Trust Region Optimization vs SGD: An Empirical Evaluation with Automatic Hyperparameter Tuning

Final Project Report

Team Name:
Team Members: Linsen Gao
Student IDs: 21134283
Team Members: Haotian Liu
Student IDs: 21173222

April 28, 2025

Abstract

In this project we compared the performance of first order and second order optimization algorithms in aspects of computational efficiency, robustness to hyperparameter, escaping saddle point and generalization performance. We reproduced the paper's conclusion by training one hidden layer MLP on the CIFAR-10 dataset with SGD and TR methods. Experimental results show that TR outperforms SGD in avoiding saddle points and generalization performance, and TR is more robust to hyperparameter changes, but TR has a much higher computational cost because of Hessian. SGD has fast training speed, but it performs poorly in performance stability, especially near saddle points.

1. Introduction

The paper points out problems with the popular SGD optimization algorithm in machine learning, and the advantages of second-order optimization algorithms. The paper proposes two algorithms: sub-sampled TR and ARC.

The paper aims to prove the advantages of second-order algorithms. To achieve the goal, the author set 6 questions. The first 4 questions are to compare the performance between first-order and second-order algorithms by SGD and TR. Question 5 and 6 compare various second-order methods TR, ARC, GN, and L-BFGS. And conclude that TR performs best, while L-BFGS performs worst.

The purpose of our project is to verify the author's point that second-order optimization algorithms have advantages over first-order algorithms in terms of computational efficiency, robustness to hyperparameter, escape from saddle points, and generalization performance.

2. Reproducible Research (Part 1)

2.1. Method Overview

2.1.1 Algorithm used in the paper

Trust Region

At each iteration the TR algorithm solves the following sub-problem:

$$\mathbf{s}_t \approx \underset{\|\mathbf{s}\| \leq \Delta_t}{\operatorname{argmin}} \; m_t(\mathbf{s}) \triangleq \langle \nabla F(\mathbf{x}_t), \mathbf{s} \rangle + \frac{1}{2} \langle \mathbf{s}, \mathbf{H}_t \mathbf{s} \rangle$$

F is the gradient, H is the sub-sampled Hessian approximation, and delta t is the trust-region radius. The sub-problem is solved approximately using the CG-Steihaug method, which is efficient. It usually terminates in a few iterations.

The paper also provides method of Hessian sub-sampling for large scale training.

$$\mathbf{H}(\mathbf{x}) \triangleq \frac{1}{n|\mathcal{S}|} \sum_{j \in \mathcal{S}} \frac{1}{p_j} \nabla^2 f_j(\mathbf{x}),$$

S is a random subset of indices, and p is a sampling distribution (uniform or non-uniform). This sub-sampling reduces computational cost.

Adaptive Regularization with Cubic

ARC is another second order optimization algorithm the paper used. This algorithm use a cubic-regularized term to determine the step. At each iteration it solves:

$$\mathbf{s}_t \approx \underset{\mathbf{s} \in \mathbb{R}^d}{\operatorname{argmin}} \; m_t(\mathbf{s}) \triangleq \langle \nabla F(\mathbf{x}_t), \mathbf{s} \rangle + \frac{1}{2} \langle \mathbf{s}, \mathbf{H}_t \mathbf{s} \rangle + \frac{\sigma_t}{3} \|\mathbf{s}\|^3$$

where sigma is a regularization parameter that controls the cubic term. The subproblem is solved using the generalized Lanczos method, which is less efficient than the CG-Steihaug method. It often requiring more iterations.

2.1.2 Algorithm used in our project

The paper compares first-order and second-order optimization algorithms using SGD and TR. The paper compares ARC with TR to explore differences among second-order methods.

The main objective of our project is to compare the performance of first-order and second-order optimization algorithms in non-convex machine learning tasks. Given the paper's findings that TR outperforms ARC in most experiments. We chose to focus on SGD and TR for our replication experiments.

