Online weighted Q-ensembles for reduced hyperparameter tuning in reinforcement learning

Online Weighted Q-Ensembles for Reduced Hyperparameter Tuning in Reinforcement Learning

Renata Garcia[1] and Wouter Caarls[1*]

[1]Department of Electrical Engineering, Pontifical Catholic University of Rio de Janeiro, Rua Marquês de São Vicente 225, Rio de Janeiro, 22451-900, RJ, Brazil.

*Corresponding author(s). E-mail(s): wouter@ele.puc-rio.br;
Contributing authors: renata.garcia.eng@gmail.com;

Abstract

Reinforcement learning is a promising paradigm for learning robot control, allowing complex control policies to be learned without requiring a dynamics model. However, even state of the art algorithms can be difficult to tune for optimum performance. We propose employing an ensemble of multiple reinforcement learning agents, each with a different set of hyperparameters, along with a mechanism for choosing the best performing set(s) on-line. In the literature, the ensemble technique is used to improve performance in general, but the current work specifically addresses decreasing the hyperparameter tuning effort. Furthermore, our approach targets on-line learning on a single robotic system, and does not require running multiple simulators in parallel. Although the idea is generic, the Deep Deterministic Policy Gradient was the model chosen, being a representative deep learning actor-critic method with good performance in continuous action settings but known high variance. We compare our online weighted q-ensemble approach to q-average ensemble strategies addressed in literature using alternate policy training, as well as online training, demonstrating the advantage of the new approach in eliminating hyperparameter tuning. The applicability to real-world systems was validated in common robotic benchmark environments: the bipedal robot half cheetah and the swimmer. Online Weighted Q-Ensemble presented overall lower variance and superior results when compared with q-average ensembles using randomized parameterizations.

Keywords: Reinforcement Learning, Deep Reinforcement Learning, Ensemble Algorithms, Hyperparameter Optimization.

MSC Classification: 68T40 , 68T07 , 68T05

Category: (5)

1 Introduction

Reinforcement learning (RL) is based on a mathematical framework known as a Markov Decision Process (MDP) (Sutton and Barto, 2018). Control policies for many common domains such as motor control (Liu et al, 2021), treatment planning (Watts et al, 2020), autonomous driving (Tammewar et al, 2023), and disease spread prediction (Khalilpourazari and Doulabi, 2021) can be optimized under this framework, by maximizing a reward signal in order to achieve a goal. RL can be distinguished from other optimal control approaches such as dynamic programming by the fact that an *a priori* model of the environment is not required. Recent advances in reinforcement learning using deep neural networks (Arulkumaran et al, 2017) allow it to be applied to increasingly complex environments. In robotics, this means optimizing more complex motions, requiring the simultaneous actuation of many joints. In these environments, the state space (joint positions and velocities) is continuous.

Real-world domains have dangerous or slow interactions, i.e., learning from scratch through online interactions with the real-world is highly time-consuming or highly expensive (Dulac-Arnold et al, 2021; Han et al, 2023), risking damaging the robot (Meijdam et al, 2013; Koryakovskiy et al, 2017). To this end, off-policy algorithms are used, which constantly re-use data collected in previous interactions with the environment, reducing iterations during learning. For discrete actions, the DQN (Deep Q-Learning) algorithm (Mnih et al, 2015) has shown good performance results for Atari games, while for continuous actions such as robotics, the DDPG (Deep Deterministic Policy Gradient) (Lillicrap et al, 2016) method and its variants TD3 (Fujimoto et al, 2018) and SAC (Haarnoja et al, 2018) are more suitable. All these solutions utilizing deep learning algorithms need fine-tuning of their hyperparameters to converge. One approach is to perform grid search (Oliveira. and Caarls., 2020) or genetic search (Cardeñoso Fernandez and Caarls, 2018) to automate the tuning. These algorithms have a high computational cost, and are difficult to apply if the optimization is to take place in the real world instead of in simulation.

Alternatively, an *ensemble* of different sets of hyperparameters can be trained to decrease the hyperparameter tuning effort (Oliveira. and Caarls., 2021). Ensembles were first used in RL before the advent of deep learning techniques to increase performance (Wiering and Van Hasselt, 2008; Hans and Udluft, 2010; Duell and Udluft, 2013), and recent efforts have shown that ensemble aggregations of deep neural networks perform better than a single algorithm as well (Wu and Li, 2020; Lee et al, 2021; Song et al, 2023). Some proposed ensemble aggregations have additional parameters, which add more variables to be fine-tuned, while others present a population-based approach to improve performance (Jung et al, 2020).

While ensembles in deep RL thus demonstrated good results when used to increase performance, there have been few studies of the behavior of RL ensembles with different hyperparameters, aimed at reducing the tuning effort. The history-based framework in Oliveira. and Caarls. (2021) is the first study to seek optimized techniques of ensemble deep reinforcement learning to decrease the hyperparameter tuning effort, where differently parameterized DDPG policies are trained online in a MuJoCo environment (Todorov et al, 2012). However, it was based purely on action aggregation, without taking into account the value function those actions are based on.

We believe the current work is the first use of value function ensembles specifically targeted at eliminating hyperparameter tuning. This article aims to improve the ensemble aggregation strategy by using the weighted mean of value functions trained using different, randomly chosen hyperparameters. This mean is used to choose an action from among corresponding control policies trained using those hyperparameters. In previous work, such a maximum Q-average ensemble (Hans and Udluft, 2010; Anschel et al, 2017) has been shown to work, although with unweighted averaging. Our main contribution resides in the use of a weighted average in order to deal with the larger expected variance between the outputs of vastly differently parameterized value functions. Additionally, our algorithm does not require parallel environments (Seyed Motehayeri et al, 2021), and as such can be applied on a single real-world system.

The Deep Deterministic Policy Gradient (DDPG) algorithm was chosen to validate the model, as it represents a family of state-of-the-art algorithms known for their good learning performance in continuous action settings (Shen et al, 2020). However, the approach is designed to be applicable to any off-policy actor-critic method.

This paper does not cover research that mixes RL with other ensemble optimization algorithms (Ganaie et al, 2022). In such settings, a single RL agent is used to choose an action among an ensemble of base models, such as for forecasting (Saadallah and Morik, 2021; Jalali et al, 2022; Lin et al, 2023). In contrast, our base models are RL agents themselves, a fact we use explicitly during training and inference.

