DOI: 10.48011/asba.v2i1.1547

Comparing Action Aggregation Strategies in Deep Reinforcement Learning with Continuous Action

Renata Garcia Oliveira * **Wouter Caarls** **

* *Pontifical Catholic University of Rio de Janeiro, RJ, (e-mail: renatargo2@aluno.puc-rio.br; renata.garcia.eng@gmail.com).*
** *Pontifical Catholic University of Rio de Janeiro, RJ (e-mail: wouter@ele.puc-rio.br)*

Abstract: Deep Reinforcement Learning has been very promising in learning continuous control policies. For complex tasks, Reinforcement Learning with minimal human intervention is still a challenge. This article proposes a study to improve performance and to stabilize the learning curve using the ensemble learning methods. Learning a combined parameterized action function using multiple agents in a single environment, while searching for a better way to learn, regardless of the quality of the parametrization. The action ensemble methods were applied in three environments: pendulum swing-up, cart pole and half cheetah. Their results demonstrated that action ensemble can improve performance with respect to the grid search technique. This article also presents as contribution the comparison of the effectiveness of the aggregation techniques, the analysis considers the use of the separate or the combined policies during training. The latter presents better learning results when used with the data center aggregation strategy.

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

1. INTRODUCTION

Reinforcement Learning (RL) is a mathematical framework that trains the control policy of an intelligent agent by interacting with world learning through trial and error. The agents seek to learn with minimal human intervention, autonomous, with no knowledge of robot or environment, which is the way to scalable reinforcement learning systems (Gu et al., 2017; Ha et al., 2020). This motivated research on the effectiveness of reinforcement learning and its combination with neural network and deep learning, besides pursuing the direct use in real robots (Popov et al., 2017).

The efficiency of RL in complex real-world scenarios depends on the efficient representation of the environment from high-dimensional observation inputs. In addition, learning an efficient policy (a mapping of observations to actions) for new events depends on the past experiences generalization. Efficient learning in the real world demands that the algorithm be stable and efficient. This is a challenge that addresses the security and automation of the process, which requires continuous data collection.

Deep reinforcement learning has been used to solve many problems of continuous control. The Deep Q network (DQN) (Mnih et al., 2013) learns policies using end-to-end reinforcement learning receiving as inputs direct visual information (pixels), using high-dimensional continuous state space and discrete actions. DQN is the first artificial agent capable of learning a diverse array of challenging tasks (Mnih et al., 2015).

Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2016) is an actor-critic algorithm which learns Q-function (reward-to-go) and control policy. This algorithm and its variants (e.g. TD3 (Fujimoto et al., 2018)) are in the state of the art in Deep Continuous Reinforcement Learning. It is a model-free, supports high-dimensional states space and continuous actions space. It performs very well, even though it presents a great need to fine-tune the hyperparameters involved. Another challenge is the difficult to maintain stability of learning.

The use of continuous actions ensemble, which is the object of study in this article, aims at improving performance and reducing learning variance, conducting a comprehensive study among the action aggregation techniques presented in the literature for RL (Wu and Li, 2020; Anschel et al., 2017; Duell and Udluft, 2013; Faußer and Schwenker, 2011), furthermore, this article investigates an analysis of the quality of the hyperparameters that participate in the learning process. This article presents analysis and experiments of ensemble methods from different instances of DDPG.

Studies of policy ensembles started with majority vote decision (Wiering and Van Hasselt, 2008) and average decision (Faußer and Schwenker, 2011). As for Deep RL, there are some studies using DQN (Anschel et al., 2017) and DDPG (Wu and Li, 2020) algorithms. Besides these, Density Based (based Majority Voting) and Data Center strategies were used in neural networks combined with RL (Duell and Udluft, 2013) and they presented promising

results. All these strategies will be explained later, as well their performance.

A recent study in DDPG, Bootstrapped aggregated multi-DDPG (BAMDDPG) (Wu and Li, 2020), is structured with Multiple DDPG networks for optimizing controllers (policies) in continuous actions space. It uses three DDPG networks each one with a different simulation environment. The limitation of this approach is the difficulty in carrying out training in the real world, which is a single environment, with minimal intervention.

The main contribution of this work is the comparison between the different action aggregation strategies used in DDPG ensembles. This work is organized in 6 sections. Section 2 introduces the background. An explanation of the approach in ensemble DDPG and presentation of experiments can be found in Sections 3 and 4. Section 5 presents the results and discussion and Section 6 is the conclusion.

