World Academy of Science, Engineering and Technology
International Journal of Mechanical and Mechatronics Engineering
Vol:17, No:10, 2023

Robot Movement Using the Trust Region Policy Optimization

Romisaa Ali

***Abstract*—**The Policy Gradient approach is a subset of the Deep Reinforcement Learning (DRL) combines Deep Neural Networks (DNN) with Reinforcement Learning (RL). This approach finds the optimal policy of robot movement, based on the experience it gains from interaction with its environment. Unlike previous policy gradient algorithms, which were unable to handle the two types of error variance and bias introduced by the DNN model due to over- or underestimation, this algorithm is capable of handling both types of error variance and bias. This article will discuss the state-of-the-art SOTA policy gradient technique, trust region policy optimization (TRPO), by applying this method in various environments compared to another policy gradient method, the Proximal Policy Optimization (PPO), to explain their robust optimization, using this SOTA to gather experience data during various training phases after observing the impact of hyper-parameters on neural network performance.

***Keywords*—**Deep neural networks, deep reinforcement learning, Proximal Policy Optimization, state-of-the-art, trust region policy optimization.

I. Introduction

MACHINE learning is a field of study that involves teaching computers to learn from data. There are three main sub-fields within machine learning: supervised learning, unsupervised learning, and RL. The primary difference between RL and the other two sub-fields is that RL relies on an agent learning through interacting with its environment, rather than a training set of labeled examples provided by an external supervisor. In supervised learning, each example includes a description of a circumstance and a label, which defines the correct action the system should take in response to that condition. In unsupervised learning, the focus is on discovering hidden structures within groups of unlabeled data. RL is often mistaken for unsupervised learning because it does not rely on examples of appropriate conduct. However, the goal of RL is to maximize a reward signal, rather than seeking out hidden structures. While finding structure in an agent's experience can be helpful for RL, it does not solve the problem of how to maximize a reward signal for the agent. Therefore, RL should be considered a separate machine learning method alongside supervised and unsupervised learning [1]. Recent research has focused on integrating deep learning with RL, resulting in the development of deep RL systems, algorithms, and agents that have achieved impressive outcomes. These systems are capable of outperforming human intelligence at tasks that were previously thought to require high levels of human intelligence, creativity, and planning abilities [2].

Romisaa Ali is with Dept. Computer and Control Engineering (DAUIN), Politecnico di Torino University, Turin, Italy (e-mail: romisaa.ali@polito.it).

II. Policy Gradient

In this article, we will focus on a specific subset of RL methods called policy gradient (PG) approaches. These methods aim to optimize parameterized policies in relation to the expected return, or the long-term cumulative reward. The objective of the PG method is to determine the optimal action to take in order to maximize the cumulative rewards. Unlike other RL methods that use a value table to associate each action with its state, the PG method can easily handle large state spaces. The policy, which is a DNN, decides which action to take based on the network parameters. The PG method has two optimization methods: stochastic gradient ascent (SGA) and stochastic gradient descent (SGD). SGA moves in the direction of optimization, while SGD aims to minimize the cost function or climate by moving towards the minimum point of the loss function [3]. After an action is selected, the PG algorithms evaluate the last policy and determine the best one. The improvement step for the policy is done by updating the neural network parameters. Each PG algorithm has a different way of evaluating and improving the policy. The Reinforce algorithm's performance is based on the average cumulative return G (Θ), see (1) which can have high variance due to rewards being random variables. On the other hand, the Advantage Actor-Critic (A2C) algorithm's performance is based on the advantage function AΘ(s, a), see (2) which takes into account the difference between the next state and the immediate state, resulting in lower variance but with a potential for bias. In this article, we will focus on PG methods that have the ability to balance the trade-off between variance and bias to introduce powerful algorithms. Specifically, we will focus on methods that utilize trust regions to find optimization with low variance and low bias; in particular, we shall examine the following algorithms: TRPO and PPO. In the following section, we will examine in depth the concept of trust region and provide a thorough analysis of the TRPO algorithm. We will explain how TRPO finds the optimization, and in the subsequent section, we will compare the implementation of TRPO to the performance of PPO algorithm.

$$G_t(\theta) = \sum_{k=t+1}^{T} \gamma^{k-t-t} R_k \quad (1)$$

$$A_\theta(s, a) = Q(s, a) - V(s) \quad (2)$$

Q (s, a) is the q value of the action in the immediate state; V(s) is the value of the immediate state.

