[Replication] A Unified Bellman Optimality Principle Combining Reward Maximization and Empowerment

Akhil Bagaria
Department of Computer Science
Brown University
akhil_bagaria@brown.edu

Seungchan Kim
Department of Computer Science
Brown University
seungchan_kim@brown.edu

Alessio Mazzetto
Department of Computer Science
Brown University
alessio_mazzetto@brown.edu

Rafael Rodriguez-Sanchez
Department of Computer Science
Brown University
rrs@brown.edu

Abstract

Designing learning agents that gain broad competence in a self-motivated manner is a longstanding goal of reinforcement learning. *Empowerment* is a task-agnostic information-theoretic quantity that has recently been used to intrinsically motivate reinforcement learning agents. Leibfried et al. 2019 [1] showed how to combine empowerment with traditional task-specific reward maximization. In this work, we replicate the main empirical results of their paper. In particular, we reproduce the main algorithm of the paper, empowered actor-critic (EAC) and compare its performance with state-of-the-art baselines: soft actor-critic (SAC), proximal policy optimization (PPO), and deep deterministic policy gradients (DDPG) on a series of continuous control tasks in the MuJoCo simulator. We find that the performance of our implementation of EAC closely follows that of the original paper. However, our empirical findings also suggest that EAC is unable to improve upon baseline actor-critic algorithms . We share our code, raw learning curves and the scripts used to produce the figures in this paper [1].

1 Introduction

Reinforcement Learning (RL) has been used to learn complex behaviors in challenging tasks [2–7]. In most cases, the RL agent learns behaviors that maximize a task-specific reward function. While agents trained using such extrinsic rewards may perform well on the task in which they were trained, they must usually be trained from scratch in a different, albeit related task. Intrinsic motivation (IM) augments the classical RL paradigm so that agents may develop broad competence even in the absence of such extrinsic rewards [8–11]. In the multi-task setting, IM allows RL agents to generalize their learning across a series of related problems [12–15]. In the single-task setting, IM allows RL agents to discover solutions with better asymptotic performance [16–18].

Several mathematical formulations have been proposed to capture the notion of intrinsic motivation as a computable quantity [19–23]. One such formalism of IM is known as *empowerment*. Maximizing empowerment amounts to maximizing the agent's potential to force the world into states that it can reliably distinguish [24]. In other words, empowerment describes an embodied agent's ability to control perceivable elements of its environment. Slightly more formally, empowerment is the maximum amount of information [25] that an agent can send from its actuators to its sensors

[1] https://github.com/eac-replication/eac-replication

Preprint. Under review.

through its environment [26]. In the reinforcement learning literature, empowerment has been computed as the mutual information [27] between an action sequence and the state achieved after executing that action sequence. Klyubin et al. [24] showed that this mutual information term is maximized at states where the number of reachable states is largest.

While others have employed empowerment driven exploration [28–30], Leibfried et al. 2019 [1] develop a unified RL objective that combines both empowerment and explicit reward signals. First, they derive a Bellman backup operator that jointly maximizes both empowerment and extrinsic rewards. Then, they outline how the combined objective can be used in model-free RL settings to solve continuous control problems. In this paper, we re-implement their described algorithm, provide clarifications on implementation details, and finally attempt to reproduce their main empirical results. In doing so, we compare our implementation of Empowered Actor-Critic (EAC) to popular model-free deep RL algorithms [31–33] in benchmark continuous control tasks in the MuJoCo physics simulator [34, 35].

2 Background

Sequential decision making problems may be modeled as Markov Decision Processes (MDPs). An MDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, R, \gamma)$, where $\mathcal{S}$ denotes the state space, $\mathcal{A}$ denotes the action space, $\mathcal{T}(s, a, s')$ represents the probability of next state s' given that the agent takes action a in state s; $R(s, a)$ yields the expected reward for taking action a from state s; and $\gamma \in (0, 1)$ is the discount factor, which indicates how much an agent should prioritize immediate rewards.