2. BACKGROUND

This section presents in more detail the Deep Deterministic Policy Gradient (DDPG) and action ensemble aggregation strategies.

2.1 Deep Deterministic Policy Gradient (DDPG)

Reinforcement Learning (RL) is a Machine Learning area used for optimizing controllers in complex technical system. RL learns by interacting with the environment. Given a state s, an action a is performed in the environment, which return a reward r and the next state s'.

Actions selection is modeled as a map called policy $\pi(s)$, the choice depends on the current state. The policy is trained to maximize the expected return, for example, from the start state distribution $E : J = \mathbb{E}_{r_i,s_i\sim E,a_i\sim\pi}[R]$.

The challenge in solving real world tasks with reinforcement learning is finding the optimal policy π^* using a continuous action space and high-dimensional states space. The RL's goal is to learn π^* that maximizes the expected sum of discounted future rewards Q^* (1). The discounting factor $\gamma \in [0, 1)$ guarantees finite sums in infinite-horizon tasks:

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\right], \quad (1)$$

where t is a time step.

DDPG is a type of Deep Learning (Silver et al., 2014; Lillicrap et al., 2016), it is an actor-critic algorithm based on the DPG algorithm. The implementation uses a target network, to stabilize the learning, and the replay buffer, which not only breaks the correlation of the acquired data but also avoids forgetting already learned behavior.

The target network θ' is a copy of main network θ and its output is used as a target value to measure the mean square error. The network weights θ' are updated (time delayed) according to the soft target update rate τ (2). In addition, the soft update is performed every number of steps (interval parameter).

$$\theta'^{(t)} = \tau\theta^{(t-1)} + (1-\tau)\theta'^{(t-1)} \text{ with } \tau \in [0,1] \quad (2)$$

The DDPG critic $Q(s, a)$ is optimized by minimizing the loss (3) of the mean square error of the main and target network critics. The current policy is parameterized by the function $\mu(s)$ and the Q function approximation is parameterized by θ^Q.

$$L(\theta^Q) = \mathbb{E}_{(s,a,r,s')}\left[\left(y_t - Q(s, \mu(s)|\theta^Q)\right)^2\right] \quad (3)$$

where

$$y_t = r_t + \gamma Q(s', \mu(s')|\theta^{Q'}) \quad (4)$$

This actor function $\mu(s|\theta^\mu)$ deterministically maps states to a specific action. The actor update occurs in the direction of the positive gradient criteria of the critic with respect to the actor parameters (5). ρ^β is the distribution of the (stochastic) behavior policy ρ_π and J represents the expected return $\mathbb{E}$ from the start distribution.

$$\begin{aligned}&\nabla_{\theta_\mu} J = \\ &\mathbb{E}_{s_t\sim\rho^\beta}\left[\nabla_a Q(s,a|\theta^Q)|_{s=s_t,a=\mu(s_t)} \nabla_{\theta_\mu}\mu(s|\theta^\mu|_{s=s_t})\right]\end{aligned} \quad (5)$$

As this is an off-policy algorithm, the learning environment does not have to follow the policy, so experiences (s, a, r, s') are stored in a replay buffer. Samples of the buffer, known as minibatches, are selected and used to train in order to avoid forgetting throughout learning, and they are randomly sampled to ensure no correlation between the data, avoiding over or under fitting.

Model Architecture The network has two hidden layers with 400 and 300 units respectively. The neural network learns using AdamOptimizer (Kingma and Ba, 2014), the actor and the critic learning rate are 10^{-4} and 10^{-3} respectively. The discount factor is $\gamma = 0.99$, reward scale is $rs = 0.1$ and the soft target updates used is $\tau = 0.01$. The exploration noise uses an Ornstein-Uhlenbeck process (Uhlenbeck and Ornstein, 1930), which is specific to explore physical environments that have momentum. The parameterized values are $\theta_{\mathbb{N}} = 0.15$ and $\sigma_{\mathbb{N}} = 1.0$ for all simulations discussed in this work. The importance of this technique is to avoid temporally correlate the exploration, because pure white noise would not lead to significant deviation. From these DDPG initial values, some modifications were made in order to improve the performance in each environment, this will be detailed in Section 3 below. The input of the neural network consists of the number of observations from each environment, this is described in the environment details in Section 4.1.

