Dancing like the Inbetweeners: Learning to Walk with Symphony

Alexander Read - jjxc38

Abstract

This paper presents a reimplementation of the Symphony algorithm to solve the OpenAI Gym BipedalWalker-v3 environment. The Symphony algorithm is an actor-critic algorithm based on DDPG, but uses two critic networks and a single actor-network. It includes enhancements like a fading replay buffer and a Fourier series-inspired neural network architecture. In our reimplementation, we incorporate a third critic network, akin to TD3, finding this to mitigate overestimation bias, resulting in the agent learning a more 'human-like' gait at an accelerated pace. In just 28 episodes, our modified Symphony algorithm solves the normal environment, matching the highest score on the OpenAI Gym Leaderboard, and scoring higher than any publicly available code we could find. We compare the performance of the modified algorithm with an implementation of the TD3 algorithm and conclude our findings by discussing why the Symphony algorithm performs well in many OpenAI Gym environments but may not be suitable for real-life robotics applications.

1 Introduction

1.1 Background

The algorithm presented in this paper, 'modified Symphony' [12] [13], is based on the Deep Deterministic Policy Gradient (DDPG) algorithm [15] and is implemented to solve the BipedalWalker-v3 environment from the OpenAI Gym[3] an environment which involves the task of training a robot to walk. In our implementation, we incorporate a third critic network inspired by the Twin Delayed DDPG (TD3) algorithm [5]. As such this section will first introduce how actor-critic methods work, specifically deterministic policy gradient actor-critic methods.

Advancements in deep reinforcement learning (RL), particularly through off-policy actor-critic algorithms [15] [11], have significantly improved the capability of neural-network-based agents in continuous tasks. These methods operate within the framework of Markov Decision Processes (MDP) and use the Bellman equation to iteratively improve both the policy (actor) that dictates the agent's actions and the value function (critic) that evaluates the potential long-term rewards from those actions.

In deterministic policy gradient actor-critic methods, the critic estimates the action-value function $Q^\pi(s, a; w)$, which predicts the expected return of taking action a in state s under policy π. The critic updates its parameters w by minimising the temporal difference (TD) error, which is essentially the error between the predicted Q values and the values gathered from taking real actions in an environment. Formally defined: TD error $= (r + \gamma Q^\pi(s', \pi(s'; \theta); w') - Q^\pi(s, a; w))^2$ Where, r is the immediate reward, γ is a discount factor, and s' is the next state after action a.

The actor can then updates its policy parameters θ using the gradient of the action value function $Q^\pi(s, a; w)$ Formally: $\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\rho^\pi}\left[\nabla_a Q^\pi(s, a; w)\big|_{a=\pi(s;\theta)} \nabla_\theta \pi(s; \theta)\right]$

Where $\nabla_\theta J(\theta)$ is the gradient of the performance objective, the function which decides how well a policy is performing, with respect to the policy parameters.

DDPG [15] is an actor-critic algorithm that borrows concepts from Deep Q-learning[18], specifically it uses a replay buffer $\mathcal{D}$ and target networks. The replay buffer stores the transitions (s, a, r, s') after an agent samples and performs an action.

Then at every training step a mini-batch of N experiences (s_i, a_i, r_i, s'_i) are sampled uniformly at random from the replay buffer $\mathcal{D}$. The sampled experiences are used to compute the loss for updating the critic. This is done to minimise correlation between consecutive experiences, improving policy stability [16].

Target networks, defined by θ^- for the critic and ϕ^- for the actor, stabilize training by providing smoothed value estimates. They update less frequently, incorporating gradual changes from the more frequently updated online networks.

They are periodically updated with the formulas: $\theta^- \leftarrow \tau\theta + (1-\tau)\theta^-$, $\phi^- \leftarrow \tau\phi + (1-\tau)\phi^-$ where τ is a smoothing constant.

As the described policy network is deterministic, exploration is introduced by applying noise ϵ_t to an action. Thus every final action sampled for training interactions with the environment is given by: $a_t^{\text{noisy}} = a_t + \epsilon_t$

TD3 [5], was proposed to improve DDPG, specifically addressing the issue of overestimation bias. Overestimation bias occurs when estimated Q-values $Q(s, a)$ exceed true values $Q^*(s, a)$ due to the maximization in Q-learning updates: $Q(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a')$. This bias can lead to suboptimal policy choices and unstable learning. TD3's solution to this is to introduce a twin Q-network system and a technique called "delayed" policy updates. In TD3, each state-action pair (s, a) is evaluated by two separate Q-networks, and the minimum of these two estimated Q-values is used for the Bellman update. This dual Q-network approach, often called "clipped double-Q learning", helps in mitigating the overestimation by reducing the impact of noisy or outlier value estimates.