Given a state $s \in \mathcal{S}$, the agent follows the policy π if it takes an action $a \in \mathcal{A}$ according to a conditional distribution $\pi(a|s)$. In reinforcement learning, the goal is to find a policy that maximizes the expected future cumulative reward, i.e. $\operatorname{argmax}_\pi \mathbb{E}_{\pi,\mathcal{T}}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$, where r_t is the reward obtained at time step t. By using Bellman's optimality principle [36], we have that the optimal expected future cumulative reward, starting from a states $s \in \mathcal{S}$, is given by:

$$V^*(s) = \max_a \left(R(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s, a, s') V^*(s') \right) = \max_a Q^*(s, a) \tag{1}$$

where V^* and Q^* are referred to as the optimal value and optimal action-value functions, respectively. The value of following a policy π is given by:

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s, a, s') V^\pi(s') \right) \tag{2}$$

A policy π^* is optimal if it holds that $V^{\pi^*}(s) = V^*(s)$ for any $s \in \mathcal{S}$.

2.1 Policy-Search Algorithms

Recent work in reinforcement learning has successfully used non-linear function approximation to solve problems with high dimensional state and action spaces. Deep Q-Networks (DQN) [4] learn impressive behaviors in problems with continuous states and discrete actions. Deep Deterministic Policy Gradients (DDPG) [37] is a model-free actor-critic algorithm that extends deep Q-learning to continuous action spaces. Twin Delayed DDPG (TD3) [32] mitigates the overestimation problem in deep Q-learning [38, 39] by maintaining two independent copies of the Q-function. They showed superior performance to vanilla DDPG on benchmark continuous control tasks in the MuJoCo simulator.

Another line of work has focused on addressing the challenges of extending policy gradient methods [40, 41] to high dimensional problems. Trust-Region Policy Optimization (TRPO) [42], a policy-search algorithm addresses the problems of high variance policy gradients and large policy changes between learning updates. Proximal Policy Optimization (PPO) improves upon TRPO by augmenting their objective function with clipped probability ratios [33].

Soft actor-critic (SAC) [31] shows how the actor-critic formulation can be used to learn policies in the entropy-regularized deep RL setting. Intuitively, SAC encourages exploration by explicitly encoding the exploration-exploitation dilemma into the RL objective. It does so by solving for a policy that maximizes extrinsic rewards as well as its entropy, thus having a more generalized version of utility:

$$\mathbb{E}_{(s_t,a_t)\sim\rho_\pi}\left[R(s_t,a_t)+\alpha H\left(\pi(\cdot|s_t)\right)\right], \tag{3}$$

where H is the entropy function [27] and $\rho_\pi(s)$ is the transition dynamics induced by the policy $\pi(a|s)$. The SAC algorithm uses function approximators to learn value function V (parametrized by ψ), action-value function Q (parametrized by θ), and policy function π (parametrized by ϕ). This is done by minimizing the following loss functions:

$$J_V(\psi)=\mathbb{E}_{s_t\sim D}\left[\frac{1}{2}(V_\psi(s_t)-\mathbb{E}_{a_t\sim\pi_\phi}[Q_\theta(s_t,a_t)-\log\pi_\phi(a_t|s_t)])^2\right] \tag{4}$$

$$J_Q(\theta)=\mathbb{E}_{(s_t,a_t)\sim D}\left[\frac{1}{2}(Q_\theta(s_t,a_t)-\hat{Q}\left(s_t,a_t\right))^2\right] \tag{5}$$

$$J_\pi(\phi)=\mathbb{E}_{s_t\sim D,\epsilon_t\sim\mathcal{N}}\left[\log\pi_\phi(f_\phi(\epsilon_t;s_t)|s_t)-Q_\theta(s_t,f_\phi(\epsilon_t;s_t))\right] \tag{6}$$

where D is the replay buffer [43] and $\hat{Q}(s_t,a_t)$ is the target Q value (see [31] for a more detailed definition). In particular, Equation 6 defines loss function for learning the policy π_ϕ using the "reparametrization trick", where f_ϕ transforms the noise ϵ_t from a standard normal $\mathcal{N}(0,1)$ to the normal distribution predicted by π_ϕ.