This article is organized in 6 sections. Section 2 presents the DDPG algorithm and Q-Average ensemble method, while Section 3 presents the novel Online Weighted Q-Ensemble. The experiments are defined in Section 4 and their results discussed in Section 5. Finally, Section 6 presents the conclusion.

2 Background

Reinforcement learning is a type of machine learning technique that enables an agent to learn in an interactive environment by trial and error using feedback from its own actions and experience, seeking to maximize rewards (Sutton and Barto, 2018). Every time step t, an action $a \in \mathcal{A}$ at state $s \in \mathcal{S}$ is taken in the environment, which returns a reward $r \in \mathbb{R}$ and results in the next state $s' \in \mathcal{S}$. The goal is to find the control policy $\pi(a \mid s)$ that gives the probability of taking action a in state s that maximizes the expected sum of future rewards, also called the *return* R:

$$R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i+1}, \tag{1}$$

where γ is a *discount factor* introduced to avoid infinite returns.

In order to scale RL techniques, Deep RL learns its own state representation and can therefore solve complex problems, allowing its application in many domains of decision making tasks such as healthcare, robotics, smart grids and finance (François-Lavet et al, 2018).

2.1 Deep Deterministic Policy Gradient (DDPG)

DDPG (Lillicrap et al, 2016) is based on the Deterministic Policy Gradient (DPG), which was one of the first model-free and off-policy actor-critic algorithms for continuous state and action spaces (Silver et al, 2014). DDPG, illustrated in Figure 1, is an extension of DPG that uses deep neural networks to approximate the actor and critic. The actor approximates a deterministic policy $\mu(s;\ \theta)$ with weights θ, such that $\pi(a \mid s) = 1$ iff $a = \mu(s;\ \theta)$. The critic estimates the expected return of μ by approximating the action-value

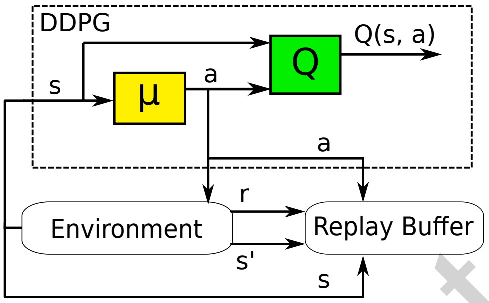

Fig. 1 Deep Deterministic Policy Gradient (DDPG) algorithm (Lillicrap et al, 2016), its interaction with the environment and the storage of transactions in the Replay Buffer.

function

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[R_t \mid s_t = s, a_t = a\right], \tag{2}$$

using a neural network, as in deep Q-learning (Mnih et al, 2014).

The critic network, with weights ζ, is updated to minimize the loss function

$$\begin{aligned} L(\zeta) &= \mathbb{E}_{(s,a,r,s')}\left[\left(Q(s,a;\ \zeta) - y\right)^2\right], \\ y &= r + \gamma Q(s', \mu(s';\ \theta');\ \zeta'), \end{aligned} \tag{3}$$

where θ' and ζ' are Polyak-averaged versions of the main network parameters, also called *target networks*, used to stabilize the learning. They are updated at given intervals using an averaging parameter τ:

$$\zeta' = (1-\tau)\cdot\zeta' + \tau\cdot\zeta, \tag{4}$$
$$\theta' = (1-\tau)\cdot\theta' + \tau\cdot\theta. \tag{5}$$

The actor update takes a step in the positive gradient criteria of the critic with respect to the actor parameters, given by the chain rule

$$\begin{aligned} \nabla_{\theta} J &= \mathbb{E}_{s_t\sim\rho}\left[\nabla_{\theta} Q(s,\mu(s;\ \theta);\ \zeta)|_{s=s_t}\right], \\ &= \mathbb{E}_{s_t\sim\rho}\left[\nabla_a Q(s,a;\ \zeta)|_{s=s_t,a=\mu(s_t)} \nabla_{\theta}\mu(s;\ \theta)|_{s=s_t})\right], \end{aligned} \tag{6}$$

thus moving the control policy in the direction of increased returns. In Eq. (6), J represents the expected return over the start distribution and ρ is an exploratory stochastic behavior policy.

An experience replay buffer $\mathcal{R}$ stores observed transitions (s,a,r,s'), in order to learn from past experience. Updates are performed using a random minibatch $\mathcal{D}$ sampled from $\mathcal{R}$, used to temporally decorrelate the observations.

The behavior policy ρ in Eq. (6) is derived from the actor by adding noise, $\rho \sim \mu(s;\ \theta) + \mathcal{N}$, to improve the exploration. The $\mathcal{N}$ uses the Ornstein-Uhlenbeck process (Uhlenbeck and Ornstein, 1930) for physical environments that have momentum to generate time-correlated exploration for increased efficiency ($\mathcal{N}_\theta = 0.15$ and $\mathcal{N}_\sigma = 1$). The Ornstein-Uhlenbeck process models the velocity of a Brownian particle with friction, which results in temporally correlated values centered around zero (Lillicrap et al, 2016).

2.2 Q-Average Aggregation

In Ensemble RL, multiple value functions and/or policies are learned at the same time, and their actions are aggregated to determine the ensemble action. One of the best ensemble aggregation methods developed in previous work is value function averaging. This has been successful in both regular reinforcement learning in discrete environments (Sun and Peterson, 1999; Ernst et al, 2005) as well as deep RL in continuous action spaces (Huang et al, 2017).

The Actor-Critic Ensemble (ACE; Huang et al (2017)) introduced the use of the Deep Deterministic Policy Gradient (DDPG) algorithm for ensembles. At inference time, the best action is selected from all actors running in parallel, each using the outputs of all critic networks which are combined by taking the average. This work showed significant improvement in the performance of DDPG in a bipedal walking environment; it increased the learning speed and lowered the number of falls.

3 Online Weighted Q-Ensemble

Our Q-ensemble model builds upon the Actor-Critic Ensemble method, by weighing the critics' predictions. Such a weighing, similar to conventional classifier Boosting (Freund and Schapire, 1996), aims to emphasize the input of the critics that better estimate the return when selecting the ensemble action. Considering that in our case the DDPG ensemble hyperparameters are chosen randomly with only little user input, it is important that critics with bad performance do not destabilize the final policy. Our critic weight update is therefore designed to decrease the weight of such critics.