1.2 Symphony

Symphony [12] [13] is an algorithm based on DDPG and achieves some of the highest scores in many OpenAI Gym Environments [23]. Symphony versions 1 and 2 are publicly available, and we incorporate elements from both in our implementation. Although version 3 is not currently accessible, its convergence within 28 episodes for the BipedalWalker-v3 environment holds the current record on the OpenAI Gym leaderboard [23]. Our implementation produces on par results compared with the unreleased version 3 of the algorithm and adds more stability than available versions, ensuring that once convergence is achieved, it largely remains stable.

Symphony employs only a single actor network, aimed at achieving faster convergence, particularly in OpenAI Gym environments. Without a target network, the updates to the policy π and value functions incorporate the latest environmental feedback immediately, meaning the algorithm is sensitive to parameter tuning and less stable.

Both the actor and critics are represented using unique network blocks. These blocks consists of two linear layers with a combined sine-based and leaky ReLU activation function [17], mirroring a Fourier series. Fourier series inspired neural network architectures have been shown to efficiently learn and represents functions with periodic features, ideal for scenarios where an agent's states or actions are cyclic [2] [19] [8].

The final two unique features of Symphony are 1. a 'fading prioritised replay buffer' which prioritises recent experiences by adjusting their sampling probabilities, as proposed in similar work [9] [21], and 2. a custom loss function similar to the huber loss function.

The fading replay buffer is achieved using a fade function: $\text{fade}(x) = \tanh(\text{fade\_factor} \times x^2)$ where x is the normalized index of each experience in the buffer. Weights for sampling are then computed as: weight $= 1e - 7 \times \text{fade}(x)$ making recent experiences more likely to be sampled and thus focusing the learning process on newer data. This approach helps the network adapt more effectively to changes in the environment by emphasizing newer experiences over older ones.

The Rectified Huber Error (ReHE) and Rectified Huber Asymmetrical Error (ReHAE) are two custom loss functions used by the Symphony algorithm. ReHE minimizes errors by scaling the mean absolute error and applying a hyperbolic tangent function to it, denoted as $\mathrm{ReHE}(error) = \mathrm{mean}(|\mathrm{error}|) \times \tanh(\mathrm{mean}(|\mathrm{error}|))$. ReHAE, is used for policy updates but uses an asymmetrical error distributions, represented as $\mathrm{ReHAE}(error) = |\mathrm{mean}(error)| \times \tanh(\mathrm{mean}(error))$. In our experiments, we tested various loss functions however found these loss functions to outperform both a mean square error function and Huber loss function.

2 Methodology

We initially trained a TD3 model for the BipedalWalker-v3 environment. This produced satisfactory results; however, the algorithm struggled to converge over 1000 episodes. We conducted a hyperparameter search with Bayesian optimization [1], running 200 trials in the environment, eventually finding the tuned TD3 implementation shown in Figure 1, which converged at a score of 300 at approximately 600 episodes.

Further enhancements to the TD3 implementation were investigated specifically RedQ [4] and DroQ [10]. However, notable high online scores were achieved using the novel Symphony algorithm, which led to the implementation proposed in this paper. We first implemented Symphony as described in similar work [13], experimenting with network architecture, different loss functions and hyperparamaters to achieve convergence in fewer than 130 episodes. However a notable issue with our implementation was that it wasn't very stable and never converged for long periods of time in the normal environment.

As such, we 1. performed a Bayesian optimization parameter search as previously described, and 2. introduced a second critic network, with the aim of increasing the robustness of value estimation by reducing overestimation bias. Similar to TD3, updates to the target network used the lower estimate from two critic networks.