2.2 Empowerment and Intrinsic Motivation

Consider an agent which starting from state s takes a sequence of actions $\mathbf{a}=(a_1,...,a_k)$ to finally reach state s'. The mutual information $I[\mathbf{a},s'|s]$ describes the conditional dependence between the agent's action sequence $\mathbf{a}$ and its final state s'. Maximizing empowerment in RL is equivalent to maximizing this mutual information term. In other words, the empowerment policy $\pi_k:\mathcal{S}\times\mathcal{A}^k\rightarrow[0,1]$ is defined as a conditional distribution $\pi_k(\mathbf{a}|s)$ that maximizes $I[\mathbf{a},s'|s]$.

Let $\mathcal{T}^{(k)}(s,\mathbf{a},s')$ be the k-step transition probability, that is the probability to reach state s' performing the sequence of k actions $\mathbf{a}$ starting from state s. Then, for every $s\in\mathcal{S}$, and a policy π_k, we can define the following quantity:

$$\mathbb{E}^{\pi_k}(s)=I[\mathbf{a},s'|s]=\mathbb{E}_{\pi_k(\mathbf{a}|s)\mathcal{T}^{(k)}(s,\mathbf{a},s)}\left[\log\frac{p(\mathbf{a}|s',s)}{\pi_k(\mathbf{a}|s)}\right] \tag{7}$$

The quantity $\mathbb{E}^{\pi_k}(s)$ is the expected empowerment value of an agent following the policy π_k starting from state s. The function p is the inverse dynamics model of π_k and is defined as $p(\mathbf{a}|s',s)=\frac{\mathcal{T}^{(k)}(s,\mathbf{a},s')\pi_k(\mathbf{a}|s)}{\sum_{\mathbf{a}}\mathcal{T}^{(k)}(s,\mathbf{a},s')\pi_k(\mathbf{a}|s)}$. The optimal empowerment values are obtained by the policy π^* that maximizes E^{π^*}.

The core idea of Leibfried et al's work [1] is to add the empowerment term (with $k=1$, i.e. using one-step policy) as an additional contribution to the reward. In particular, we seek a policy that maximizes the following quantity:

$$\max_\pi\mathbb{E}_{\pi,\mathcal{T}}\left[\sum_{t=0}^{\infty}\gamma^t\left(\alpha\cdot R(s_t,a_t)+\beta\cdot\log\frac{p(a_t|s_{t+1},s_t)}{\pi(a_t|s)}\right)\right] \tag{8}$$

where α and β are two non-negative hyperparameters that are used to scale the reward and the empowerment term. The equation above extends the normal definition of the MDP by adding an additional term to the reward, for which the authors [1] show the existence of unique values for the value function and the convergence of value iteration.

3 Empowered Actor-Critic

Empowered Actor-Critic (EAC) algorithm combines the concept of empowerment with reward maximization. EAC is built upon the SAC algorithm baseline, with slightly modified versions of J_V, J_Q, J_π objectives in SAC. It also additionally learns inverse dynamics model $p(\mathbf{a}|s',s)$