2.2. Implementation Details

Model and dataset

The paper compared first-order and second-order optimization methods by training a one-hidden-layer MLP for image classification. In our reproduction implementation we adopt the same method and use the CIFAR-10 dataset consisting of 50000 training images and 10000 test images across 10 classes.

To reduce training time while preserving the ability to compare model performance, we randomly selected a subset of 5000 training images from the original training set.

Experimental Settings

Experiments were conducted by using PyTorch. To evaluate the algorithms, we plotted training loss and test error against the total number of propagation.

To simplify the experiment and reduce the training time, we set a maximum number of propagation and selected specific hyperparameter for each experiment and.

- For experiments with random initialization, we chose 10^6 propagation to more clearly distinguish the trend changes in the results of different algorithms. SGD was trained with learning rates 0.05, 0.2 and TR was trained with initial trust region 10, 100.
- For experiments starting from a saddle point, we chose a smaller number of propagation, 10^5, because in theory the difference between first-order and second-order is so large that a small number of propagation can show the difference. And we used 0.01 learning rate for SGD and 1 trust region for TR.

The GitHub link of source code is provided in the Appendix section.

2.3. Results and Comparison

We reproduced two experiments in the paper, including 4 figures, to compare the performance of SGD and TR.

Figure 1 and 3 are the training loss and test error changes when starting training from a random point from the paper. Figure 2 and 4 show the results of reproduced experiments, where the trend between the curves is consistent.

