Efficient Levenberg-Marquat for SLAM

Amir Belder*
Technion and Reality Labs, Meta inc.
amirbelder@campus.technion.ac.il

Refael Vivanti
Reality Labs, Meta inc.
refaelv@meta.com

Abstract

The Levenberg-Marquardt optimization algorithm is widely used in many applications and is well-known for its use in Bundle Adjustment (BA), a common method for solving localization and mapping problems. BA is an iterative process in which a system of non-linear equations is solved using two optimization methods: Gauss-Newton (GN), which requires considerable computational resources due to the calculation of the Hessian, and Gradient Descent (GD). Both methods are weighted by a damping factor, λ, which is heuristically chosen by the Levenberg-Marquardt algorithm at each iteration. Each method is better suited to different parts of the solving process. However, in the classic approach, the computationally expensive GN is calculated in every iteration, even though it may not be necessary in all cases. Therefore, we propose predicting in which iterations the GN calculation can be skipped altogether by viewing the problem holistically and formulating it as a Reinforcement Learning (RL) task, by extending a previous solution that also predicts the value of λ. We demonstrate that our method reduces the time required for BA convergence by an average of 12.5%.

1 Introduction

The *Levenberg-Marquat (LM)* optimization process (1; 2) is used in many applications, the main being *Bundle Adjustment (BA)* which is a *Simultaneous Localization And Mapping (SLAM)* solving method that is highly used in autonomous driving (3; 4; 5). The input consists of multiple 2D images of a scene captured by a single camera from various viewpoints. From these images, a set of 2D matches is derived. The objective is to determine the 3D positions of the objects and the camera's poses (both locations and orientations) during the capture, based on these 2D matches. See Fig.1 where the 3D locations appear in black and the camera's poses form the trajectory in red.

BA occupies roughly 60%–80% of the execution time needed for the mapping (6). During each iteration, the 3D locations and camera poses are initially assessed using two optimization techniques: *Gradient Descent (GD)* and *Gauss-Newton (GN)*, which are weighted according to a *damping factor*, termed λ. Then, the determined locations of the objects are projected back into 2D using the assessed poses to compute the projection error. The iterative process typically concludes (converges) when the difference between these evaluated 2D projections and the original 2D matches falls below a predefined threshold.

Two main factors influence the execution time: (1) the duration of a single iteration, which is mainly affected by the Hessian's calculation that GN entails; (2) the required number of iterations to reach convergence, caused by inefficient choosing of λ.

In the classic approach, the *Levenberg-Marquardt (LM)* algorithm (2) sets the value of λ heuristically on each iteration. λ's value may change only by one of two specific constant factors between consecutive iterations. This limits the the flexibility to effectively switch the optimization scheme between GD and GN, even when it can be beneficial.

*Corresponding author

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

Figure 1: Given a series of 2D images taken by a camera from different positions, the iterative *Bundle Adjustment (BA)* process evaluates the 3D locations of the objects in the images (in black) and the camera's poses, as seen in the red trajectory. We propose a method to accelerate the process by relinquishing calculating GN in iterations it is less required.

Some previous works focus on the first factor, and try to accelerate the Hessian's calculation (4; 7), while Belder et al. (8) address the second factor and reduce the number of required iteration by using *Reinforcement Learning (RL)* to achieve a dynamic and efficient weighting between GN and GD.

Our key idea is to address both considerations by learning a dynamic value of λ whilst deciding whether or not to use the GN on every iteration. We therefore extend Belder et al.'s (8) method to also consider if GN should be calculated on each iteration. For completeness, we include the principals of Belder et al's (8) explanations of their method. As the choice of λ's value and the use of GN on each iteration may influence the solving for *several* iterations, we propose to view the process in a holistic manner as a game similarly to (8). Thus, we employ a RL framework which determines both λ's value and whether or not to calculate the GN on every iteration.

RL problems are commonly represented by an agent and an environment, and are defined by actions, states and rewards. Each time the agent preforms an action (a), the environment responds by preforming a step according to that action and returns an observation (state s) and a reward (r). The agent chooses its actions according to a stochastic policy π, which determines the probability to choose each action in the action space. The state provides information about the environment, like the estimation error in our case, while the reward encourages the agent to reach convergence. Value functions (v) evaluate the sum of expected rewards, and are evaluated according to a specific policy, i.e. v_π. The agent aims to maximize the sum of expected rewards, which is the key to handling delayed and sparse rewards like the BA's single and delayed convergence rate.