A. Trust Region Method (TRM)

Line searching and trust region searching are two types of searches used in the PG method for optimization. While the PPO and TRPO rely on the trust region search, the algorithms A2C, reinforce, DDPG, etc., rely on the line search. In this paper, we propose a method for optimizing the performance of a neural network using a line search algorithm. The method involves tuning two parameters, w1 and w2, with the goal of maximizing policy performance. Our approach involves identifying the direction of the maximum accent and then selecting an appropriate step size. As shown in Fig. 1, this approach allows for efficient and precise optimization of the neural network's performance. Additionally, we incorporate a trust region approach to further enhance the optimization process. For each new update and policy point, we identify the trust region border inside the point, as shown in Fig. 2. We then search inside this region for a different policy with higher performance until we achieve optimization. This approach allows for the identification of a more optimal policy, and thus, an improvement in the overall performance of the neural network.

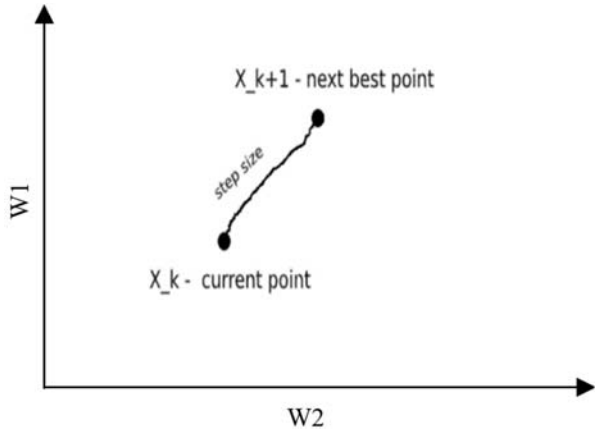

Fig. 1 Policy performance of line search method with the two neural network parameters w1 and w2

The Trust Region Method (TRM) [12] initially identifies a region that includes the current best solution and can approximate the basic goal function. The TRM then moves forward in keeping with what the region's model predicts. TRM often decides the step size before the improving direction, in contrast to line search approaches in the previous PG algorithms A2C, reinforce, etc. [4]. TRM helps us to make the learning process more efficient and reliable. This method addresses two things: 1) sample efficiency, the ability of an RL algorithm to learn from a small number of interactions with the environment, as opposed to a large number of interactions. 2) The reliability of the steps taken to find that optimal policy while avoiding the steps that decrease the performance. In this method, first, we must determine the region of trust by creating a bound around the last policy point. Then inside this region, we must find the other parameters that can give us better policy and the new policy point, and so on until obtaining the best policy, with reliable improvement for the new policy. TRM are based on using the Kullback-Leibler divergence (KL divergence) as a tool to measure the difference between two policies. The better policy in the trust region bound must be equal to or less than the delta δ. Inside the region, all available policies have a divergence less than or equal to the delta. When we have continuous action, the DKL will be the integration of the equation (see (4)), and the KLB equation (3) applies for the discrete action space.

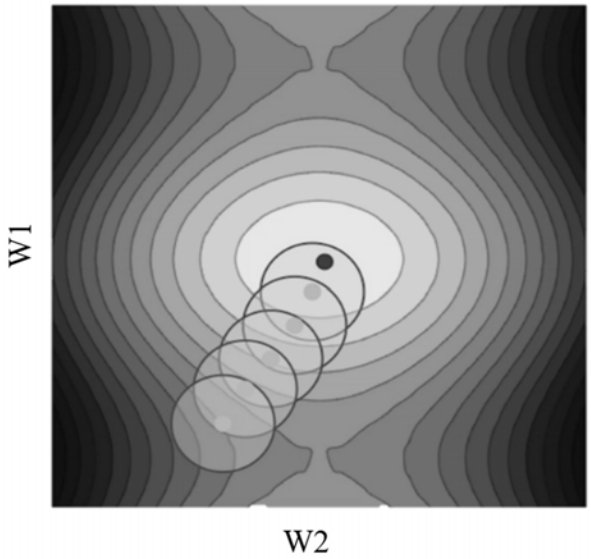

Fig. 2 Policy performance of trust region method with the two neural network parameters w1 and w2