3.1 Inference

Figure 2 presents the process by which Q values are calculated according to the Online Weighted Q-Ensemble model. The ensemble is composed of n DDPG agents, each composed of a critic $Q_i = Q(\cdot,\cdot;\ \zeta_i)$ and an actor $\mu_j = \mu(\cdot;\ \theta_j)$, $i, j \in 1 \ldots n$. At a given time step t and state s, each actor μ_j calculates an action $a_j = \mu(s_t;\ \theta_j)$. Subsequently, each of the actions is evaluated by all critics, generating the corresponding Q-values $Q(s_t, a_j;\ \zeta_i)$. The generation of these values results in a matrix $\mathbf{Q}$ with elements q_{ij}.

The DDPG networks are updated independently, since they use different hyperparameters. This creates a challenge when analyzing the $\mathbf{Q}$ matrix to select the ensemble action because their Q-value magnitudes may not be directly comparable. We propose the use of a softmax function σ to normalize the values of a critic for the different actions:

$$\sigma(q_{ij}) = \frac{e^{q_{ij}}}{\sum_{k=1}^{n} e^{q_{ik}}}. \tag{7}$$

The $\sigma(q_{ij})$ normalizes the q-values q_{ij} over all q-values related to value function Q_i, resulting in $\bar{q}_{ij}$. This procedure is equivalent to interpreting each critic as defining a softmax policy over the suggested actions. Averaging these action probabilities does not suffer from the incomparability of Q-values.

In the final step, the Q values are combined across critics using weighted averaging. The raw critic weights W_{raw} are normalized with the same softmax function σ to ensure they form a proper distribution, resulting in W. Then, we use the weighted average to calculate the critic ensemble prediction for all actions

$$q_{\text{w}} = W^T \mathbf{Q}, \tag{8}$$

where $W = \frac{1}{n}\mathbf{1}$ results in standard Q value averaging. Due to the normalization of the $\mathbf{Q}$ matrix, this procedure is equivalent to weighted Boltzmann addition (Wiering and Van Hasselt, 2008). The final ensemble action, a_t, is the one with the highest probability:

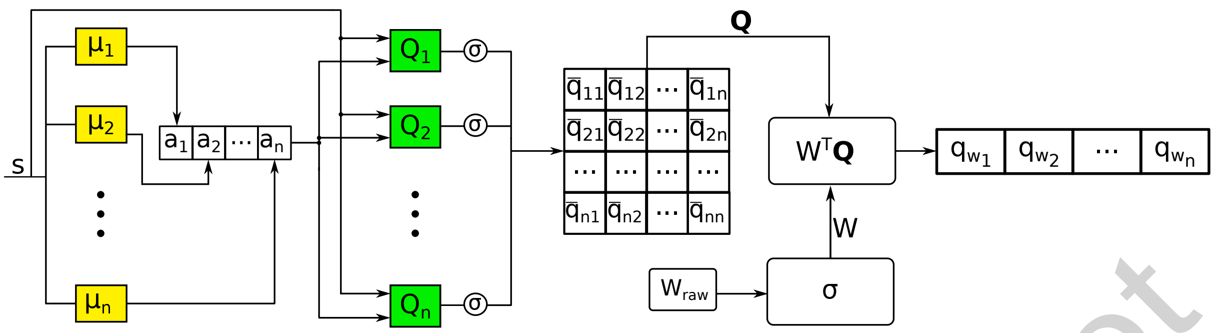

Fig. 2 Online Weighted Q-Ensemble. Each critic Q_i evaluates the actions suggested by all actors μ_j, and the resulting values are normalized using the softmax function σ. The final Q-value q_{w_j} of each action a_j is the sum of the critics' values $\{\bar{q}_{1j}, \bar{q}_{2j}, \cdots, \bar{q}_{nj}\}$, weighted by their respective weights w_i.

$$a_t = \mu_z(s_t), \tag{9}$$

where $z = \arg\max q_{\text{w}}$.

3.2 Online Training of Weights

All DDPG agents are trained in a single environment, using a shared replay buffer. The behavior policy υ^β is either derived from the ensemble action (online training) or from each actor in sequence on a per-episode basis (alternate training). The former can be expected to learn faster, while the latter ensures at least some near on-policy transitions for all actors, which may increase robustness.

The raw weights, W_{raw}, are initialized uniformly, and passed through a softmax layer before being used for the weighting. Therefore, at the beginning of the training, the critic weights W remain close to the uniform distribution. During training, we minimize the temporal difference (TD) error of the critic ensemble by minimizing the loss

$$\mathcal{L}(W_{\text{raw}}) = \sum_{(s,a,r,s')\in\mathcal{D}} \sum_{i=1}^{n} w_i \delta_i^2,$$
$$\delta_i = \quad r + \gamma Q(s', \mu(s';\ \theta_i');\ \zeta_i') - Q(s, a;\ \zeta_i), \tag{10}$$

over the weight parameters. Minimizing Eq. (10) reduces the weights of the critics with higher squared TD error δ, which can be assumed to have a worse value function prediction, see Eq. (3). Since $\sum_i w_i = 1$, due to the softmax function applied to W_{raw}, this necessarily increases the better critics' weights.

Note that although the Q values are normalized for action selection during inference, the TD errors calculated in Eq.(10) during training are not. As such, we do not optimize hyperparameters that inherently greatly influence the Q values, specifically the discount rate γ and reward scale.

The code used in the Online Weighted Q-Ensemble can be found online[1].

3.3 Performance Measure

We introduce a performance metric to compare the different forms of aggregation between different environments. It has the property of being invariant to both constant addition and multiplication of the reward function, which allows some measure of robustness in comparing environments that have performance values at different scales.

As such, to measure the overall performance of the aggregations used, the *average relative regret* is calculated as

$$\mathcal{R}(k) = \sum_e \sum_g \left| \frac{\max_l p_{egl} - p_{egk}}{\max_l p_{egl} - \min_l p_{egl}} \right|, \tag{11}$$

where p_{egk} is the performance of aggregation strategy k in environment e for ensemble group g (an ensemble group is a specific way of constructing the ensemble). This metric measures how much worse a certain aggregation strategy is, relative to the best aggregation for that experiment. Higher regrets mean worse overall performance.

[1] https://github.com/renata-garcia/wce_ddpg

4 Experiments

To validate the model, we test its performance by ablating the two differences with respect to Q-value averaging: using a weighted average, and using Bolzmann addition. This results in the following combinations:

- *Softmax TDError*: uses the model presented in Section 3;
- *TDError*: skips the softmax normalization of the Q-values presented in Eq. (7);
- *Softmax Average*: maintains $W = \frac{1}{n}\mathbf{1}$, but otherwise implements the model of Section 3;
- *Average*: standard Q-value averaging (Huang et al, 2017).