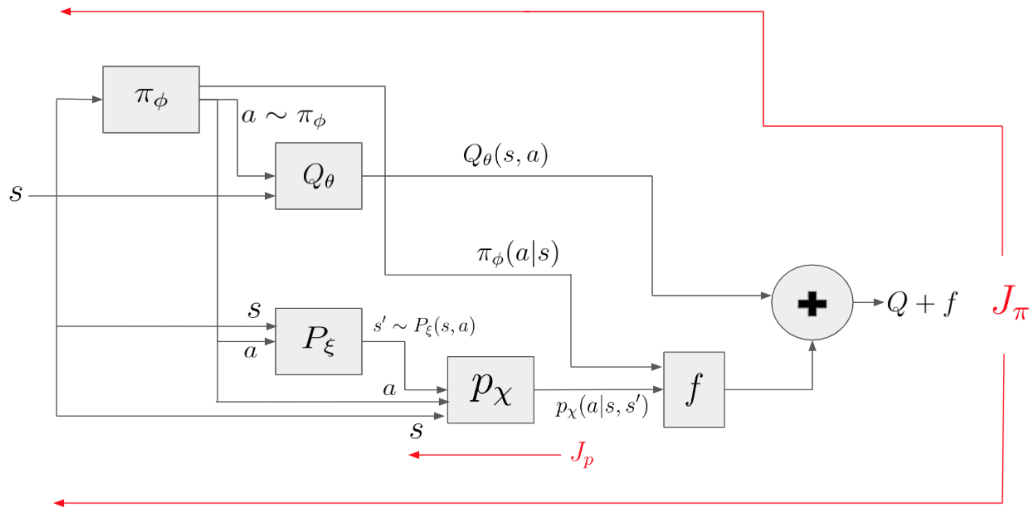

Figure 1: Computational graph needed to learn the policy π_ϕ and the inverse dynamics model p_χ. s denotes the current state, a is an action sampled from the current policy π_ϕ and s' denotes the next state sampled from the learned transition model P_ξ. f represents the empowerment term from section 3. Black lines denote computations involved in the forward-pass through the network architecture; the red lines represent the backward pass.

and transition dynamics model $T(s,a,s')$ by optimizing inverse dynamics objective J_p and transition dynamics objective J_P. EAC uses the same objective function $J_Q(\theta)$ as in SAC. As for the value objective J_V, we add a new quantity $\beta f(s,a)$, which is defined as $f(s,a) = \mathbb{E}_{P_\xi(s'|s,a)}[\log p_\chi(a|s',s) - \log \pi_\phi(a|s)]$. This term replaces the negative log policy term in the J_V of SAC. Similarly in the policy objective J_π, the log probability distribution of policy term is replaced with $\beta f(s,a)$ term. The modified form of J_V and J_π in EAC are described below:

$$J_V(\psi) = \frac{1}{B}\sum_{b=1}^{B}\left[\frac{1}{2}\left(V_\psi(s_b) - \mathbb{E}_{\pi_\phi(a|s_b)}\left[Q_\theta(s_b,a) + \beta f(s_b,a)\right]\right)^2\right] \tag{9}$$

$$J_\pi(\phi) = -\frac{1}{B}\sum_{b=1}^{B}\mathbb{E}_{\pi_\phi(a|s_b)}\left[Q_\theta(s_b,a) + \beta f(s_b,a)\right], \tag{10}$$

where B is the size of batch tuples of (s,a,r,s') that we sample from replay buffer. Additionally, EAC employs two new objectives for inverse dynamics and transition dynamics model as below:

$$J_p(\chi) = -\frac{1}{B}\sum_{b=1}^{B}\mathbb{E}_{\pi_\phi(a|s_b)P_\xi(s'|s_b,a)}\left[\log p_\chi(a|s',s_b)\right] \tag{11}$$

$$J_P(\xi) = -\frac{1}{B}\sum_{b=1}^{B}\log(P_\xi(s'_b|s_b,a_b)) \tag{12}$$

At every iteration, EAC draws batches from its replay buffer, and updates Q-critic, V-critic, policy π, inverse dynamics, and transition model, using the five objective functions, $J_Q, J_V, J_\pi, J_p, J_P$.

4 Reproducibility

4.1 Model Architecture

In addition to the quantities approximated by SAC, EAC seeks to learn the quantities f, P_ξ and p_χ. While implementing the EAC algorithm, we found it challenging to set up the loss functions J_π and

J_p. As a result, we include a brief description of how these quantities were computed. We hope that this discussion will aid others in the community seeking to replicate EAC.