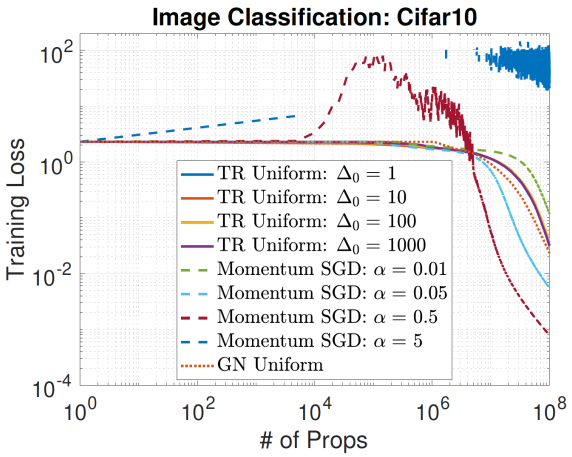

Figure 1: Original random point init training loss

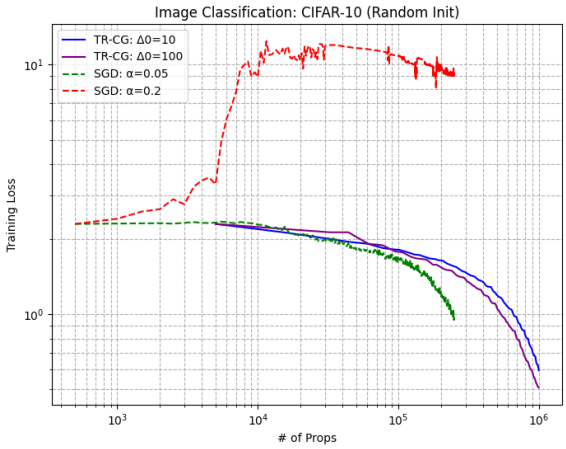

Figure 2: Reproduced random point init training loss

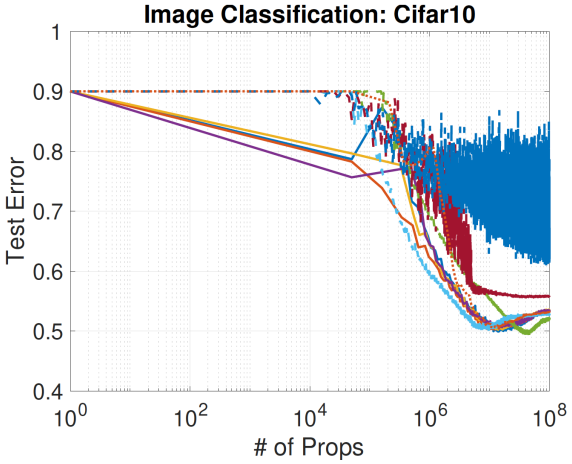

Figure 3: Original random point init test error

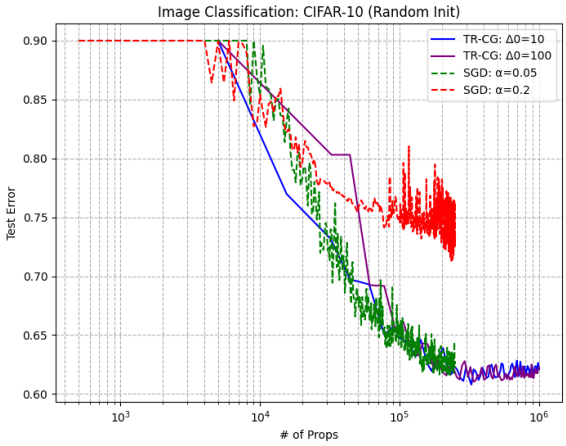

Figure 4: Reproduced random point init test error

Figures 5 and 7 are the paper's results on starting training from a saddle point. Figure 6 and 8 present our reproduced results. Although the number of propagation is relatively smaller, it is clear that our reproduced result is consistent with the paper that SGD performed poorly compared to TR.