The average policy ensemble, with DDPG agents independently trained and no q-ensemble, recently presented good performance with 3 fine-tuned hyperparameter sets (Wu and Li, 2020) in a single environment 2D robot arm simulator. Based on this result, one ensemble group with 3 fine-tuned DDPG (*3 Good*) instances is used to validate the model.

However, the Online Weighted Q-Ensemble seeks to minimize the effort of fine-tuning in an ensemble, and to that end 3 more ensemble groups were created, mixing good (fine-tuned) and bad (not fine-tuned and non-converging) DDPG hyperparameters: *1 Good and 1 Bad*; *1 Good and 3 Bad*; and *1 Good and 7 Bad*.

In addition, two types of training mode are used in order to expand the validation of model. In the alternate training mode, at the beginning of each training episode, the policy is chosen alternately between each of the algorithms of the ensemble, as in Wu and Li (2020). In the online training mode and in the testing phase of the ensemble, the ensemble action is chosen at each step of the episode.

As environments, we chose two simple control problems, and two harder robotics tasks to evaluate scalability. In all cases, the episodes start at the resting point of the environment, the observations of the environments are in trigonometric format and there are 1000 observation steps before starting training. We use the standard feed-forward DDPG networks with two dense layers; the ranges of the hyperparameters are given in Table 1.

The specific environments used are the Inverted Pendulum Swing-up (2 state variables) and Cart-Pole environments (4 state variables) from the Generic Reinforcement Learning Library (GRL) [2], and half cheetah v2 (17 state variables) and swimmer v2 (8 state variables) from the OpenAI Gym framework (Brockman et al, 2016) with the MuJoCo environments (Todorov et al, 2012).

Swimmer v2 was used as a final validation for hyperparameter randomization. For this environment, 30 random configurations of ensembles formed with 8 parameterizations were trained once. The network architecture and the hyperparameters were randomly generated around the limits given before (Table 1).

In order to measure the overall performance of the aggregations used, the average relative regret in Eq. (11) is calculated over the first three environments and all four ensemble groups.

5 Results

In the this section, the average performances presented are calculated based on the cumulative rewards of the last 20 episodes in each run, and the 95% confidence interval is calculated over 30 runs of each configuration for the simple control problems and 10 runs for the half cheetah v2.

5.1 Performance

Figure 3 shows the final performance and its confidence interval. Each bar graph compares the performance on the 4 ensemble groups for the different aggregations, with separate graphs showing distinct environments and training modes. Also shown is the performance of the best single parameterization for each environment.

We can observe that the online training mode almost always outperforms alternate training; furthermore, there is no significant variation between the aggregations in the *3 Best* ensemble performance. Ensembles with a majority of bad parameterizations perform worse, which is especially evident in the more complex half cheetah v2 environment.

In general, the *Softmax TDError* aggregation performs better than, or within the confidence

[2]GRL Library (Generic Reinforcement Learning Library) (https://github.com/wcaarls/grl).

Table 1 Hyperparameters used to build each parameterization of the ensemble.

Hyperparameters	**Value**	**Description**
discount factor	0.99	Discount factor used in the Q-learning update.
reward scale	0.01	Scaling factor applied to the environment's rewards.
soft target update rate	0.01	Update rate of the target network weights.
update interval	10 or 100	Number of steps, or frequency with which the soft target update is applied.
learning rate (actor or critic network)	0.001 or 0.0001	Update rate used by AdamOptimizer.
replay steps[1]	64, 128 or 256	Total number of training samples per timestep.
minibatch size[1]	16, 64 or 128	Number of training samples per minibatch.
layer 1 size[2] (actor and critic network)	100, 200, 300 or 400	Number of neurons in regular densely-connected NN layers
layer 2 size[2] (actor and critic network)	50, 100, 200 or 300	Number of neurons in regular densely-connected NN layers
activation function	relu or softmax	Activation function of the Q Layer. (Critic Network)
replay memory size	1000000	Size of the replay memory array that stores the agent's experiences in the environment.
observation steps	1000	Observation period to start replay memory using random policy.

[1] Fixed per environment.
[2] Layer 2 size network was always smaller than layer 1 size network. Therefore, layer 1 size was never set with minimum value (50) and layer 2 size was never set with maximum value (400).