2.2 Ensemble Aggregation Strategies

This section presents a recent article that uses the mean for action aggregation (Wu and Li, 2020) and details others aggregations proposed for continuous actions (Duell and Udluft, 2013).

BAMDDPG A recent study in DDPG, Bootstrapped aggregated multi-DDPG (BAMDDPG) (Wu and Li, 2020), uses DDPG instances for optimizing controllers (policies)

in continuous actions space. In particular, it uses three DDPGs using a single or multiple environments. When using a single environment, at each episode an alternate different DDPG is chosen to interact with the environment, for multiple environment interaction occurs independently at each step for all environments. Average actions ensemble were evaluated in the reacher and in the car racing simulation in multiple environments. After experiments, three algorithms composing the ensemble demonstrated to be more effective. In all cases, the replay buffer (sample experiences) is shared between the DDPG instances.

Below, aggregation approaches for continuous learning policies will be listed.

Mean Aggregation It is simple arithmetic mean of the actions proposed by different algorithms.

Data Center Aggregation The algorithm is based on the idea of using the data group weighted average. Instead of the mean aggregation, the action that is the farthest of the group average is removed in an iterative process, until only two actions remain and its average is used. Algorithm 1 presents the data center pseudo code that uses Euclidean distance as measure. Note that each a_i can be a vector of multiple action dimensions.

Algorithm 1. Datacenter Pseudocode.

$\mathcal{A} = (a_1 + a_2 + \cdots + a_n)$
while $|\mathcal{A}| > 2$ **do**
  $\mu = (\sum a_i)/|\mathcal{A}|$
  for *each element* a_i *in* $|\mathcal{A}|$ **do**
    $d_i = ||a_i - \mu||$
  end
  $\mathrm{j} = \arg\max_i d_i$
  $\mathcal{A} \leftarrow \mathcal{A} \setminus \{a_j\}$
end
$data\_center = (a_1 + a_2)/2$

Density Based Aggregation Created as an extension of majority voting to continuous actions space. The density of each action d_i is a sum of Gaussian functions as showed in Figure 1. The final chosen action has the highest density in the actions space.

Equation (6) calculates the density action, the exponent of e is the sum of the difference between the N others actions of the ensemble and the actual action a_i, considering k dimension (k actions required by the environment in each steps). Considering an environment with more than 1 action, the actions space probably will not have same scale and because that they are normalized for the interval $[-1, 1]$. The distance parameter $r = 0.025$ was chosen empirically. This is therefore not a parameter-free algorithm.

$$d_i = \sum_{j=1}^{N} e^{\frac{-\sum_{l=1}^{k}\left(a_{il}-a_{jl}\right)^2}{r^2}} \tag{6}$$

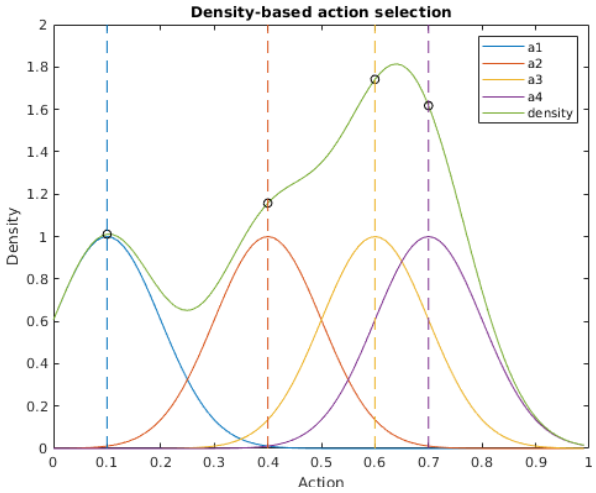

Figure 1. Illustration of density action.

3. ENSEMBLES IN DDPG

Despite the BAMDDPG (Section 2.2) considering the alternately policy selection during training when using single environment, it is not clear the performance behavior, once the tests presented uses multiple environment. So this section presents the policy selection during training and testing group to benchmark the ensemble methods performance.

3.1 Policy Selection in Training Phase

The Alternately Persistent and Online Training are the policy selection used during the training phase of algorithm. The mode of policy selection using DDPG ensemble can lead to a better convergence of the learning curve. Considering the use of a single environment, in Reinforcement Learning, it is important to pursue its self-adaptation feature of the algorithm during learning the environment.

Alternately Persistent The selected policy is alternated every episode between the DDPG ensemble in training. In the testing phase, at each step, the action aggregation technique is applied.