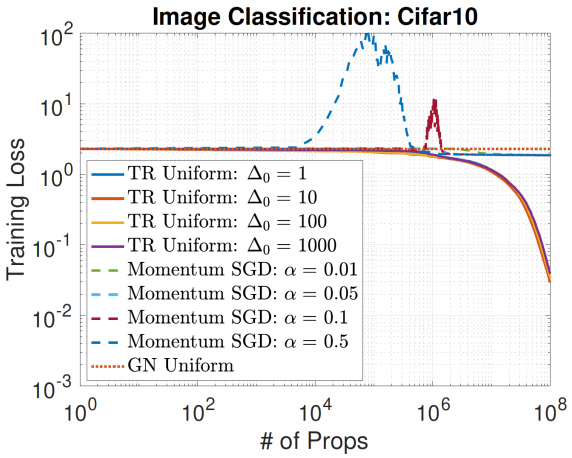

Figure 5: Original saddle point init training loss

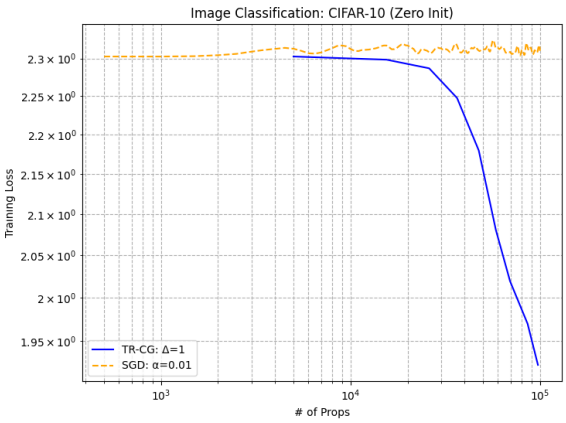

Figure 6: Reproduced saddle point init training loss

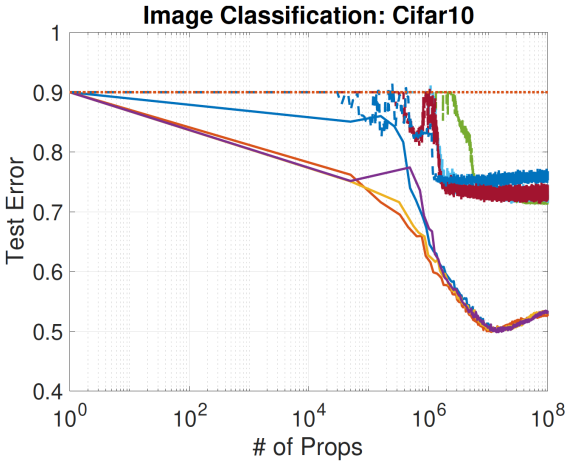

Figure 7: Original saddle point init test error

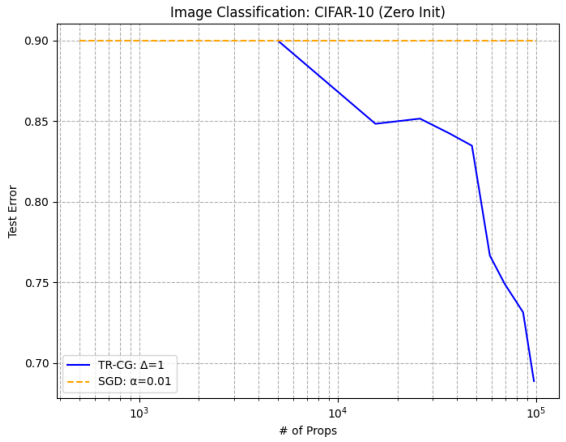

Figure 8: Reproduced saddle point init test error

2.4. Analysis and Discussion

From the experimental results, we can draw the same conclusions about the TR algorithm as in the paper.

- In terms of computational efficiency and generalization performance, the TR algorithm is very competitive with SGD. TR can be slightly slower than SGD, if SGD's learning rate is appropriately tuned. From the test error experimental results, there is no difference between TR and fine tuned SGD.
- TR is more robust to hyperparameter than SGD. From the above results, it is obvious that the computational efficiency of SGD varies greatly under different learning rates 0.05 and 0.2. In contrast, the TR algorithm has basically no difference when the trust region radius is 10 and 100.

- Regarding the saddle points escaping performance, TR has a clear advantage, and the result is the same as the result of starting from a random point. SGD has difficulty avoiding saddle points. We can conclude that the sub sampling TR method is a robust and effective method for solving non-convex machine learning problems, especially in avoiding saddle points.

3. Code Recycling and Extension (Part 2)

3.1. New Problem Description

In our final example, we address the fundamental problem challenge of training deep neural networks efficiently and reliably based on existing . While Stochastic Gradient Decent(SGD) and its variants have become the de facto standard for neural network optimization, they suffer from several limitations:

1. Sensitivity to hyperparameter, especially learning rate.
2. Susceptibility to local minima and saddle points.
3. Inconsistent convergence behavior across different architectures and datasets.

These issues become particularly pronounced when training large neural networks or working with limited computational resources. The need for extensive hyperparameters tuning creates a significant barrier for practitioners and researchers alike.

To valid the efficiency of TR algorithm, we employ an automated hyperparameter tuning tool to compare the performance of the TR algorithm against SGD.

3.2. Method Justification and Adaptation

We build upon the TR optimization approach, which naturally incorporates second-order information through the Hessian, to enhance convergence properties in training deep neural networks. This method has been integrated within the PyTorch framework, leveraging its automatic differentiation capabilities to efficiently compute gradients and Hessian-vector products.

Also, we integrate Ray Tune's hyperparameter optimization capabilities. Ray Tune is a scalable hyperparameter tuning library that supports various search algorithms and integrates seamlessly with PyTorch. It allows for efficient exploration of hyperparameter spaces, leading to improved model performance without extensive manual intervention [3].

3.3. Implementation and Results

We compare the TR algorithm and SGD algorithm on a simple neural network architecutures, specifically a feedforard network with two hidden layers. The model is trained on the CIFAR-10 dataset, a strandard benchmark for image classification tasks. The comparison is conducted under autotuning settings to evaluate the performance of both algorithms.

Model and Parameters Settings:

- **Hidden Layer 1 Size (l_1)**: Chosen from powers of two in the range {1, 2, 4, 8, 16, 32, 64, 128, 256}.
- **Hidden Layer 2 Size (l_2)**: Chosen from the same set {1, 2, 4, 8, 16, 32, 64, 128, 256}.
- **Stochastic Gradient Descent (SGD)**
  - Learning Rate (`lr`): Sampled from a log-uniform distribution between 1×10^{-3} and 1×10^{-1}.
  - Momentum: Fixed at 0.9.
  - Batch Size: Chosen from {2, 4, 8, 16}.
- **Subsampled Trust Region (TR)**
  - Learning Rate (`lr`): Sampled from a log-uniform distribution between 1×10^{-3} and 1×10^{-1}.
  - Trust Region Radius (Δ): Initialized at 0.01.
  - Batch Size: Chosen from {2, 4, 8, 16}.