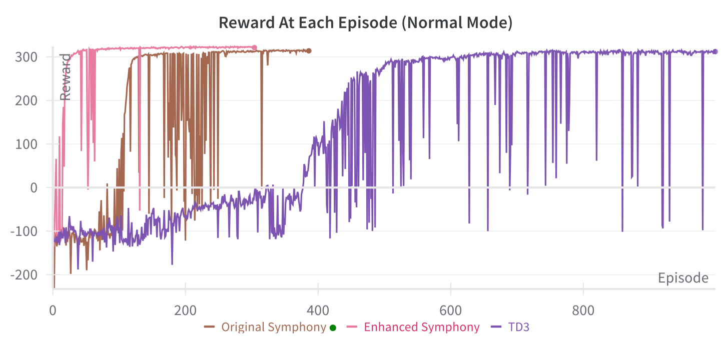
We ran all algorithms for 24 hours, which is the maximum runtime on Google Colab [6]. However some algorithms, like the modified Symphony algorithm did not complete 1000 episodes in that time. We conducted parameter searches using a Paperspace CPU enhanced machine [20] and used Weights and Biases [24] to log the training process.

3 Convergence Results

Overall our final implementation of the Symphony algorithm, far outperformed any available algorithms we could find in the normal enviroment, achieving a score of 300 at just 28 episodes. The implemented algorithm is deterministic when ran in the seeded environment, producing the same results every run. A notable difference in the stability of our enhanced Symphony implementation and the original untuned implementation is evident in Figure 1, where the performance of aforementioned algorithms is compared over 1000 episodes. The enhanced Symphony algorithm has only one unstable episode after convergence is achieved, unlike the original implementation.

Perhaps most interesting is the difference in gait between our enhanced Symphony implementation and the original implementation, visually depicted in Figure 3. We found our implementation to produce a more 'human like' gait quicker, categorised as the walking-running gait [22], learning a more natural movement and not only achieving faster convergence but also faster general scores.

However, in the hardcore environment, all of the algorithms tested were unsuccessful, with the maximum score of 43 achieved by the TD3 algorithm. Whilst this is disappointing, both original and enhanced implementations of the Symphony algorithm timed out after 24 hours at around 250 episodes; it is unclear if better performance could have been achieved. As well as this, we used the same hyperparamaters found for the normal enviroment for the agent in the hardcore environment, limited by time.



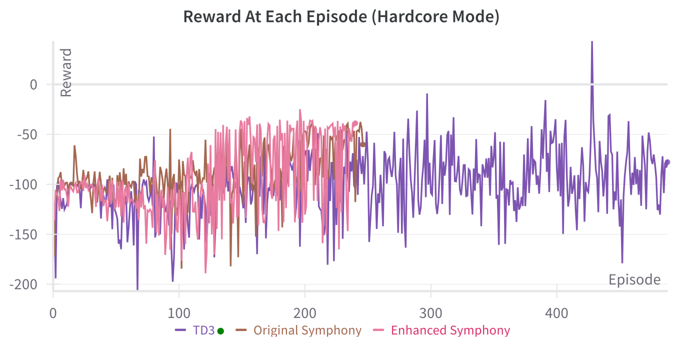
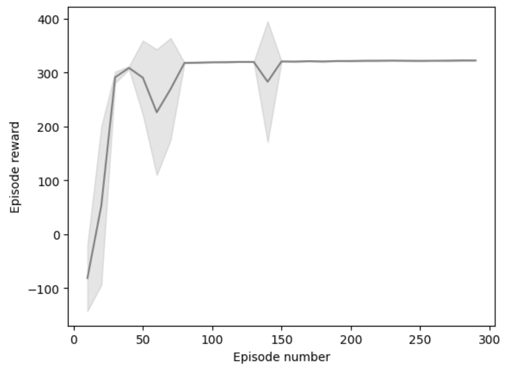


Figure 1: Comparison of TD3, Original Symphony, and Enhanced Symphony Across 1000 Episodes in the normal environment (left) and hardcore environment (right), charting the reward gathered at each episode



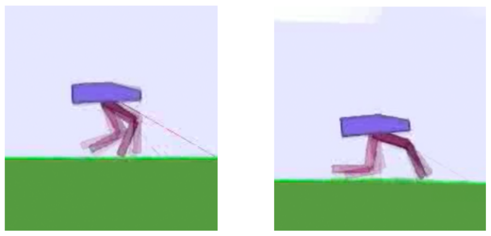


Figure 2: *
Graph charting Average Reward Every 10 Episodes using the enhanced Symphony algorithm in the normal environment

Figure 3: *
Comparison of Enhanced Symphony (left) and Symphony (right) gait in normal environment at episode 200

4 Limitations

The obvious limitation of our implementation is the failure of the agent in the hardcore environment. This is most likely a limitation of the the network architecture. Whilst Symphony performs well in many OpenAI Gym environments due to their periodic or cyclic dynamics, such as tasks requiring balance or repetitive motion. The Fourier series inspired neural network architecture has been shown to less effective in practical applications because real-world data is often noisier, less periodic, and more irregular [13]. As such this would be the next point of investigation to try and achieve better results in the hardcore environment. Another limitation is the excessive time the agent takes to train, when compared to algorithms such as TD3. This is a limitation possibly imposed from our introduction of a third critic network.