$$D_{kl}(\pi_1//\pi_2) = \sum_{a\in A} \pi_1(a \setminus s) \quad \ln\left(\frac{\pi_1(a\setminus s)}{\pi_2(a\setminus s)}\right) \leq \delta \qquad (3)$$

$$D_{kl}(P//Q) = \sum_{x\in X} P(x) \ln\left(\frac{P(X)}{Q(X)}\right) \leq \delta \qquad (4)$$

We present (5) that allows us to compare the performance of two policies: $J(\pi)$ the last policy and $J(\tilde{\pi})$ the new policy. The equation expresses the performance of the new policy as the performance of the last policy, plus the expectation $\mathbb{E}$ of the simulation of selection for each action in each state for the expected discounted advantage $A_\pi(s_t, a_t)$. The expectation of the advantage of the new policy is compared with the previous one. If the expectation of the advantage of the new policy is higher, then the performance of the new policy is better than the previous one. This means that the new policy will expect to select the best actions more often than the last one. In other words, the new policy will select the actions with more positive advantages when it visits the same states.

$$J(\tilde{\pi}) = J(\pi) + \mathbb{E}_{s_0 a_0 \ldots \sim \tilde{\pi}}[\sum_{t=0}^{\infty} \gamma A_\pi(s_t, a_t)] \qquad (5)$$

The $L(\pi)$ value is the surrogate function [5], which represents the frequency sampled for the new policy to easily approximate the performance of the new policy by using the number of the visitation frequency of each state of the last policy $\rho_\pi(s)$ instead of the new one $\rho\tilde{\pi}(s)$, which is the hardest to estimate. Through this function, we can simplify the search for a better policy by

making the number of visitations states be fixed during the research process, see (6):

$$J(\tilde{\pi}) = J(\pi) + \sum s \rho\pi(s) \sum a \tilde{\pi}(a \setminus s) A\pi(s,a) \quad (6)$$

The function $L(\pi_\Theta)$ must take the value of the $J(\pi_\Theta)$ which is the derivative of the first order to calculate the direction of the gradient descent. It must be the same, see (7) and (8); the two values must be equal for the first derivative.

$$L\pi_{\theta_0}(\pi_{\theta_0}) = J(\pi_{\theta_0}) \quad (7)$$

$$\nabla_\theta L\pi_{\theta_0}(\pi_{\theta_0})\|_{\theta=\theta 0} = \nabla_\theta J(\pi_{\theta_0}) \|_{\theta=\theta 0} \quad (8)$$

After calculating the derivative using $L(\pi_\Theta)$ instead of $J(\pi_\Theta)$, the purpose of (8) is to determine the step size, denoted by α, the step size α is a hyperparameter that determines the magnitude of the update to the parameters θ at each iteration of the PG optimization algorithm that must be taken to guarantee improvements in the policy (π). The left side of the equation represents a constant (C) multiplied by the maximum Kullback-Leibler divergence (Dkl) between the current and new policy. By applying this formula, we can find a policy with better performance using the minorize-maximization algorithm. The minorize-maximization algorithm is a technique used to optimize a function by iteratively finding a lower bound of the function, called the minorizer, and then maximizing the minorizer. By applying this equation within the minorize-maximization algorithm [13], we can iteratively improve the policy and find the policy that yields the best performance, see Fig. 3 [6] To find the optimal policy, we must try all policies within the region and apply (9) to each one of them. In this way, the optimal policy is the one that minimizes the left side of the equation.

$$J(\tilde{\pi}) \geq L_\pi(\tilde{\pi}) - CD_{kL}^{\max}(\pi, \tilde{\pi}) \quad (9)$$