Figure 1 suggests that the policy loss J_π contributes to gradients that are used for learning all functions in our computation graph. By contrast, the gradient of the inverse dynamics loss J_p only backpropagating through the inverse dynamics model p_χ. This is apparent from the following expressions for the gradient of the two loss functions:

$$\nabla_\phi J_\pi(\phi) = -\frac{1}{B}\sum_{b=1}^{B} \nabla_\phi \mathbb{E}_{\pi_\phi(a|s_b)}\left[Q_\theta(s_b, a) + \beta f(s_b, a)\right] \tag{13}$$

$$\begin{aligned}\nabla_\chi J_p(\chi) &= -\frac{1}{B}\sum_{b=1}^{B} \nabla_\chi \mathbb{E}_{\pi_\phi(a|s_b)P_\xi(s'|s_b,a)}\left[\log p_\chi(a|s', s_b)\right] \\ &= -\frac{1}{B}\sum_{b=1}^{B} \mathbb{E}_{\pi_\phi(a|s_b)P_\xi(s'|s_b,a)} \nabla_\chi\left[\log p_\chi(a|s', s_b)\right]\end{aligned} \tag{14}$$

Equation 13 shows that the gradient of J_π with respect to the model parameters cannot be taken inside the expectation without using the re-parameterization trick [31]. Even after using the re-parameterization trick, one has to backpropagate through all functions in the computational graph. However, Equation 14 shows that the gradient with respect to the model parameters *can* be taken inside the expectation. This difference in gradient computation informs an important implementation detail: when computing the loss function $J_p(\chi)$, all other parts of the computational graph must be detached [2].

4.2 Learning the transition model

In Figure 2, we report the loss of the transition model for the environment Ant-v2. We observe that the loss is very large with values in the order of 10^9—results in similar order of magnitude are also obtained for the other domains. In Equation 12, we see that this loss represents an average negative log-likelihood of our prediction which means that our transition model assigns an incredibly small probability to the next state s' from the experience tuples. The difficulty of learning a transition model in MuJoCo is well supported in the RL literature. For example, Nagabandi *et al.* [44] argue that learning a function $h_\theta(s_t, a_t)$ that predicts the next state s_{t+1} in an MDP with complicated dynamics is difficult using a feed-forward neural network—as is attempted in the EAC experiments.

4.3 Learning the Inverse Dynamics Model

Figure 1 shows that training the inverse dynamics function p_χ depends on the state s' sampled from P_ξ. If the current state s and the sampled state s' have little to do with each other, we cannot hope to derive a useful learning signal for training the inverse dynamics model p_χ. In other words, the difficulty of learning a good transition model P_ξ leads to an under-performing inverse dynamic model p_χ. In practice, this leads to p_χ predicting actions that uniformly have a very low likelihood under the learned model. This further leads to numerical approximation problems when computing the log-likelihood in the objective function $J_p(\chi)$ in Equation 12. While this problem is somewhat mitigated by bounding the log standard deviation [1, 45], it highlights the difficulty of learning p_χ and P_ξ in complex dynamical systems such as those considered in this paper.

5 Experimental Setup

Following the experimental setup of Leibfried et al. 2019 [1], we compare our implementation of EAC with SAC, PPO, and TD3 in different environments of the robotics simulator MuJoCo: Ant-v2, HalfCheetah-v2, Hopper-v2, Humanoid-v2, and Walker2d-v2. Throughout this section, assume that the the experimental details are the same as in the original paper [1], unless otherwise stated.

[2] `https://pytorch.org/docs/stable/autograd.html`

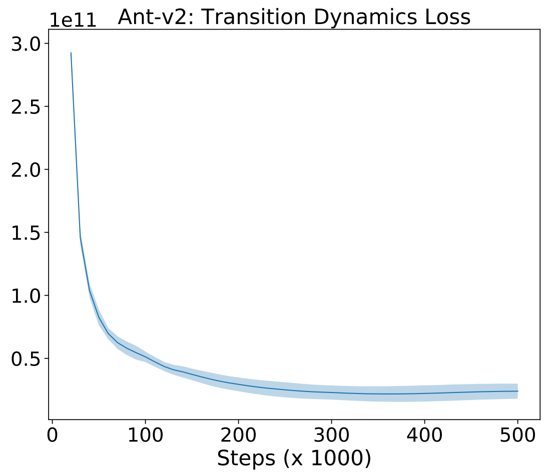

Figure 2: Loss function (cross-entropy) for the Transition Dynamic model learned by EAC in the Ant-v2 environment. This curve is averaged over 5 runs of EAC and the shaded area is the standard error. Note the high scaling factor on the vertical axis.