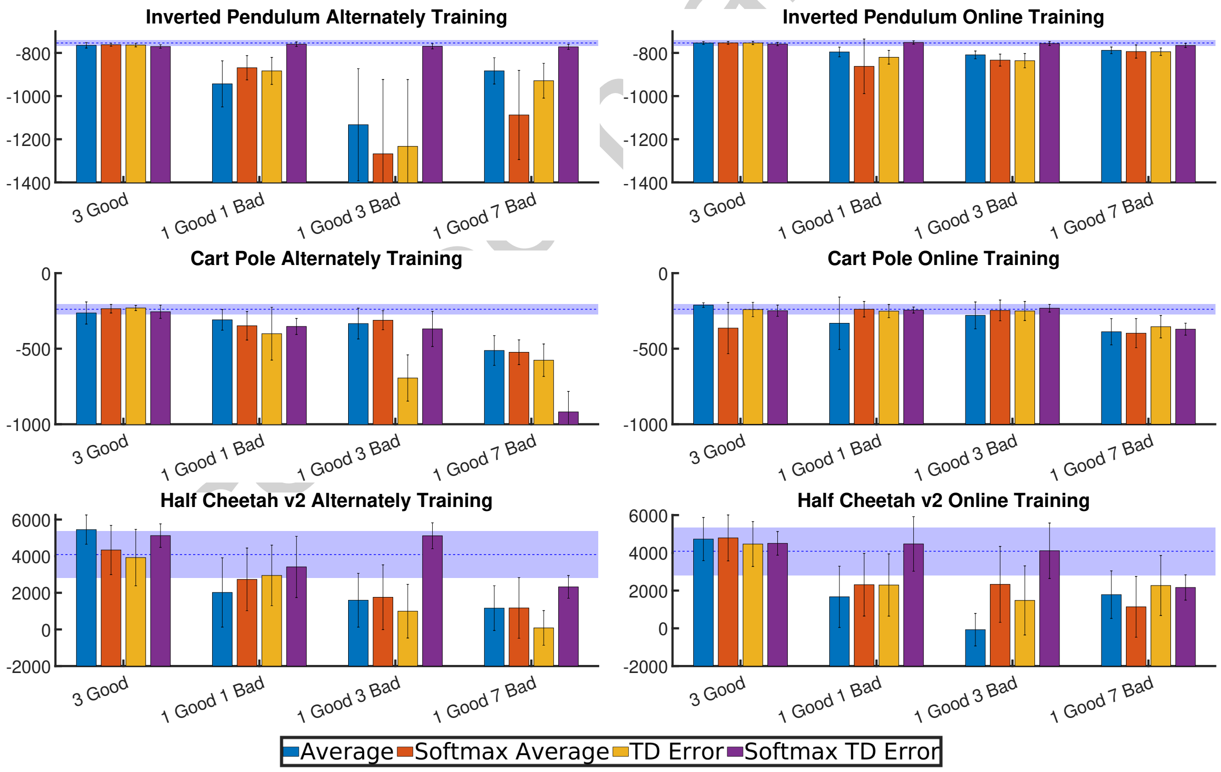

Fig. 3 Performance and 95% confidence interval of Online Weighted Q-Ensemble comparing groups (*3 Best, 1 Good 1 Bad, 1 Good 3 Bad and 1 Good 7 Bad*) and Q-Aggregation methods (*Average, Softmax Average, TD Error, Softmax TD Error*). Columns separate the Alternate and Online training and the rows present the environments: inverted pendulum, cart pole, and half cheetah v2. The horizontal line marks the mean of 30 (10 for half cheetah) single runs of the best parameterization, and the shadowed area represents its 95% confidence interval.

Table 2 Average relative regret using alternate and online training.

Training Mode	*Average*	*Softmax Average*	*TDError*	*Softmax TDError*
Alternate	4.2280	4.7868	5.5208	2.5675
Online	5.5059	6.0599	4.9232	2.3890

Table 3 One of 30 swimmer v2 random ensembles, with 8 randomly generated parameterizations.

LR Actor	LR Critic	Activa-tion	Layer 1 Size	Layer 2 Size	Inter-val
0.0009	0.0006	"softmax"	203	211	20
0.0006	0.0001	"softmax"	417	20	31
0.0001	0.0014	"softmax"	398	20	60
0.0014	0.0003	"relu"	202	310	69
0.0007	0.0011	"relu"	328	352	120
0.0013	0.0014	"softmax"	87	117	87
0.0004	0.0007	"softmax"	407	225	31
0.0001	0.0001	"relu"	326	32	56

interval of, the other aggregations in the *1 Good 3 Bad* group, and on par with the single and *3 Best* ensemble even in the half cheetah v2 environment. As such, even when there are a majority of bad parameterizations in the ensemble, we can expect our proposed algorithm to perform similar to a single finetuned solution.

Regarding the single ablations, there is no obvious trend as for which has the better performance. In fact, sometimes the intermediate strategies perform worse than simple Q-averaging. Clearly, both are required for optimum performance.

Table 2 presents the average relative regret of all strategies. Regardless of the training mode, the final model (*Softmax TDError*) has a better evaluation, standing out in relation to the other strategies. The latter rank differently depending on the training mode, making an impartial comparison between them impossible, thus it is not clear which aspect is more important for our model's final performance.

The swimmer v2 validation uses 30 different ensembles, each with 8 randomly generated parameterizations. See Table 3 for an example configuration. The performance of the full model (*Softmax TDError* aggregation) was compared with simple Q-averaging, using online training. The mean and confidence intervals are 85 ± 13 for averaging, and 110 ± 18 for our model, showing a significant improvement.

5.2 Learning curves

Figure 4 shows the online training mode learning curves of the *1 Good 1 Bad* and *1 Good 3 Bad* ensembles for the *Average* and *Softmax TDError* aggregations. In the inverted pendulum, *Softmax TDError* learns a bit faster than *Average*, while in the cart-pole this is behavior is reversed. In both environments learning is stable with good end performance, with *Softmax TDError* having higher mean and lower final variance.

In half cheetah v2 , the performance difference is huge. Both curves show high variance, but in both ensemble groups *Softmax TDError* performs better. Specifically, *Average* does not manage to learn in the *1 Good 3 Bad* ensemble, while *Softmax TDError* maintains the same performance as in the *1 Good 1 Bad* group.

5.3 Action preference and Q-weights

To better understand our model's behavior, an investigation of how the actions are chosen was made. Figure 5 presents the behavior of the proposed method (*Softmax TDError* with online training) on the half cheetah v2 MuJoCo environment, where the left column shows the weights assigned to each critic (**W**), and the right column shows how often the different actors' actions were chosen in the episode. All bad parameterizations have the same color, while the good ones are highlighted.

At the beginning of the learning process, **W** is uniformly distributed, and at the end the weights tend to choose the critic with the lowest calculated TD Error. The same process happens with the action counts; the beginning of the learning process has an equal distribution of the chosen actions, while at the end the choice of actions is influenced by the quality of the policy and the critics' acquired Q-Weights.

The first row shows the *3 Good* group, where each parameterization has a different color. Intuitively, the expectation is that both weights and action choices should stay equally distributed, since all parameterizations have roughly the same individual performance. Initially, this is not the case, as some choices may learn faster than others, but, at least for the actions, the end result is as expected. Note that one critic has a very