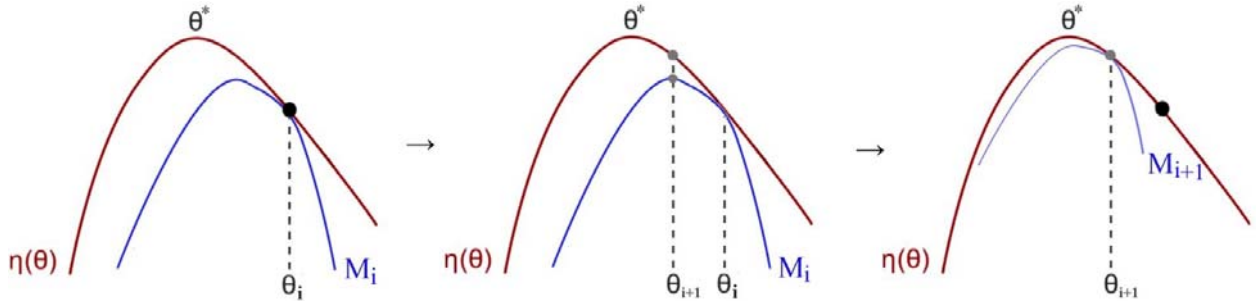

Fig. 3 Minorize Maximization Algorithm of policy performance

The surrogate function represented by L minus the term $CD_{kL}^{\max}$ ensures that the new policy is close enough to the previous policy. By applying this local function to each new policy, we find the policy that results in the local maximum, as shown in (10). This process is repeated until the optimal policy or a policy that is sufficiently close to the previous policy is found. Equation (10) describes the updates of the new neural networks parameters made to the new policy during this process.

$$\theta_{t+1} = \arg^{\max_\theta} L(\theta|\theta_t) - CD_{kl}^{max}(\theta_t, \theta) \quad (10)$$

Equation (10) describes a method for updating the policy in which the value of C is very small in comparison to CD_{kl}^{max}. This means that many steps must be taken for each local surrogate value to reach the optimal policy. To update the local surrogate function and take larger steps without losing the guarantee, one approach is to remove the value of C from the equation. This can be achieved by maximizing the value of L while ensuring that the constraints $CD_{kl}^{max} \leq \delta$ are met. By removing the value of C, the optimization process can be accelerated and the optimal policy can be reached faster. This allows for a more efficient search for the optimal policy while still maintaining the guarantee of similarity between the current and new policies.

$$\mathbf{max}_{\boldsymbol{\theta}}(\theta) \ \textit{subject to the value of average} \quad (11)$$

In (11), we must compare the average of all samples in the trust region, and if this average is smaller than or equal to the delta value, then we can take values of any samples in the region even if the sample is bigger than delta. In the next paragraph, we will discuss how the TRPO method utilizes the truth region for optimization.

B. Trust Region Policy Optimization

Our goal is to maximize the surrogate function, which is a measure of how well our policy is performing, by making the Dkl value, a measure of the difference between two probability distributions, smaller than or equal to delta as stated in (11). To achieve this, we use the TRPO method, which is a part of the TRM family. The TRPO method involves expanding the surrogate function to maximize the expression in (12). This is done by removing the discounting factor (gamma) from the term that represents the sum of the discounted number of visitations of states under the old policy $\rho_{\theta_{old}}(s)$. This simplifies the equation and makes it easier to understand. Additionally, we also remove the part of the equation that represents the sum of the new expected policy times the expected advantage of the old policy $\pi_\theta(a \setminus s) A_{\theta_{old}}(s,a)$. It is important to note that these simplifications do not change the overall value of the surrogate function, they only make the equation clearer.

$$\mathrm{Max}_{\Theta} \sum_{s} \rho_{\theta_{old}}(s) \sum_{a} \pi_{\theta}(a \setminus s)\, A_{\theta_{old}}(s,a) \qquad (12)$$

