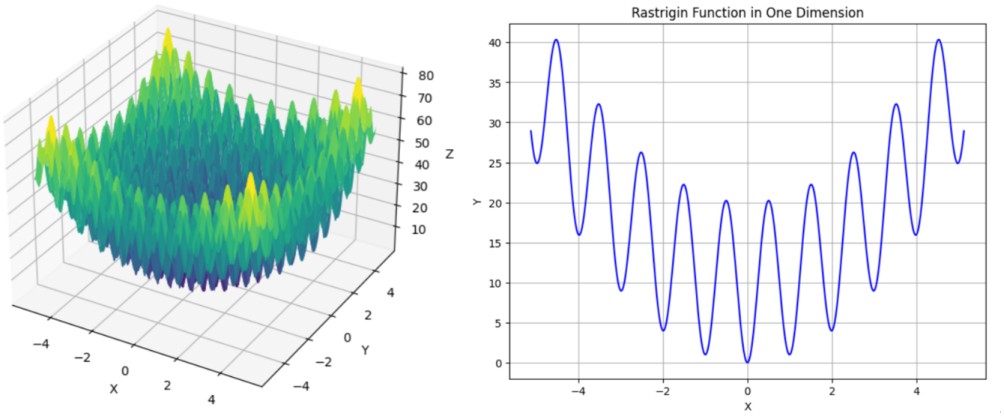

Figure 1: Rastrigin Function

3.1.1 Parameter Choice

In order to find the best parameter combination that balances both running time and performance, we use grid search. The parameters for `init_alpha`, `beta`, `T0`, `max_iter`, `epsilon`, and `delta_threshold` are 1, 0.5, 10, 1000, 1e-4, 0.1 respectively. The initial theta, denoted as $\theta = [\theta_1, \theta_2]$, is randomly chosen from the range $(-5.12, 5.12)$.

3.1.2 Experiment results

In this section, we repeat the experiment 100 times, calculate the frequency of terminal theta, the final loss, and the average running time. We compare our GD-SA algorithm with single Gradient Descent method, simulated Annealing method, and Simple Gradient Descent with line search method.

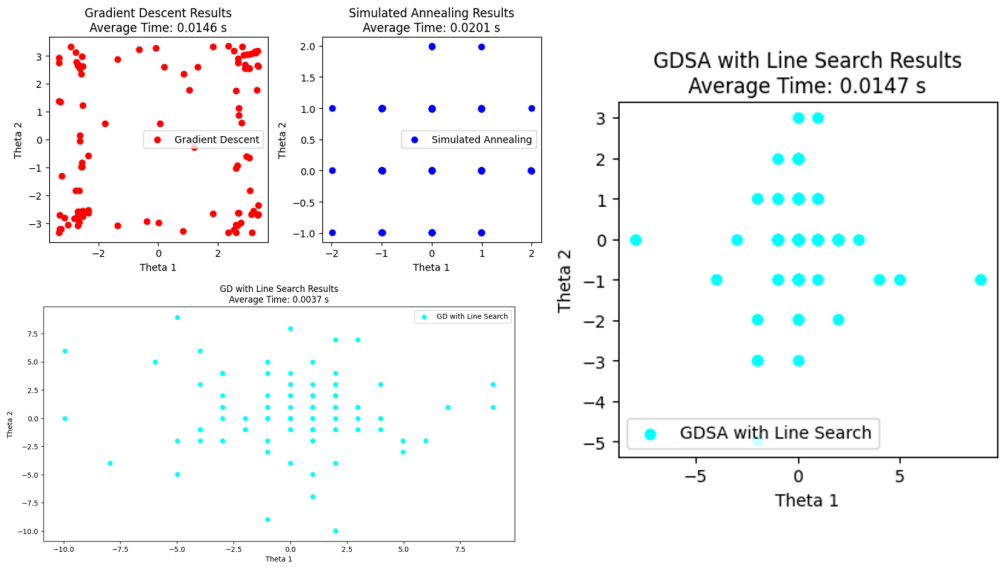

Figure 2: The distribution of terminal θ

In the given figure, we can observe that gradient descent results in a scattered distribution of terminal θ values across the plot, demonstrating its weaker ability to explore the entire space effectively. In contrast, simulated annealing exhibits a stronger capability to thoroughly investigate the space, yielding more accurate results that are closer to 0. However, the running time for simulated annealing (0.0201 seconds) is longer than that for gradient descent (0.0146 seconds), which

aligns with the typical expectations for these methods. By employing the GDSA method, there appears to be a better balance between running time and exploration effectiveness. The terminal results are densely clustered around the optimal solution, although there are some instances where the algorithm does not achieve the global minimum. Overall, the performance strikes a good balance between efficiency and accuracy.