5.1 Baseline Implementations

Through extensive experimentation, Henderson et al show that different popular implementations of common deep RL baselines lead to dramatically different results [46]. Consequently, we argue that while comparing against other algorithms, it is important to pay close attention to (a) the choice of codebase used to reproduce baseline results and (b) the hyperparameters for all baseline algorithms. In some cases, our choice of baseline implementation differs from the ones reported by the Leibfried et al because these optimized choices allow for fairer comparison. In this section, we describe the design decisions we made for each of the baselines that EAC was compared to.

Comparison with DDPG: Leibfried et al compare their EAC algorithm with DDPG. However, in Appendix C.2 of their paper, they state that they use the hyperparameters outlined in the TD3 paper [32]. Moreover, they used the same model architecture as EAC and SAC. Given the sensitivity of deep RL models to the particular choice of network architecture and hyperparameter settings [46], we argue that this design decision made by Leibfried et al leads to an unfair comparison. By using the author implementation of the baseline being compared against, we observe significantly stronger performance in the tasks considered here.

Furthermore, we chose to compare the proposed algorithm against TD3 [3] as opposed to a vanilla DDPG. We made this experimental decision because of the following reasons:

- In Appendix C.2, Leibfried et al state that they compared against a DDPG model which used the hyperparameters from the TD3 paper. Since TD3 is often seen as a variant of the DDPG algorithm, this could be taken to mean that they compared against TD3, which they called DDPG in their main paper to refer to the general class of algorithms that learn deterministic policies in high dimensional continuous state and action spaces.
- TD3 addresses function approximation errors introduced by the vanilla implementation of DDPG and shows superior performance in continuous control problems in MuJoCo environments [32]. Since TD3 outperformed DDPG, we argue that TD3 represents a more suitable actor-critic baseline than DDPG in MuJoCo environments.
- Much like SAC and EAC, TD3 implements the double Q-learning idea by learning two copies of the critic. As a result, it makes more sense to compare EAC with TD3 than DDPG as it eliminates the possibility that the difference in performance between the algorithms could be because of the over-estimation bias when learning Q-functions in the function approximation setting [39].