We conducted 20 trials in total, with the first 10 trials using the TR optimizer and the last 10 trials using SGD. The results were evaluated based on time and validation accuracy.

Trial Name	l_1	l_2	lr	Batch Size	Iter	Time (s)	Loss	Accuracy
c02a1_00000	4	128	0.00268	4	10	2862.1	2.3091	0.0960
c02a1_00001	256	256	0.01972	4	10	3514.4	1.7389	0.3676
c02a1_00002	256	256	0.05176	2	10	6257.5	1.8955	0.3272
c02a1_00003	128	4	0.03689	16	10	1140.6	1.6011	0.4114
c02a1_00004	32	128	0.00137	2	1	555.4	2.0390	0.2277
c02a1_00005	4	16	0.00611	4	1	292.8	2.3238	0.0963
c02a1_00006	1	64	0.00101	8	1	176.4	2.0823	0.1930
c02a1_00007	256	4	0.07869	8	2	379.0	1.8259	0.3048
c02a1_00008	16	16	0.00213	8	1	176.3	1.9938	0.2382
c02a1_00009	256	128	0.01386	2	10	6044.5	1.7707	0.3675
c02a1_00010	8	32	0.00691	8	10	530.2	1.5908	0.4405
c02a1_00011	32	64	0.03329	2	1	164.0	2.3491	0.0949
c02a1_00012	8	2	0.00688	8	4	207.2	1.7917	0.3040
c02a1_00013	64	128	0.00449	8	10	549.1	1.3512	0.5591
c02a1_00014	16	2	0.06787	16	1	36.6	2.3107	0.1006
c02a1_00015	8	32	0.00135	2	10	1561.5	1.4129	0.5047
c02a1_00016	128	16	0.03465	4	1	102.6	2.3447	0.0973
c02a1_00017	2	16	0.03939	16	1	37.2	2.3074	0.1045
c02a1_00018	1	8	0.03898	8	1	55.3	2.3149	0.1050
c02a1_00019	2	32	0.01232	4	1	92.0	2.3089	0.0947

Table 1: Trial results comparing Trust Region (first 10 trials) and SGD (last 10 trials) on validation accuracy.