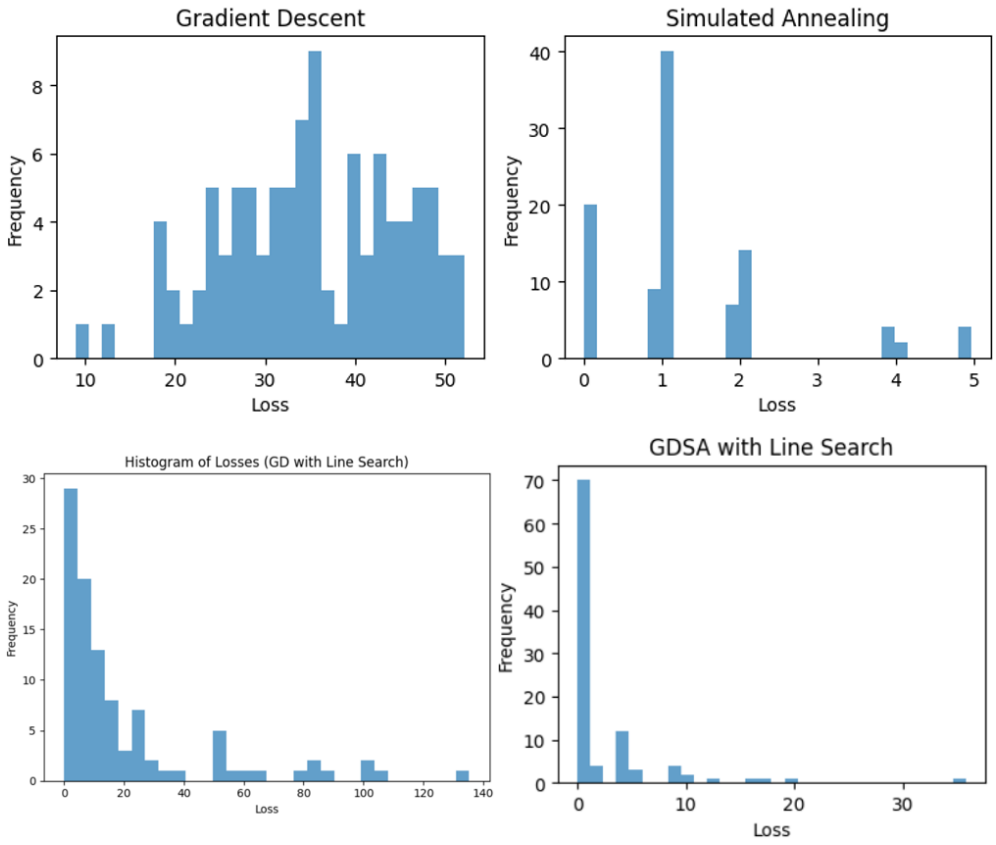

Figure 3: The frequence of final loss

According to figure 3, the histogram for Gradient Descent with Line Search

further shows that while it achieves a broader spread of losses up to 30, there is a notable concentration of outcomes near lower losses, suggesting improved performance over plain gradient descent.

We also tested the performance of simply using gradient descent with line search to further examine whether exploration ability relies greatly on line search or on simulated annealing. The results show that adding line search does provide some improvement in finding the global minimum, but it is still not strong enough to consistently reach the global minimum.

3.2 Application II: Neural network Optimization

In this section, the GD-SA method is further evaluated for adjusting parameters in a neural network. We use the normalized MNIST dataset and simple 1 layer neural networks with a softmax activation function and cross-entropy loss. The gradient is calcuated by chain rule. The softmax function is defined as:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{11}$$

where z_i is the input to the softmax function for the i-th class and the denominator is the sum of exponential values of all inputs.

$$\text{cross-entropy}(y, \hat{y}) = -\sum_i y_i \log(\hat{y}_i) \tag{12}$$

where y is the one-hot encoded true label vector, $\hat{y}$ is the predicted probabil-

ity distribution from the softmax function, and y_i is 1 for the true class c and 0 otherwise.

In backward propagation, we calculated the gradient using chain rule.

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial W} \tag{13}$$

3.2.1 Experiment results

In this section, we compare the performance of using two optimizer methods in backward propagation. One uses simple gradient descent, and the other applies the GDSA algorithm. We plot the loss every 100 epochs to check the convergence speed of the two methods and compare their final losses after 1000 iterations.