Our environment solves the BA problem and its step performs a single BA iteration. The agent's action is twofold, it both sets λ's value like in (8) and determines if the GN should be used. The reward is positive only on the iteration convergence is achieved and if it is not achieved the reward is set as: $iteration_{time} * -1$. Therefore, in every iteration convergence is not achieved the agent gets a negative reward as a "fine". Moreover, the "fine" is higher when GN is used. As the agent aims at maximizing the sum of the expected rewards, it is encouraged to find a valid solution (reach convergence) within as little time as possible. This is the key of reaching an accurate solution faster.

We show that our method reduces the number of iterations required to achieve the BA convergence by a factor of 5 on both KITTI (9) and BAL (10) benchmarks when compared to the classic approach, thereby preserving (8)'s results. In addition, in 15% of the iterations GN was not used, leading to a average reduction of 12.5% in the solving's duration.

Hence our work makes the following contributions:

1. We propose a general and unified approach to improve LM's efficiency by both reducing the number of required iteration to reach a solution and reducing the average iteration's duration by relinquishing calculating GN whenever it is not required. Our approach is demonstrated on BA solving, but may be applied to other LM applications.

2. We propose a network that utilizes this approach using Reinforcement Learning. We show that it achieves a significant reduction in the running time. On the KITTI benchmark for

instance, a $1/5$ of the iterations were required. Morevere, in 15% of the iterations GN was not calculated, leading to an overall speedup of 3.

2 Related Work

Levenberg-Marquardt. This is a known optimization algorithm used in various application including fitting transition curves, classification error reduction, SLAM and many more (8; 11; 12; 13; 14; 15; 7; 5; 6; 4; 16; 17).

Bundle Adjustment Acceleration. In recent years several works that accelerate BA were introduced (8; 5; 6; 4; 16; 17). In (16; 17) Demmel et al. utilize fixed point approximations to accelerate the solving. Tanaka et al. (6) try to replace the BA process entirely by splitting the solving into smaller ("local") parts, and solve each local part using a NN. Other methods focus on accelerating the time of a single iteration. Huang et al. (7) use domain decomposition to split the solving into smaller clusters. Both Zhou et al. (4) and Clark et al. (18) use a NN to calculate the Jacobian matrix. Unlike these works, Belder et al. (8) focus on reducing the number of iterations of the BA solving. Our approach extends (8)'s method and decreases the number of solving iterations while also diminishing the average iteration's duration by eliminating the calculation of the Hessian when it is unnecessary.

Soft Actor Critic (SAC). This is a RL framework that seeks to enhance the conventional RL goal of maximizing rewards by incorporating a term for entropy maximization. This addition significantly boosts exploration capabilities. In their work, Haarnoja et al. (19) show that SAC achieves fast and stable convergence on various RL tasks.

3 Method

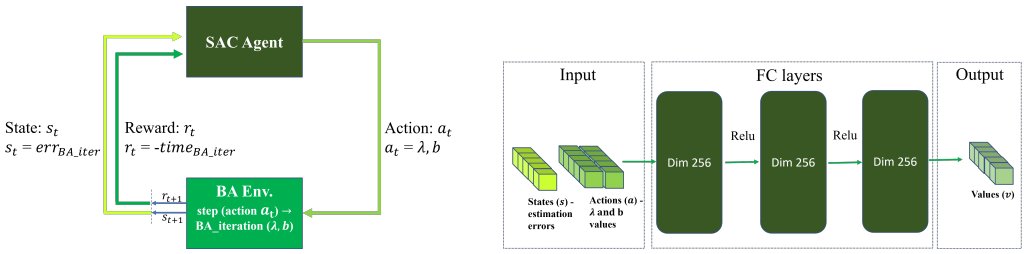
Levenberg-Marquat (LM) is a highly used optimization algorithm (1; 2) that is known for its use in *Bundle Adjustment (BA)*. Given a series of 2D images taken by a single camera, the BA iterative optimization process aims at evaluating the camera's poses and objects' 3D locations. Two optimization methods are used for the evaluation on each iteration: *Gradient Descent (GD)* and *Gauss-Newton (GN)*, that are weighted by a damping factor, λ.