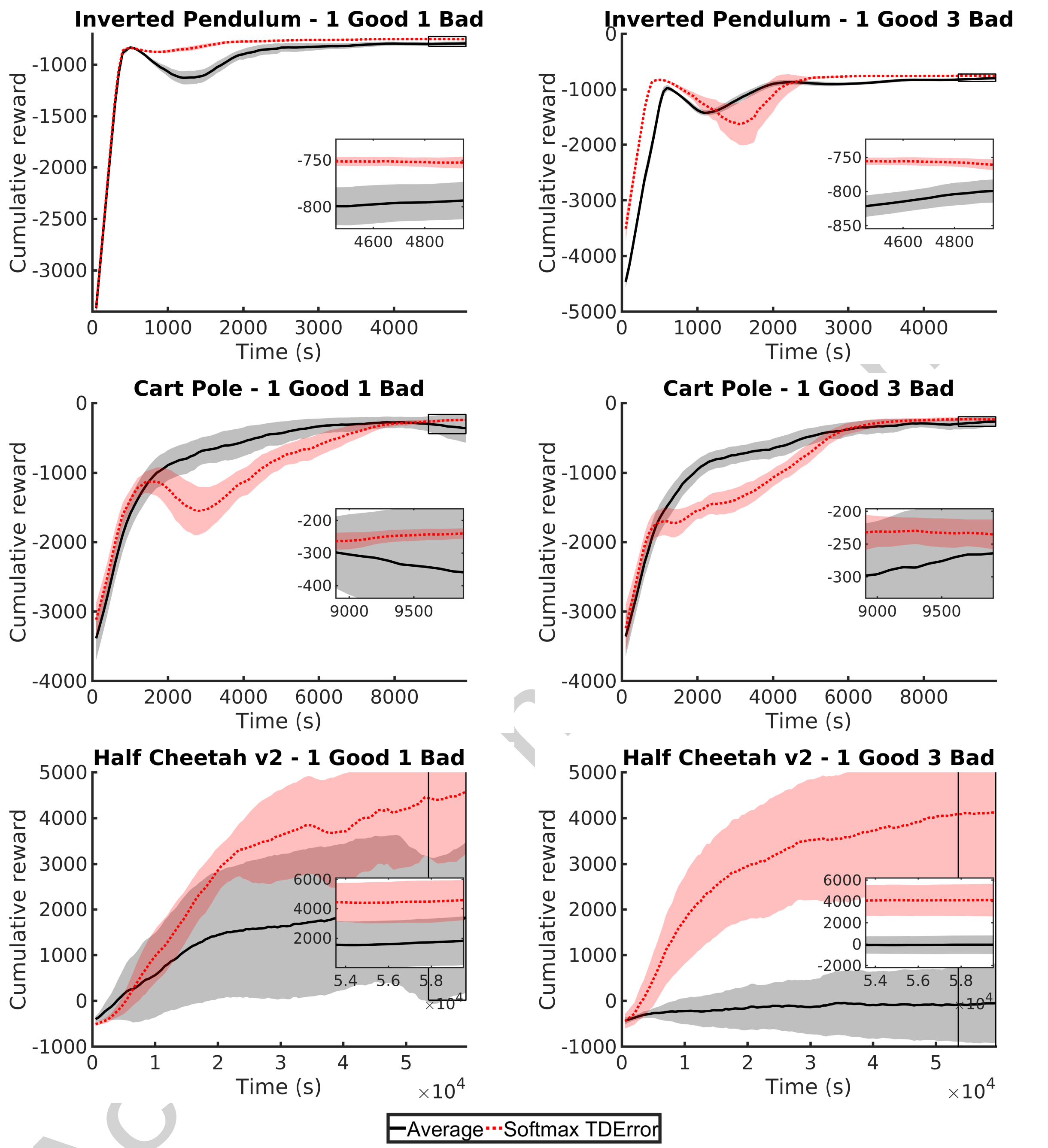

Fig. 4 Learning Curve of Online Weighted Q-Ensemble with *1 Good 1 Bad, and 1 Good and 3 Bad* and Q-Aggregation (*Average, Softmax TD Error*). All cases are online training, lines present the environments: inverted pendulum, cart pole and half cheetah v2. The graphic also shows the 95% confidence interval.

low weight, but its actor's action is chosen normally (red line). This shows that having a higher TD-error critic does not always imply a worse actor.

The second row presents the *1 Good 1 Bad* group, which is composed of two opposing parameterizations, with an expectation that the good parameterization will be chosen from the beginning. Indeed, there is a very clear distinction from the Q-Weights, that is reflected in the choice of actions with little noise. The third row of Figure 5 presents the *1 Good and 3 Bad* group, which behaves similarly to the *1 Good 1 Bad*, but with more challenges, as there are more bad parameterizations to compete with.

Finally, the *1 Good and 7 Bad* group shown in the last row struggles to find the good agent. The Q-Weights do not manage to converge to the best individual critic, nor is its action chosen more often than the others. However, the results in Figure 3 show that the ensemble still reaches an adequate (although not optimal) performance.

Overall, the best individual agent has both lower weight and its actions are generally chosen less in larger ensembles. Even so, there is an improvement in the final performance when *Softmax TDError* is used.

6 Conclusion

This article proposed the Online Weighted Q-Ensemble to decrease the hyperparameter tuning effort for deep reinforcement learning in continuous action spaces. Based on previous work which uses an average of Q-ensembles in an actor-critic setting, we introduced a weighing approach that adjusts the critics' weights by minimizing the temporal difference error of the ensemble as a whole. Additionally, instead of combining the Q-values directly, they were applied through a softmax layer, in order to focus on relative preferences rather than absolute values.

In both simple and complex robotic simulation environments, our model showed better results than the standard Q-value averaging, and managed to maintain performance comparable to the best individual run even if the ensemble included up to 3-7 bad parameterizations. Validation using ensembles with 8 randomized parameterizations also showed a 30% performance increase compared to normal q-value averaging.

Future Work

Our tests used a single environment, as they were aimed at the system's applicability in real-world robotic applications. In future work, it would be interesting to extend the simulations to more environments, and validate its use in real robots. Other interesting points to be further expanded in possible subsequent works are the acceleration of learning and the extension of tests with further algorithms, such as TD3 and SAC, or even the combination of completely different algorithms (Ali and Öztürk, 2023). Furthermore, while performance with mostly bad parameterizations was adequate, algorithmic improvements could be made to further suppress their influence, increasing robustness and decreasing the need to select good hyperparameter ranges. Finally, in our experiments the discount rate and reward scale were kept fixed, and therefore still require manual tuning. Extensions to include variations of these hyperparmeters in the ensemble could be considered, as well as the target update interval and replay memory size.

Declarations

Funding

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance code 001 and the National Council for Scientific and Technological Development – CNPq under project number 314121/2021-8.

Competing interests

The authors have no relevant financial or non-financial interests to disclose.

Ethics approval

Not applicable.

Consent to participate

Not applicable.

Consent to publish

Not applicable.

Data availability

The code used to generate the results in this paper is available at https://github.com/renata-garcia/wce_ddpg.