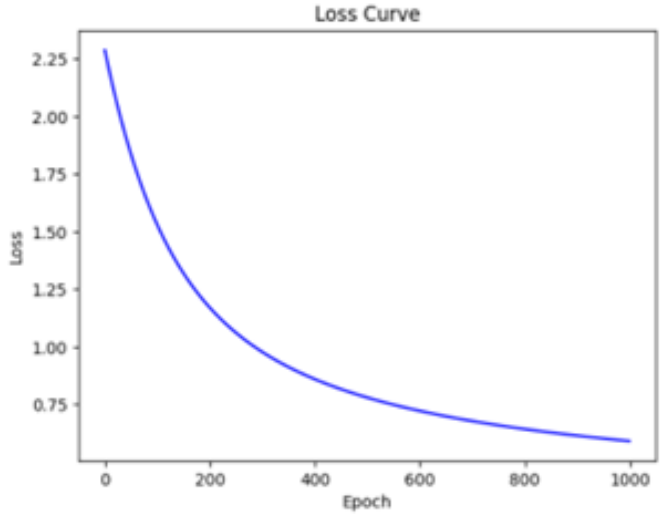

Figure 4: Loss Progress over iterations using Gradient Descent

The final loss for gradient descent is around 0.61, and the final loss for GDSA is around 0.43, showing that the hybrid method is more likely to explore the entire

Figure 5: Loss Progress over iterations sing Gradient Descent

space and reach the global minimum. From Figure 5, we can observe that the loss sometimes increases as the iteration progresses, and finally reaches its best solution, demonstrating the perturbation caused by the simulated annealing method in the algorithm

However, the running time for using GDSA is 100 seconds longer than using the gradient descent method. This increase might be because there are more gradient calculations in GDSA, and as the number of parameters increases, the computational cost for calculating gradients is significantly higher with the hybrid method. In this case, we should avoid applying the GDSA method to cases with high-dimensional spaces. In contrast, the new algorithm is more suitable for problems with limited parameters but very complex non-convex loss functions.

We also applied simulated annealing in isolation for this experiment, although it is not commonly used as the only method in neural network scenarios, and the results showed strong divergence of the loss.

3.3 Comparison

Table 1: Performance Comparison of Optimization Algorithms

Model	**Algorithm**	**Metrics**		
		Final Loss	**Running Time (s)**	**Optimal Solution**
Rastrigin Function	GD	37.76	0.0199	0
	SA	1.39	0.0275	0
	GDSA	4.25	0.0198	0
NN	GD	0.616	106	None
	SA	2.329	27	None
	GDSA	0.439	254	None

4 Future work

In this project, the parameter tuning for the algorithm itself is straightforward, we simply using the grid research. The stopping signals and transition signals are very important in this algorithm. Finding a way that is quick and efficient to adjust parameters to achieve a better balance between exploration and running when facing different problems might be the next challenge regarding this topic.

Some of the calculations might be repeated throughout the entire algorithm. There may be ways to simplify the whole process to reduce redundant computation, or to replace less important calculations to decrease the overall running time.

When developing the algorithm, I found that adding momentum can improve the overall performance of the algorithm but at the cost of a much higher running time. There might be a way to adjust it to achieve better performance in both

convergence speed and accuracy.

In line search, it ensures that each step results in a loss smaller than the previous loss within a certain range. However, it might take steps that are smaller than necessary, which can make it difficult for the subsequent SA (Simulated Annealing) method to escape from local minima. Using a relaxed stochastic line search might help with improvement. Moreover, adding another parameter to the perturbation that multiplies the α * gradient might be another solution to help SA escape from deep local minima and enhance performance

5 Conclusion

We have introduced a hybrid gradient-simulated annealing algorithm that is able to find a balance between computational cost, performance, and generalization ability, leading to better performance on complex non-convex optimization problems. We have demonstrated its application in two specific cases: adjusting parameters in the Rastrigin Function and in neural networks. We show that the new hybrid method has a higher exploration ability to find global minima, and also achieves reasonable convergence times. We acknowledge the limitations of the new algorithm, such as potentially higher costs in high-dimensional spaces compared to gradient descent, and discuss possible ways to improve it.

References

[1] K.A. Alnowibet, S. Mahdi, M. El-Alem, M. Abdelawwad, and A.W. Mohamed, *Guided Hybrid Modified Simulated Annealing Algorithm for Solving Constrained Global Optimization Problems*. Mathematics, vol. 10, p. 1312, 2022.

[2] Zhicheng Cai, *SA-GD: Improved Gradient Descent Learning Strategy with Simulated Annealing*. arXiv preprint arXiv:2107.07558, School of Electronic Science and Engineering, Nanjing University, Nanjing, China 210023, 2021.

[3] M. EL-Alem, A. Aboutahoun, S. Mahdi, *Hybrid gradient simulated annealing algorithm for finding the global optimal of a nonlinear unconstrained optimization problem*. Soft Computing, vol. 25, pp. 2325–2350, 2021. *DOI*: https://doi.org/10.1007/s00500-020-05303-x10.1007/s00500-020-05303-x.

[4] Jingrun Chen, Shi Jin, and Liyao Lyu, *A Consensus-Based Global Optimization Method with Adaptive Momentum Estimation*. arXiv preprint arXiv:2012.04827, December 10, 2020.

[5] Leonardo Galli, Holger Rauhut, and Mark Schmidt, *Don't be so Monotone: Relaxing Stochastic Line Search in Over-Parameterized Models*. Accepted at NeurIPS 2023, RWTH Aachen University, University of British Columbia, arXiv preprint arXiv:2306.12747v2, October 25, 2023.