$\boldsymbol{\rho_{\theta_{old}}}$**:** The number of visitation frequency of each state for the old policy; $\boldsymbol{\pi_{\theta}}$ **(**$\boldsymbol{a \setminus s}$**):** The expected new policy; $\boldsymbol{A_{\theta_{old}}(s,a)}$**:** The advantage function of the old policy.

$$\textbf{replace} \sum_{s} \rho_{\pi_{old}}(s) \qquad \textbf{with} \qquad \frac{1}{1-\gamma}\mathbb{E}_{s\sim\rho\theta_{old}}$$

$$\textbf{replace} \quad \sum_{a} \pi_{\theta}(as_n)\, A_{\theta_{old}}(s_n,a)$$

$$\textbf{with} \qquad \mathbb{E}_{a\sim\pi\theta_{old}}\left[\frac{\pi_{\theta}(a\setminus s_n)}{\pi_{\theta_{old}}(a\setminus s_n)} A_{\theta_{old}}(s_n,a)\right]$$

After making the changes to the equation as previously explained, the surrogate function can be calculated more easily. The expression for this calculation can be found in (13). It is the maximum expectation of state visits under the previous policy, as well as the expectation of actions taken under the previous policy.

$$\max_{\theta} \mathbb{E}_{s\sim\rho\theta_{old},\ a\sim\pi_{\theta_{old}}}\left[\frac{\pi_{\theta}(a\setminus s_n)}{\pi_{\theta_{old}}(a\setminus s_n)} A_{\theta_{old}}(a \setminus s_n)\right] \qquad (13)$$

Because the probability of choosing the action in a given state under the new policy is not yet accessible, (13) is still difficult to use. The Taylor Series Technique [7] is used by the TRPO algorithm to resolve this issue. The surrogate function $L(\theta)$, represented by (14), helps to evaluate the performance of a set of policy parameters (shown as θ). The change in this performance is measured by comparing it to the previous set of policy parameters θ_{old} using the first derivative of the surrogate function g^T. The TRPO algorithm uses a Kullback-Leibler divergence D_{KL} to measure this change as shown in (15). In this equation, H represents the second derivative of D_{KL} with respect to the policy parameters θ and θ_{old}, and the variable δ represents the maximum allowed change in the policy.

$$L(\theta) \approx g^T(\theta - \theta_{old}) \qquad (14)$$

$$D_{KL}(\theta) \approx \tfrac{1}{2}(\theta - \theta_{old})^T H(\theta - \theta_{old}) \leq \delta \qquad (15)$$

The surrogate function, represented by (14), is a mathematical formula that helps to evaluate the performance of a certain set of policy parameters θ. These parameters are chosen by finding the highest Taylor gradient value during a search on the trust area. These selected parameters are then used to update the neural networks as shown in (16). However, this update is subject to a constraint, the Kullback-Leibler divergence D_{KL} should not exceed a certain value called delta.

$$\theta_{k+1} = \arg\max_{\theta} \approx g^T(\theta - \theta_k) \qquad (16)$$

Subject to

$$\tfrac{1}{2}(\theta - \theta_{old})^T H(\theta - \theta_{old}) \leq \delta \qquad (17)$$

Equation (18) represents the calculation of new policy parameters based on the maximum value of the surrogate function. As you can see, the formula is like the one used to update parameters in the SGD method.

$$\theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g \qquad (18)$$

In (18), the update of the new parameters is based on the maximum value of the surrogate function, using a step size represented by the second part of the equation, on the right side of the equation. The direction of the step size is represented by $H^{-1}g$, which is the direction towards optimization and the growth of the maximum policy. However, there is a problem of maintaining the guarantee of improved policy while staying within the Kullback-Leibler divergence D_{KL} bounds by using this formula, (19):

$$\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g \qquad (19)$$

It includes a constant value, α, which is between 0 and 1, and it is raised to the power of j. The value of j can be a constant (1, 2, 3, 4...) and it changes until finding the value of the most recent update inside the trust region boundary. This means that the value of j is adjusted until the step size is found that does not cause the boundary to be exceeded. To optimize the strategy, this algorithm is applied in the next part.