Authors' contributions

All authors contributed to the study conception, design and programming. Renata Garcia performed the experiments and wrote the first draft of the manuscript. All authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

References

Ali MH, Öztürk S (2023) Efficient congestion control in communications using novel weighted ensemble deep reinforcement learning. Computers and Electrical Engineering 110:108,811. https://doi.org/https://doi.org/10.1016/j.compeleceng.2023.108811, URL https://www.sciencedirect.com/science/article/pii/S0045790623002355

Anschel O, Baram N, Shimkin N (2017) Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning. In: Proceedings of the 34th International Conference on Machine Learning - Volume 70. JMLR.org, Sydney, NSW, Australia, ICML'17, p 176–185

Arulkumaran K, Deisenroth MP, Brundage M, et al (2017) Deep reinforcement learning: A brief survey. IEEE Signal Processing Magazine 34(6):26–38. https://doi.org/10.1109/MSP.2017.2743240

Brockman G, Cheung V, Pettersson L, et al (2016) Openai gym. CoRR https://arxiv.org/abs/arXiv:1606.01540

Cardeñoso Fernandez F, Caarls W (2018) Parameters tuning and optimization for reinforcement learning algorithms using evolutionary computing. In: 2018 International Conference on Information Systems and Computer Science (INCISCOS). IEEE, Quito, Equador, pp 301–305, https://doi.org/10.1109/INCISCOS.2018.00050

Duell S, Udluft S (2013) Ensembles for continuous actions in reinforcement learning. In: ESANN 2013 proceedings, pp 24–26

Dulac-Arnold G, Levine N, Mankowitz DJ, et al (2021) Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. Machine Learning 110:2419–2468. https://doi.org/10.1007/s10994-021-05961-4

Ernst D, Geurts P, Wehenkel L (2005) Tree-based batch mode reinforcement learning. Journal of Machine Learning Research 6:503–556

François-Lavet V, Henderson P, Islam R, et al (2018) An introduction to deep reinforcement learning. Foundations and Trends® in Machine Learning 11(3-4):219–354. https://doi.org/10.1561/2200000071

Freund Y, Schapire RE (1996) Experiments with a new boosting algorithm. In: Proceedings of the Thirteenth International Conference on International Conference on Machine Learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, ICML'96, p 148–156

Fujimoto S, van Hoof H, Meger D (2018) Addressing function approximation error in actor-critic methods. In: Dy J, Krause A (eds) Proceedings of the 35th International Conference on Machine Learning, Proceedings of Machine Learning Research, vol 80. PMLR, Stockholmsmässan, Stockholm Sweden, pp 1587–1596, URL http://proceedings.mlr.press/v80/fujimoto18a.html

Ganaie M, Hu M, Malik A, et al (2022) Ensemble deep learning: A review. Engineering Applications of Artificial Intelligence 115:105,151. https://doi.org/https://doi.org/10.1016/j.engappai.2022.105151, URL https://www.sciencedirect.com/science/article/pii/S095219762200269X

Haarnoja T, Zhou A, Abbeel P, et al (2018) Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. CoRR abs/1801.01290. URL http://arxiv.org/abs/1801.01290, https://arxiv.org/abs/1801.01290

Han D, Mulyana B, Stankovic V, et al (2023) A survey on deep reinforcement learning algorithms for robotic manipulation. Sensors 23(7). https://doi.org/10.3390/s23073762, URL https://www.mdpi.com/1424-8220/23/7/3762

Hans A, Udluft S (2010) Ensembles of neural networks for robust reinforcement learning. In: 2010 Ninth International Conference on Machine Learning and Applications, pp 401–406, https://doi.org/10.1109/ICMLA.2010.66

Huang Z, Zhou S, Zhuang B, et al (2017) Learning to run with actor-critic ensemble. arXiv preprint arXiv:171208987

Jalali SMJ, Osório GJ, Ahmadian S, et al (2022) New hybrid deep neural architectural search-based ensemble reinforcement learning strategy for wind power forecasting. IEEE Transactions on Industry Applications 58(1):15–27. https://doi.org/10.1109/TIA.2021.3126272

Jung W, Park G, Sung Y (2020) Population-guided parallel policy search for reinforcement learning. arXiv preprint arXiv:200102907

Khalilpourazari S, Doulabi HH (2021) Designing a hybrid reinforcement learning based algorithm with application in prediction of the covid-19 pandemic in quebec. Annals of Operations Research pp 1–45

Koryakovskiy I, Vallery H, Babuška R, et al (2017) Evaluation of physical damage associated with action selection strategies in reinforcement learning**i. koryakovskiy, h. vallery and r.babuška were supported by the european project koroibot fp7-ict-2013-10/611909. IFAC-PapersOnLine 50(1):6928–6933. https://doi.org/https://doi.org/10.1016/j.ifacol.2017.08.1218, URL https://www.sciencedirect.com/science/article/pii/S240589631731724X, 20th IFAC World Congress

Lee K, Laskin M, Srinivas A, et al (2021) Sunrise: A simple unified framework for ensemble learning in deep reinforcement learning. In: Proceedings of the 38th International Conference on Machine Learning, pp 6131–6141, URL https://proceedings.mlr.press/v139/lee21g.html

Lillicrap TP, Hunt JJ, Pritzel A, et al (2016) Continuous control with deep reinforcement learning. In: Proceedings of International Conference on Learning Representations, San Juan, Puerto Rico

Lin W, Xie L, Xu H (2023) Deep-reinforcement-learning-based dynamic ensemble model for stock prediction. Electronics 12(21). https://doi.org/10.3390/electronics12214483, URL https://www.mdpi.com/2079-9292/12/21/4483

Liu R, Nageotte F, Zanne P, et al (2021) Deep reinforcement learning for the control of robotic manipulation: a focussed mini-review. Robotics 10(1):22. https://doi.org/https://doi.org/10.3390/robotics10010022

Meijdam HJ, Plooij MC, Caarls W (2013) Learning while preventing mechanical failure due to random motions. In: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp 182–187, https://doi.org/10.1109/IROS.2013.6696351

Mnih V, Heess N, Graves A, et al (2014) Recurrent models of visual attention. In: Advances in neural information processing systems

Mnih V, Kavukcuoglu K, Silver D, et al (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529–533. https://doi.org/http://doi.org/10.1038/nature14236

Oliveira. RG, Caarls. W (2020) Comparing action aggregation strategies in deep reinforcement learning with continuous action. In: Anais do XXIII Congresso Brasileiro de Automática - Volume 2 No 1: CBA 2020,, https://doi.org/10.48011/asba.v2i1.1547

Oliveira. RG, Caarls. W (2021) A history-based framework for online continuous action ensembles in deep reinforcement learning. In: Proceedings of the 13th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART,, pp 580–588, https://doi.org/10.5220/0010199005800588

Saadallah A, Morik K (2021) Online ensemble aggregation using deep reinforcement learning for time series forecasting. In: 2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA), pp 1–8, https://doi.org/10.1109/DSAA53316.2021.9564132

Seyed Motehayeri SM, Baghi V, Miandoab EM, et al (2021) Duplicated replay buffer for asynchronous deep deterministic policy gradient. In: 2021 26th International Computer Conference, Computer Society of Iran (CSICC), pp 1–6, https://doi.org/10.1109/CSICC52343.2021.9420550