[3]Author implementation of TD3: `https://github.com/sfujim/TD3`

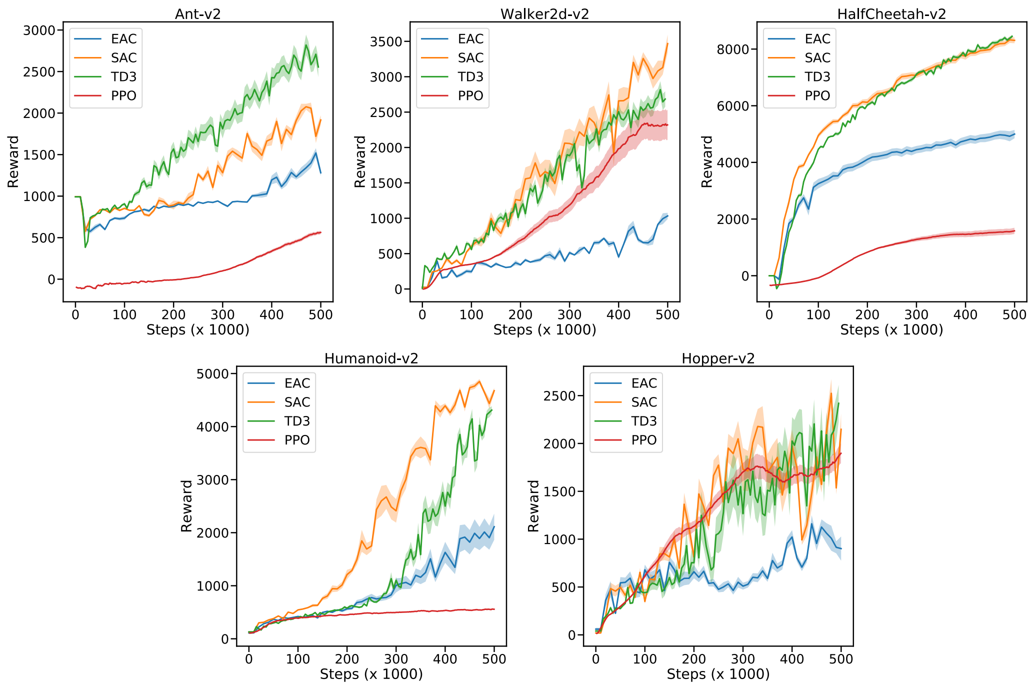

Figure 3: Learning curves comparing our implementation of Empowered Actor-Critic [1] (EAC), Soft Actor-Critic (SAC), Twin Delayed Deep Deterministic Policy Gradients (TD3) and Proximal Policy Optimization (PPO) on benchmark continuous control tasks in the MuJoCo simulator. All curves are averaged over 5 runs, with shaded areas denoting standard error.

	Ant	Walker	HalfCheetah	Humanoid	Hopper
Our EAC	**1670.6**	1032.8	4999.2	**2112.1**	**1009.7**
Their EAC (approximate)	1600	1500	3000	2200	1000

Table 1: Comparison between our EAC results and the original EAC results [1] (reported). Bold values represent similarity with the original paper.

Comparison with SAC: We use the implementation of SAC in *rlpyt* [47] with the same network architecture and hyperparameters reported by Leibfried et al. As we show in Figure 3, our baseline implementation of SAC significantly outperforms the results reported in the original EAC paper.

Comparison with PPO: Leibfried et al use the same network architecture as their EAC algorithm, while using the same hyperparameters that were reported in the PPO paper. Since deep RL algorithms are sensitive to the combination of architecture-hyperparameter *pairs*, we argue that using the architecture and hyperparameters reported in the PPO paper makes for a fairer comparison. In the case of PPO, we use OpenAI Baselines with default hyperparameters [48], which is considered to be community standard in deep RL research.

6 Results

We tested our implementation of EAC on five continuous control tasks in the OpenAI gym framework [35]: Ant-v2, Walker-v2, HalfCheetah-v2, Humanoid-v2 and Hopper-v2. All of these tasks involve learning good gait policies in continuous state and action spaces. The learning curves of EAC and three other baseline algorithms are presented in Figure 3.

First, we computed average rewards of our EAC implementation and compared them with the original results of their EAC implementations. The results are described in the Table 6. In Ant-v2, Humanoid-v2, and Hopper-v2, we observed that our results are very similar to the average rewards reported in the original EAC paper. In HalfCheetah-v2, our EAC results outperformed their reported results, while in Walker2d-v2, our implementation of EAC slightly under-performed with respect to theirs.

However, when comparing to the baseline algorithms, our results were significantly different from those reported in the original paper. In the original paper, EAC sometimes outperforms the SAC, PPO, and DDPG baselines. Although our EAC results were similar to theirs, our EAC never performed better than the baselines. For example, in all five domains, SAC baseline yielded higher results than our EAC with a significant margin. We also note that our baseline performances were closer to those reported in the original papers that introduced them. For example, our SAC baseline results are closer to the ones reported in the original SAC paper [31].

7 Conclusion

We presented a reproducibility analysis of the paper *A Unified Bellman Optimality Principle Combining Reward Maximization and Empowerment* [1]. We implemented the main algorithm of the paper EAC, and compared its performance to state-of-the-art baselines such as SAC, TD3 and PPO. We described missing implementation details that we found to be critical in reproducing the original paper. Furthermore, we discussed a fundamental shortcoming of the EAC algorithm – the difficulty of learning an effective transition model for sufficiently complex dynamical systems. Finally, as suggested by Figure 3 and Table 1, our implementation of EAC yields results similar to the ones reported in the original paper. However, we find that our baselines perform significantly better than those reported by Leibfried et al. Consequently, we conclude that our implementation of EAC is unable to outperform baseline actor-critic algorithms.

References

[1] Felix Leibfried, Sergio Pascual-Diaz, and Jordi Grau-Moya. A unified bellman optimality principle combining reward maximization and empowerment. *33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.*, 2019.