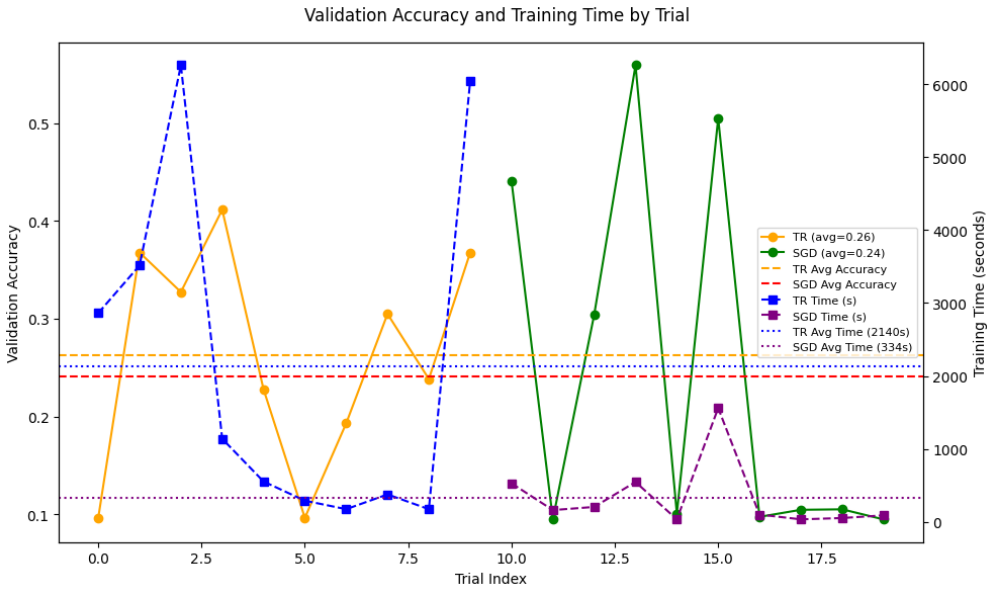

Figure 9: Validation Accuracy and Training Time Comparison between Trust Region and SGD across trials.

Based on the results summarized in Table 1 and visualized in Figure 9, several important patterns emerge. The TR optimizer achieves a slightly higher average validation accuracy (0.26) compared to SGD (0.24). TR also demonstrates more consistent performance across trials, with smaller fluctuations in accuracy. In contrast, SGD exhibits significant variance, with some trials achieving high accuracy above 0.5 while others perform poorly around 0.1.

These findings suggest that TR benefits from its ability to incorporate second-order information, enabling it to escape saddle points and better navigate non-convex optimization landscapes. However, this advantage comes at the cost of computational efficiency: the average training time per trial for TR is approximately 2140 seconds, whereas SGD requires only about 334 seconds on average. While SGD achieves faster convergence and lower computational overhead, it is more prone to instability and convergence to suboptimal solutions.

3.4. Critical Discussion

Our experiments reveal clear trade-offs between TR and SGD methods.

TR delivers consistent, reliable results by leveraging curvature information, effectively navigating saddle points in non-convex problems. However, its substantially higher computational cost limits practicality in resource-constrained environments.

SGD offers speed advantages but suffers from high variance and occasional convergence failures. These limitations typically require additional techniques like learning rate scheduling or momentum to mitigate.

Future work should explore hybrid approaches that balance TR's stability with SGD's efficiency, potentially through adaptive mechanisms that dynamically adjust second-order approximation depth based on training dynamics. Such methods could optimize the reliability-to-computation ratio for complex machine learning tasks.

4. Conclusion

In this project, we successfully reproduced the experiment results between SGD and TR in the paper, and verified the advantages of TR in non-convex optimization on the CIFAR-10 dataset. The results show that TR is better than SGD in escaping saddle points, hyperparameter robustness, and generalization performance, but its computational cost is high, especially in large-scale tasks.

In the extended experiments of automatic hyperparameter tuning, we found that TR shows higher performance stability on two hidden layer networks, while SGD had inconsistent performance due to its sensitivity to hyperparameter. TR is suitable for tasks with stability and solution quality requirements, while SGD is more suitable for training with limited computing resources. In the future, we can explore hybrid methods that combine the advantages of both methods to improve optimization efficiency.

References

[1] P. Xu, F. Roosta, and M. W. Mahoney, *Second-order Optimization for Non-convex Machine Learning: An Empirical Study*, Proceedings of the 2020 SIAM International Conference on Data Mining, pp. 199–207, 2020.
`https://epubs.siam.org/doi/10.1137/1.9781611976236.23`

[2] A. Krizhevsky and G. Hinton, *Learning Multiple Layers of Features from Tiny Images*, Technical Report, University of Toronto, 2009.
`https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`

[3] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, *Tune: A Research Platform for Distributed Model Selection and Training*, Proceedings of the 2018 Workshop on Systems for ML and Open Source Software at NeurIPS, 2018.
`https://arxiv.org/abs/1807.05118`

[4] N. Agarwal, B. Bullins, and E. Hazan, *Second-order Optimization for Non-convex Machine Learning: An Empirical Study*, Proceedings of the 34th International Conference on Machine Learning (ICML), 2017.
`https://arxiv.org/abs/1611.04970`

Appendix

Source Code Repository

The complete source code, along with scripts for reproducing the experiments and generating the figures, is available at:
`https://github.com/Linsen-gao-457/AutoTune-TR/tree/main`