Shen Q, Li Y, Jiang H, et al (2020) Deep reinforcement learning with robust and smooth policy. In: III HD, Singh A (eds) Proceedings of the 37th International Conference on Machine Learning (PMLR), pp 8707–8718, URL http://proceedings.mlr.press/v119/shen20b.html

Silver D, Lever G, Heess N, et al (2014) Deterministic policy gradient algorithms. In: Proceedings of the 31st International Conference on International Conference on Machine Learning, vol 32. JMLR.org, Bejing, China, pp 387–395

Song Y, Suganthan PN, Pedrycz W, et al (2023) Ensemble reinforcement learning: A survey. Applied Soft Computing 149:110,975. https://doi.org/https://doi.org/10.1016/j.asoc.2023.110975, URL https://www.sciencedirect.com/science/article/pii/S1568494623009936

Sun R, Peterson T (1999) Multi-agent reinforcement learning: weighting and partitioning. Neural networks 12(4-5):727–753

Sutton RS, Barto AG (2018) Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA

Tammewar A, Chaudhari N, Saini B, et al (2023) Improving the performance of autonomous driving through deep reinforcement learning. Sustainability 15(18). https://doi.org/10.3390/su151813799, URL https://www.mdpi.com/2071-1050/15/18/13799

Todorov E, Erez T, Tassa Y (2012) Mujoco: A physics engine for model-based control. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp 5026–5033, https://doi.org/10.1109/IROS.2012.6386109

Uhlenbeck GE, Ornstein LS (1930) On the theory of the brownian motion. Physical review 36(5):823

Watts J, Khojandi A, Vasudevan R, et al (2020) Optimizing individualized treatment planning for parkinson's disease using deep reinforcement learning. In: 2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC), pp 5406–5409, https://doi.org/10.1109/EMBC44109.2020.9175311

Wiering MA, Van Hasselt H (2008) Ensemble algorithms in reinforcement learning. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 38(4):930–936

Wu J, Li H (2020) Deep ensemble reinforcement learning with multiple deep deterministic policy gradient algorithm. Mathematical Problems in Engineering 6:1–12

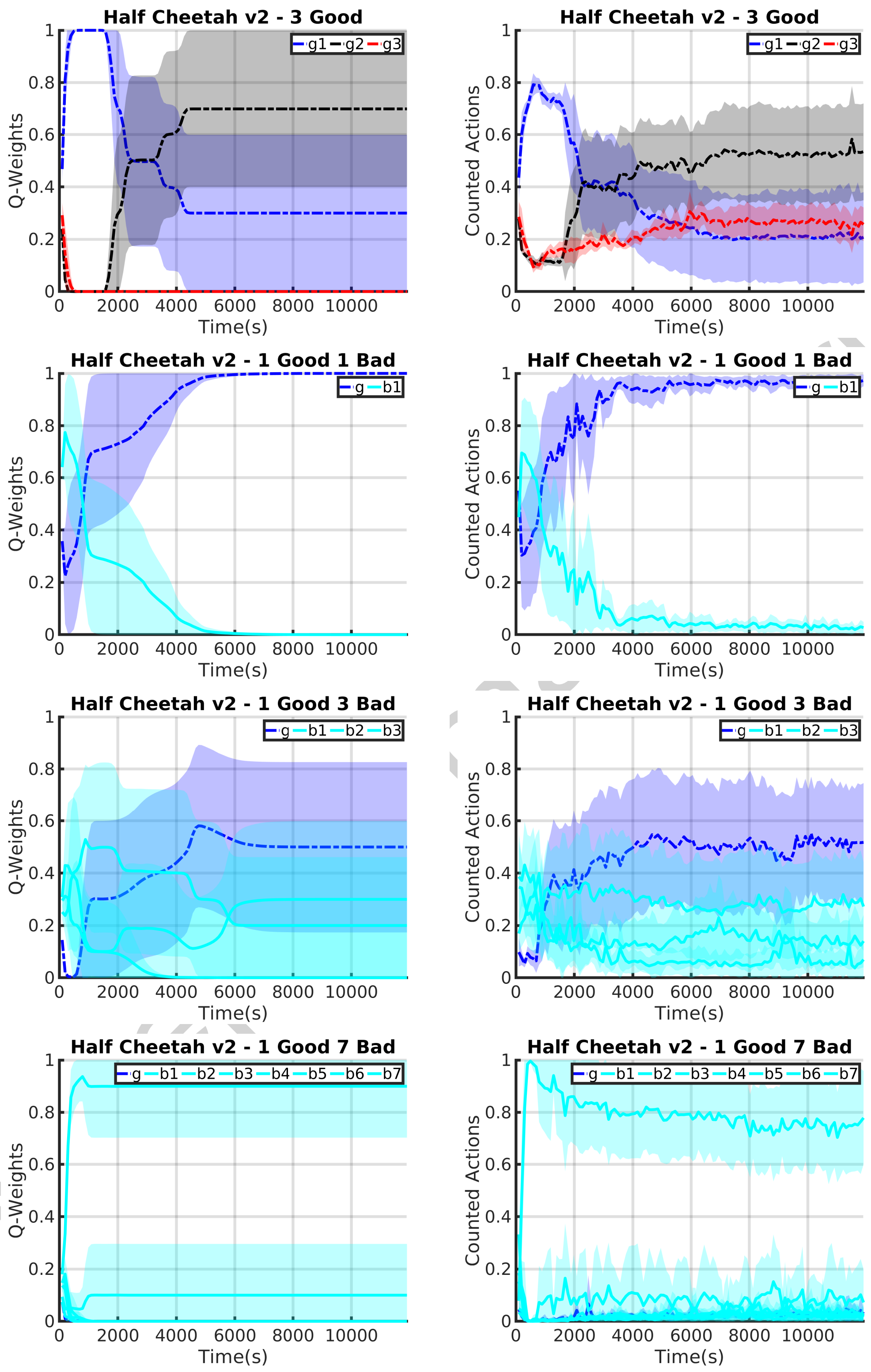

Fig. 5 Action count and Q-Weights of Online Weighted Q-Ensemble with *3 Best, 1 Good 1 Bad, 1 Good and 3 Bad, and 1 Good and 7 Bad* and Q-Aggregation *Softmax TD Error*. All cases are online training in half cheetah v2 environment. The graphic also shows the 95% confidence interval.