[2] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[3] Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[5] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[6] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[7] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in neural information processing systems*, pages 1–8, 2007.

[8] Andrew G. Barto. *Intrinsic Motivation and Reinforcement Learning*, pages 17–47. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[9] Nuttapong Chentanez, Andrew G Barto, and Satinder P Singh. Intrinsically motivated reinforcement learning. In *Advances in neural information processing systems*, pages 1281–1288, 2005.

[10] Satinder Singh, Richard L Lewis, Andrew G Barto, and Jonathan Sorg. Intrinsically motivated reinforcement learning: An evolutionary perspective. *IEEE Transactions on Autonomous Mental Development*, 2(2):70–82, 2010.

[11] Andrew Stout, George D Konidaris, and Andrew G Barto. Intrinsically motivated reinforcement learning: A promising framework for developmental robot learning. Technical report, Massachusetts Univ Amherst Dept of Computer Science, 2005.

[12] Andrew G Barto, Satinder Singh, and Nuttapong Chentanez. Intrinsically motivated learning of hierarchical collections of skills. In *Proceedings of the 3rd International Conference on Development and Learning*, pages 112–19, 2004.

[13] Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. *arXiv preprint arXiv:1703.05407*, 2017.

[14] Leslie Pack Kaelbling. Learning to achieve goals. In *IJCAI*, pages 1094–1099. Citeseer, 1993.

[15] George Konidaris and Andrew Barto. Autonomous shaping: Knowledge transfer in reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 489–496. ACM, 2006.

[16] Jürgen Schmidhuber. Self-motivated development through rewards for predictor errors/improvements. In *Developmental Robotics AAAI Spring Symposium, Stanford, CA*, 2005.

[17] Özgür Şimşek and Andrew G Barto. An intrinsic reward mechanism for efficient exploration. In *Proceedings of the 23rd international conference on Machine learning*, pages 833–840. ACM, 2006.

[18] Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. *arXiv preprint arXiv:1707.05300*, 2017.

[19] Alexander L Strehl and Michael L Littman. An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, 2008.

[20] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 1471–1479, 2016.

[21] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

[22] Özgür Şimşek and Andrew G Barto. Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 95. ACM, 2004.

[23] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

[24] Alexander S Klyubin, Daniel Polani, and Chrystopher L Nehaniv. All else being equal be empowered. In *European Conference on Artificial Life*, pages 744–753. Springer, 2005.

[25] Claude Elwood Shannon. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423, 1948.

[26] Alexander S. Klyubin, Daniel Polani, and Chrystopher L. Nehaniv. Keep your options open: An information-based driving principle for sensorimotor systems. *PLOS ONE*, 3:1–14, 12 2008.

[27] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[28] Ildefons Magrans de Abril and Ryota Kanai. A unified strategy for implementing curiosity and empowerment driven reinforcement learning. *arXiv preprint arXiv:1806.06505*, 2018.

[29] Navneet Madhu Kumar. Empowerment-driven exploration using mutual information estimation.

[30] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems*, pages 1109–1117, 2016.

[31] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 1856–1865, 2018.

[32] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1582–1591, 2018.

[33] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[34] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[35] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[36] Richard Bellman et al. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503–515, 1954.

[37] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[38] Hado V Hasselt. Double q-learning. In *Advances in Neural Information Processing Systems*, pages 2613–2621, 2010.

[39] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[40] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[41] Sham M Kakade. A natural policy gradient. In *Advances in neural information processing systems*, pages 1531–1538, 2002.

[42] John Schulman, Sergey Levine, Pieter Abbeel, Michael I. Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 1889–1897, 2015.

[43] Long-Ji Lin. Reinforcement learning for robots using neural networks. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.

[44] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.

[45] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pages 4754–4765, 2018.

[46] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters, 2017.

[47] Adam Stooke and Pieter Abbeel. rlpyt: A research code base for deep reinforcement learning in pytorch. *CoRR*, abs/1909.01500, 2019.

[48] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. `https://github.com/openai/baselines`, 2017.