Future Work

In truth a large proportion of our efforts went into training an agent for optimal performance in the normal environment, thus future work involves running a parameter search for the hardcore environment, and further evaluating the performance of our proposed implementation. If not limited by time, it would be interesting to explore if the algorithm does manage to solve the hardcore environment. If not then further research could go into investigating the performance of known enhancements on top of TD3 or other algorithms such as SAC [7], including RedQ [4], DroQ [10] and TQC [14].

References

[1] Takuya Akiba et al. "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM. 2019, pp. 2623–2631.

[2] David Brellmann, David Filliat, and Goran Frehse. "Fourier Features in Reinforcement Learning with Neural Networks". In: *Transactions on Machine Learning Research Journal* (2023). ffhal-04316346f.

[3] Greg Brockman et al. "Openai gym". In: *arXiv preprint arXiv:1606.01540* (2016).

[4] Xinyue Chen et al. "Randomized ensembled double q-learning: Learning fast without a model". In: *arXiv preprint arXiv:2101.05982* (2021).

[5] Scott Fujimoto, Herke Hoof, and David Meger. "Addressing function approximation error in actor-critic methods". In: *International conference on machine learning*. PMLR. 2018, pp. 1587–1596.

[6] Google. *Google Colaboratory*. `https://colab.research.google.com/`.

[7] Tuomas Haarnoja et al. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *International conference on machine learning*. PMLR. 2018, pp. 1861–1870.

[8] Bing Han, Cheng Wang, and Kaushik Roy. "Oscillatory fourier neural network: A compact and efficient architecture for sequential processing". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 6. 2022, pp. 6838–6846.

[9] Matteo Hessel et al. "Rainbow: Combining improvements in deep reinforcement learning". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.

[10] Takuya Hiraoka et al. "Dropout q-functions for doubly efficient reinforcement learning". In: *arXiv preprint arXiv:2110.02034* (2021).

[11] Julian Ibarz et al. "How to train your robot with deep reinforcement learning: lessons we have learned". In: *The International Journal of Robotics Research* 40.4-5 (2021), pp. 698–721.

[12] Timur Ishuov. *Simphony*. `https://github.com/timurgepard/Simphony/tree/main`. Accessed: 2023-05-04. 2023.

[13] Timur Ishuov and Michele Folgheraiter. *How to Make Robot Move Like a Human with Reinforcement Learning*. Ed. by Abdubakir Qo'shboqov. Kindle Edition. Amazon Digital Services LLC, 2023.

[14] Arsenii Kuznetsov et al. "Controlling overestimation bias with truncated mixture of continuous distributional quantile critics". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 5556–5566.

[15] Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[16] Shijie Liu. "An Evaluation of DDPG, TD3, SAC, and PPO: Deep Reinforcement Learning Algorithms for Controlling Continuous System". In: *2023 International Conference on Data Science, Advanced Algorithm and Intelligent Computing (DAI 2023)*. Atlantis Press. 2024, pp. 15–24.

[17] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. "Rectifier nonlinearities improve neural network acoustic models". In: *Proc. icml*. Vol. 30. 1. Atlanta, GA. 2013, p. 3.

[18] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[19] Marieme Ngom and Oana Marin. "Fourier neural networks as function approximators and differential equation solvers". In: *Statistical Analysis and Data Mining: The ASA Data Science Journal* 14.6 (2021), pp. 647–661.

[20] Paperspace Co. *Paperspace*. `https://www.paperspace.com/`.

[21] Tom Schaul et al. "Prioritized experience replay". In: *arXiv preprint arXiv:1511.05952* (2015).

[22] Jonah Siekmann et al. "Sim-to-real learning of all common bipedal gaits via periodic reward composition". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 7309–7315.

[23] timurgepard and contributors. *Leaderboard - OpenAI Gym Wiki*. `https://github.com/openai/gym/wiki/Leaderboard`. Accessed: 2024-05-06. 2024.

[24] Weights and Biases Inc. *Weights and Biases*. `https://wandb.com/`.