In the classic approach, λ's value is set by the LM algorithm and may change by one of two constant factors between consecutive iterations. This may result in inefficient weighting of GD and GN and consequently in a large number of iterations until convergence is achieved. Moreover, the GN is computationally expensive and prolongs the time required for each iteration. As indicated in (8), efficient BA solving entails efficient weighing between GD and GN, as each is better suited for different parts of the solving. Therefore, GN might not be necessary in all iterations.

In their work Belder et al. (8) learned a *dynamic* value of λ, to dynamically weight the two optimization methods in an efficient manner. Our key idea is to extend (8)'s method and to also dynamically decide in which iterations to calculate the GN and in which not to. This is not straightforward as the BA's convergence (or failure) is achieved only once at the very end of the solving process, and since each choice of λ and GN calculation may affect the solving for several iterations. Therefore, we are required to view the solving process as a whole. Hence, we view the BA process in a holistic manner as a game as in (8).

Fortunately, *Reinforcement Learning (RL)* methods are designed to handle continuous processes, such as the BA's convergence, by viewing each process in holistic manner, that enables handling sparse and delayed rewards. Hence, RL could be harnessed to learn the optimal value of λ and whether to calculate GN on every iteration. Thus, we formulate the BA problem in RL terms by defining states, actions and rewards. As our method is not limited by two constant factors between iterations, it enables a dynamic and efficient weighting of GD and GN along the solving. This reduces the number of iterations required for convergence. Moreover, not calculating GN on every iteration reduces the average iteration's duration time.

We use the *Soft Actor Critic (SAC)* RL framework, as it is stable and adjusted to continuous state and action spaces like our problem entails (19). Our method consists of two main parts: (1) An environment which solves a single BA iteration on each step; (2) A SAC agent that predicts the value of λ and a binary value b that determines whether to use GN as its action; see Fig. 2(a).



(a) BA problem in RL terms (b) Soft-critic and Value network arch

Figure 2: **RL method.** (a) The SAC agent chooses λ's value and b (a binary value for GN's calculation) as its action (a), and then the environment preforms a single BA iteration (step), where GD and GN are weighted according to λ and b. The environment responds with: 1. a state (s) which represents the estimation error of the BA's iteration; 2. a reward (r) that represents the iteration's duration as a negative (in seconds), except for the iteration convergence is met, where r serves as a positive convergence bonus. As the agent aims at maximizing the sum of expected rewards, it is encouraged to choose λ and b in a manner that reduces the overall solving time. (b) The networks are similar in structure: 3 Fully-Connected (FC) layers with RELU as the activation function, following (20)'s implementation. The value network receives the state vector (dim $5X1$) as input, while both soft-critic networks networks receive the state and the action vectors (dim $5X3$) as input.

Environment. Following (8), the environment solves a BA problem. On each step (BA iteration), the environment receives λ and a binary value b that indicates whether GN should be calculated as an action. It weighs the GD and GN according to them, estimates the 3D locations of the objects and the camera poses, and then projects these locations into 2D according to the estimated poses. The stopping criterion is met when the estimation error is smaller than a certain threshold.

The environment provides a reward (r) and an observation (state s) on each step (iteration). Let z_{ij} be the ground truth pixel (match) in which key-point j appeared in image i. We model it as a noised projection of 3D-point q_j on camera c_i with a w Gaussian projection noise, i.e $z_{ij} = Projection(c_i, q_j) + w$. Let $\hat{c_i}, \hat{q_j}$ be the current iteration's estimated poses of the camera i and location of 3D-point j accordingly, and let $\hat{z_{ij}}$ be the respective projection i.e $\hat{z_{ij}} = Proj(\hat{c_i}, \hat{q_j})$. Let Δz_{ij} be the difference between the ground truth projection and the estimated projection, i.e. $\Delta z_{ij} = z_{ij} - \hat{z_{ij}}$. Let C, Q be all the estimated camera poses and all the estimated 3D locations respectively. The estimation error is set as the sum of Δz_{ij}, as follows:

$$\begin{aligned} &\text{Estimation error} = \Sigma_{ci}^{C}\Sigma_{qj}^{Q}||\not{\Lambda}^{-1/2}\Delta z_{ij}||^2 \\ &BA_{objective} = argmin_{CQ}[\text{Estimation error}], \end{aligned} \tag{1}$$

where $\not{\Lambda}$ is the covariance matrix, and the state (s) is set as a vector of the 5 last consecutive errors, in order to enable the agent to learn the influence of the choice of λ over a few iterations. This forms the connection between the BA solving and the estimation error.