III. Experience

This experience uses Google Colab [9] as the platform to perform the experiments and run the TRPO algorithm on the Ant and Humanoid robot environments. Google Colab is a platform that allows the execution of Python code on a Linux operating system, which is equipped with a GPU processor and 27.3 gigabytes of RAM. This platform is chosen for its ability to handle large amounts of data and its ease of use. The PyTorch toolkits are employed to implement the TRPO algorithm on the Ant and Humanoid environments. PyTorch [11], [12] is a popular deep learning framework that is widely used for machine learning and deep learning applications. The PyTorch toolkits [8], provide a convenient way to run the TRPO algorithm on the robot environments. The study compares the average return of the Ant and Humanoid environments using TRPO with the average return using the PPO method. The PPO method is a popular algorithm that is widely used in RL. The study aims to see if the TRPO algorithm can outperform the PPO method in these robot environments. The results are analyzed and visualized using TensorBoard, a tool that allows the user to easily view and analyze the performance of the algorithm and see how the algorithm is performing over time and identify any patterns or trends that may indicate areas for improvement. Lastly, the study also examines the impact of hyper-parameters. The study aims to see how the choice of hyper-parameters affects the optimization process and how it may impact the overall performance of the algorithm.

A. Exploring the Impact of Batch Size on the Efficiency of TRPO for Robot Movement

To achieve the best results, we aimed to find the optimal hyper-parameters for the TRPO method. These hyper-parameters include the training rate, epsilon greedy (the probability of taking a random action), batch size, discount factor (used to determine the importance of future rewards), and number of epochs (the number of times the model is trained on the entire dataset). In this study, we focused specifically on the batch size and its effect on the efficiency of the TRPO method. We found that smaller batch sizes [10] resulted in more steps per episode, as a smaller batch size requires more iterations to visit all the data in the dataset. However, it was also observed that larger batch sizes (2048 and 1024) resulted in faster learning but later convergence and stability. Additionally, we found that the processor and RAM used in this experience had a significant impact on the batch size efficiency, the data shown in Figs. 4 and 5 illustrate our findings and the relationship between batch size and steps per episode. The figures demonstrate that as the batch size increases, the number of steps per episode decreases. Overall, our research suggests that the batch size is an important hyper-parameter to consider when using the TRPO method and that the optimal batch size may vary depending on the specific environment and hardware being used, see Table I.

B. The Functionality of TRPO and PPO Algorithms in the Actor-Critic Network: A Comparison of Parameters

TRPO and PPO algorithms are based on two neural networks: an actor-network and a critic network. The actor chooses which action to take, and the critic estimates the action's value as selected by the actor. The critic calculates the TD error to improve the network parameters and then sends this error to the actor to improve its parameters for a more accurate prediction of the action's next selection The parameters of the source codes of TRPO actor learning rate = 0.001, critic learning rate = 0.0001, $\gamma = 0.99$, $\lambda = 0.95$, $D_{kl} = 0.25$, and the PPO source code have the same parameters except $\varepsilon = 0.3$ and the entropy coefficient = 0.1, the number of episodes was 1000.

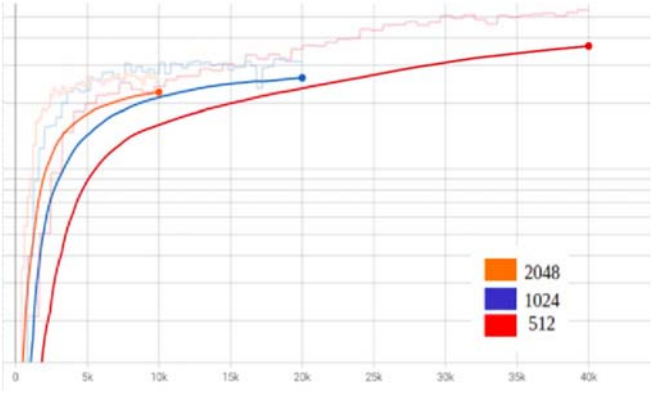

Fig. 4 Results of the average return that the neural network model produced for the various batch sizes, TRPO method