The reward (r) is set as the negative of the duration of each iteration (in seconds), apart from the convergence iteration (terminal state) where the reward is set as positive. In standard RL problems the agent is encouraged to maximize the sum of expected rewards:

$$E_\pi = \sum_{t=0}^{\infty} r_t, r_t = -time_{BAiter_t}[seconds], \tag{2}$$

where r_t is the reward at time step (iteration) t received according to policy π. In our case, the agent is encouraged to minimize the overall processing time by reaching convergence, which indirectly minimizes the number of iterations and the use of GN.

Soft Actor Critic (SAC). Following (8), our SAC framework consists of five networks that are updated according to the known actor-critic iterative optimization scheme: (1) an actor policy network that learns the actions; (2) two identical on-policy soft-critic networks, which evaluate the value function and differ in a time-delay; (3) an off-policy value network that evaluates the value; (4)

Table 1: **Average efficiency improvement.** Our approach accelerated the solving process by reducing number of iterations, similarly to (8), by a factor of 5. Moreover, it reduced the overall solving's duration in all cases by $12\% - 13\%$ more than (8). For fair comparison, all times are reported on our hardware using the original implementation of each method.

Method	dataset	#Iterations	Duration [sec]
Classic	KITTI	75	340.0
(8) + classic	KITTI	14	100.2
Ours + classic	KITTI	15	**88.1**
Classic	BAL	20	110.0
(8) + classic	BAL	4	62.04
Ours + classic	BAL	4	**55.11**

a target network that converges the values predicted by the on-policy and off-policy networks into a single target value required for the actor-critic optimization.

As the action represents the values of λ and b, it influences the optimization process directly. Let x be all the estimated camera's poses and 3D locations in the BA problem, J be the Jacobin, H be the Hessian, Δz be a vector whose entries are Δz_{ij} defined before, $\not\prec$ be the covariance matrix, b be the binary value that determines whether GN is calculated and λ be the damping factor. The optimization step taken on each iteration to update x is defined as:

$$\begin{aligned} \Delta x &= -\frac{1}{\lambda} J(x)^T \not\prec^{-1} \Delta z \\ \text{O}ptimization_{gradient} &= -(b * H + \lambda I)\Delta x. \end{aligned} \tag{3}$$

Following (20)'s implementation, both soft-critic, target and value networks have similar architecture, as seen in Fig. 2(b). The value network gets only the state as input, while the two soft-critic networks get both the state and the action. The target network gets the values (predictions) of the value network and the chosen critic network and predicts a target (value) according to both on each iteration. The policy network consists of four FC layers (dim 256), gets the state as input and predicts the next action. The loss functions of all networks are set to MSE.

4 Experiments

Datasets. We utilize two common real-life large datasets for our experiments: KITTI (9) and BAL (10), in which each scene may include tens of thousands of points.

Results. We compare our results to those of the classic BA approach with LM python implementation (2), and between Belder et al.'s (8) acceleration of the classic approach to our acceleration. The stopping condition threshold was set to 10^{-6} for both datasets. All time measurements are reported in seconds and all methods ran on the same hardware. Each of the reported times includes both the approach's set-up time and its BA solving time.

Table 1 compares our results on both KITTI and BAL datasets. In addition to reducing the number of iterations required to reach convergence by factor of 5, similarly to (8)'s acceleration, our method also reduces (8)'s solving duration by $12\% - 13\%$. This is achieved by not calculating the Hessian in 15% of the iterations. We note that all method had the same (100%) success rate. In terms of accuracy, we compared the MSE between the final estimations coordinated and the ground truth coordinates on the BAL (10) dataset, and got similar results (difference < 0.002) for all methods.