C. Comparison of TRPO and PPO Methods for Training a Robot: Results and Analysis

The TRPO and PPO methods were both used to train a robot. The results of this training are presented in Figs. 6 and 7. The TRPO method was found to have better average return performance when applied to the ant and humanoid robot environments, but required more time for training compared to PPO. However, after sufficient training, TRPO still performed better than PPO when the robot was required to move forward while also moving its arms. PPO, on the other hand, required less time for training and produced better returns initially. Additionally, using a batch size of 1024 resulted in better performance for the humanoid robot, which has a larger dataset than the ant robot, see Table II. The results indicate that PPO had more stability during training in both the ant and humanoid robots. The humanoid robot training in TRPO in the iteration 40k was raised to the highest value recorded 605, but with training, it started to decline until reaching 508. This suggests that the ant robot can learn easily and move forward and fast, but the humanoid robot may require more training or has not yet learned enough to move forward and fast after being trained with both algorithms.

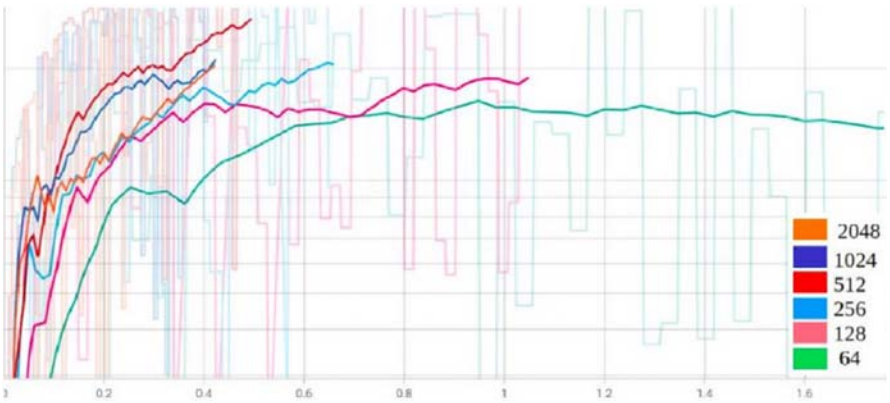

Fig. 5 Results of the average return that the neural network model produced for the various batch sizes, PPO method

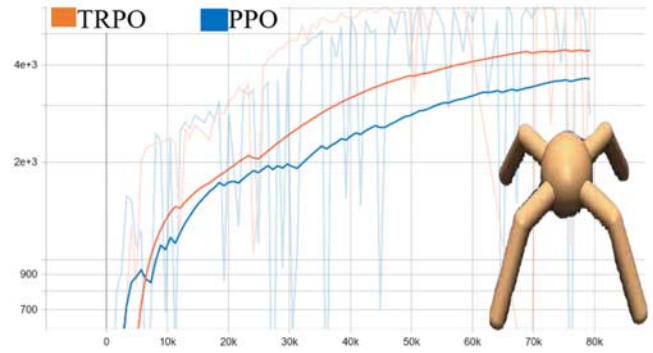

Fig. 6 Ant environment result

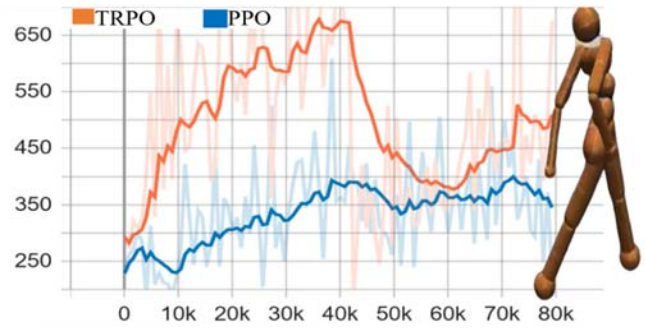

Fig. 7 Humanoid environment result

TABLE I
BATCH SIZE

Symbol	Number of samples in one batch TRPO	Average return	Number of samples in one batch PPO	Average return
1	**64**	Value error	**64**	1384
2	128	Value error	128	1889
3	256	Value error	256	2045
4	512	4442	512	3630
5	1024	2622	1024	2110
6	2048	2251	2048	2047

TABLE II
EXPERIENCED AVERAGE RETURN

Environment	TRPO Average Return	PPO Average Return	TRPO Training time	PPO Training time
Ant	4442	3630	1 hour, 52 minutes	58 minutes
Humanoid	508	345	1 hour, 50 minutes	53 minutes

IV. CONCLUSION AND RECOMMENDATIONS

To improve the effectiveness of the TRPO and PPO methods for robot movement, it is important for future research to address the trade-off between training time and performance. Specifically, efforts should be made to reduce the training time for TRPO while maintaining its superior performance in tasks such as moving forward while also moving the robot's arms. Additionally, it would be valuable to investigate the adaptability and performance of the two methods in environments with a high number of obstacles, as this could have a significant impact on the robot's ability to navigate. Furthermore, it is important to evaluate the performance of both methods in different scenarios, such as environments with and without obstacles, and to study the effect of obstacle density and types on the performance. Additionally, it is crucial to monitor the performance of the robot in real-world environments, as simulation results may not always accurately reflect real-world performance. Finally, it would be beneficial to investigate the scalability and robustness of the TRPO and PPO methods when dealing with high numbers of obstacles in the environment. To make the TRPO and PPO methods work better for robot movement, it is important to experiment with different hyperparameters and not just focus on the batch size, as this could help improve the performance and effectiveness of the methods.

ACKNOWLEDGMENT

Invaluable assistance from Prof. Marcello Chiaberge of Politecnico di Torino University and Maurizio Griva from REPLY Company was essential in the completion of this work and the supporting research. Deep gratitude goes to colleagues at the PIC4SeR center and for the support from REPLY Company. Special appreciation is directed to the book "Reinforcement Learning: An Introduction" by Richard S. Sutton and Andrew G. Barto, as well as the Reinforcement Learning course series on Udemy.com. Both have been constant sources of inspiration and knowledge throughout the duration of this work.

REFERENCES

[1] R. S. Sutton and A. G. Barto, A Reinforcement Learning: Introduction. Mit Press, 2012.
[2] M. Sewak, Deep reinforcement learning: Frontiers of artificial intelligence, 1st ed. Singapore, Singapore: Springer, 2020.
[3] Ott Toomet, "Stochastic Gradient Ascent in maxLik," 2020.
[4] Jorge Nocedal, Stephen J. Wright, Sequential Quadratic Programming, Springer, New York, NY, 1999.
[5] Iris Smit, Reinforcement Learning, and surrogate reward functions based on graph Laplacians, Utrecht University, 2022
[6] J. Hui, "RL — The Math behind TRPO & PPO," Medium, Sep. 14, 2018. https://jonathan-hui.medium.com/rl-the-math-behind-trpo-ppo-d12f6c745f33
[7] C. Canuto and A. Tabacco, Mathematical analysis II, 2nd ed. Basel, Switzerland: Springer International Publishing, 2015
[8] "PyTorch Lightning," Pytorchlightning.ai. (Online). Available: https://www.pytorchlightning.ai/. (Accessed: 03-Oct-2022).
[9] "Google colab," Google.com. (Online). Available: https://research.google.com/colaboratory/faq.html. (Accessed: 03-Oct2022).
[10] Machinelearningmastery.com. (Online). Available: https://machinelearningmastery.com/difference-between-a-batch-and-anepoch/. (Accessed: 03-Oct-2022).
[11] Sehgal A, La H, Louis S, Nguyen H, editors. Deep reinforcement learning using genetic algorithm for parameter optimization. 2019 Third IEEE International Conference on Robotic Computing (IRC); 2019: IEEE.
[12] A. Mohapatra, "Trust region methods for deep reinforcement learning," Analytics Vidhya, 04-Jul-2021. (Online). Available: https://medium.com/analytics-vidhya/trust-region-methods-for-deep-reinforcement-learning-e7e2a8460284. (Accessed: 07-Oct-2022).
[13] Lange, K. (2016). MM optimization algorithms. SIAM. This book offers a comprehensive introduction to MM optimization algorithms and their applications.