Limitations. When small BA problems which are commonly solved in a few iterations by the classic approach are considered, we cannot improve them to the same extent as bigger BA problems. For example, when 5 points and 5 camera poses are used, the classic approach reaches convergence within 5 iterations on average, while our method and (8) required 4 iterations on average. Moreover, GN was calculated in all iterations.

References

[1] F. D. Foresee and M. T. Hagan, "Gauss-newton approximation to bayesian learning," in *Proceedings of international conference on neural networks (ICNN'97)*, vol. 3. IEEE, 1997, pp. 1930–1935.

[2] J. J. Moré, "The levenberg-marquardt algorithm: implementation and theory," in *Numerical analysis*. Springer, 1978, pp. 105–116.

[3] K. Ni, D. Steedly, and F. Dellaert, "Out-of-core bundle adjustment for large-scale 3d reconstruction," in *2007 IEEE 11th International Conference on Computer Vision*. IEEE, 2007, pp. 1–8.

[4] L. Zhou, Z. Luo, M. Zhen, T. Shen, S. Li, Z. Huang, T. Fang, and L. Quan, "Stochastic bundle adjustment for efficient and scalable 3d reconstruction," in *European Conference on Computer Vision*, 2020, pp. 364–379.

[5] J. Ortiz, M. Pupilli, S. Leutenegger, and A. J. Davison, "Bundle adjustment on a graph processor," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2416–2425.

[6] T. Tanaka, Y. Sasagawa, and T. Okatani, "Learning to bundle-adjust: A graph network approach to faster optimization of bundle adjustment for vehicular slam," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 6250–6259.

[7] J. Huang, S. Huang, and M. Sun, "Deeplm: Large-scale nonlinear least squares on deep learning frameworks using stochastic domain decomposition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 10 308–10 317.

[8] A. Belder, R. Vivanti, and A. Tal, "A game of bundle adjustment - learning efficient convergence," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2023, pp. 8428–8437.

[9] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[10] S. Agarwal, N. Snavely, S. M. Seitz, and R. Szeliski, "Bundle adjustment in the large," in *European conference on computer vision*. Springer, 2010, pp. 29–42.

[11] I. Khan, M. A. Z. Raja, M. Shoaib, P. Kumam, H. Alrabaiah, Z. Shah, and S. Islam, "Design of neural network with levenberg-marquardt and bayesian regularization backpropagation for solving pantograph delay differential equations," *IEEE Access*, vol. 8, pp. 137 918–137 933, 2020.

[12] A. P. Piotrowski and J. J. Napiorkowski, "Optimizing neural networks for river flow forecasting–evolutionary computation methods versus the levenberg–marquardt approach," *Journal of hydrology*, vol. 407, no. 1-4, pp. 12–27, 2011.

[13] R. Zhou, D. Wu, L. Fang, A. Xu, and X. Lou, "A levenberg–marquardt backpropagation neural network for predicting forest growing stock based on the least-squares equation fitting parameters," *Forests*, vol. 9, no. 12, p. 757, 2018.

[14] J. Shawash and D. R. Selviah, "Real-time nonlinear parameter estimation using the levenberg–marquardt algorithm on field programmable gate arrays," *IEEE Transactions on industrial electronics*, vol. 60, no. 1, pp. 170–176, 2012.

[15] Z. Song, F. Yang, P. Schonfeld, J. Li, and H. Pu, "Heuristic strategies of modified levenberg–marquardt algorithm for fitting transition curves," *Journal of Surveying Engineering*, vol. 146, no. 2, p. 04020001, 2020.

[16] N. Demmel, C. Sommer, D. Cremers, and V. Usenko, "Square root bundle adjustment for large-scale reconstruction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 11 723–11 732.

[17] N. Demmel, D. Schubert, C. Sommer, D. Cremers, and V. Usenko, "Square root marginalization for sliding-window bundle adjustment," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2021, pp. 13 260–13 268.

[18] R. Clark, M. Bloesch, J. Czarnowski, S. Leutenegger, and A. J. Davison, "Ls-net: Learning to solve nonlinear least squares for monocular stereo," *CoRR*, vol. abs/1809.02966, 2018. [Online]. Available: http://arxiv.org/abs/1809.02966

[19] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[20] Z. Ding, "Popular-rl-algorithms," https://github.com/quantumiracle/Popular-RL-Algorithms, 2019.