Online Training In ever training episode, at each step, the action aggregation technique is applied. The idea is taking advantage from a time-related learning actions, once all DDPG are learning together.

3.2 Testing group to benchmark

Hyperparameters influences the speed of learning and the ability to generalize the algorithm, they are variables of network structure and related to the trained network. The value possibilities ranges used to compose the trained DDPG are shown in the Table 1.

Each environment was extensively executed to find the best performance. For this, a narrow grid search for each hyperparameter value was performed, in which when finding high performance for one variable, it is frozen and a grid variation is performed on another variable, until the process ends. This is quite computationally costly, yet it

Table 1. Table with the hyperparameters' description used to build each algorithm that is part of the ensemble.

Hyperparameters	**Value**	**Description**
discount factor	0.98 or 0.99	Discount factor used in the Q-learning update.
reward scale	0.001, 0.1 or 1	Scaling factor applied to the rewards received from the environment.
soft target update rate	0.01	Update rate of the target network weights.
target network update interval	10 or 100	Steps number, or frequency, with which the target network parameters are updated.
learning rate	0.001 and 0.0001	Update rate used by AdamOptimizer.
replay steps	32 to 512	Number of minibatches used in a single step.
minibatch size	32 to 512	Number of transitions from the environment used in batch to update the network.
layer size	50 to 400	Number of neurons in regular densely-connected NN layers
activation layer	relu or softmax	Output function of the final network layer.
replay memory size	1000000	Size of the replay memory array that stores the agent's experiences in the environment.
observation steps	1000	Observation period in which the policy is chosen at random and the result of the experiment is used to initialize the replay memory.

is important to find the best single DDPG baseline for comparisons.

Below are presented three groups designed to evaluate the performance behavior information when the DDPG hyperparameters of the ensemble are not fine tuning specifically to the environment. Adding the best DDPG hyperparameters performance for each environment is used in analysis. It is important to note that the best fine tuning for one environment may not be the best for another. Table 2 summarizes the composition of formed groups.

Table 2. Groups of algorithm parameterizations.

parameterizations	***3 best***	***mostly good***	***mostly bad***
best	3	12	4
worst	0	4	12

3 Best The three best configurations of the DDPG hyperparameters of the environment.

Mostly good Ensemble is composed by 16 DDPGs, the idea is to gather 12 good parameterizations of the algorithm, that is 75% of the group and reserve 25%, that is, 4 bad parameterizations, that are not well configured and result in a non-convergence of learning. The idea is to verify the behavior of the ensemble's learning strategy when there are not always good parameterizations, and thus, to verify the learning resilience.

Mostly bad Similar to the composition previously presented. So 75% is composed by bad parameterizations and 25% of good parameterizations. As most are composed of bad parameterized algorithms, which do not converge, it becomes an immense challenge to carry out learning based on those good algorithms.

4. EXPERIMENTS

The experiments with DDPG were performed using the action aggregation techniques, training mode and testing group as benchmark in the environments previously presented.

4.1 Environments

The environments are Inverted Pendulum Swing-up, Cart Pole and Half Cheetah v2. Generic Reinforcement Learning Library (GRL) [1] provides the first two environments. The last one is used from the OpenAI Gym framework (Brockman et al., 2016) with the MuJoCo environments (Todorov et al., 2012). The training environments consider random initialization so as not to bias the learning.

Inverted Pendulum Swing-up (PD) It is a free pole attached only by an axis that, if there is no external force, remains down immobile, as shown in Figure 2. The goal is to learn the necessary torque to swing the pole, rotate it on the axis and maintain balanced on top (Busoniu et al., 2017). The observation variable considers derived information of the angle, sine and cosine, in order to avoid the $[-\pi, \pi]$ wrapping problem, more details of training and variables characteristics can be found in Table 3. The reward function is represented by (7), where x is the observation variables, pendulum position α and pendulum velocity $\dot{\alpha}$, and u is the control action.

$$\rho(x, u) = -x^T Q_{rew} x - R_{rew} u^2 \tag{7}$$

where

$$Q_{rew} = \begin{bmatrix} 5 & 0 \\ 0 & 0.1 \end{bmatrix}, R_{rew} = 1$$

Table 3. Variables of Inverted pendulum swing-up training.

condition	**values**
init:	random position of pole.
goal:	swinging pendulum up so it maintain upright.
observation variables:	(1) pendulum position $\sin(\alpha)$. (2) pendulum position $\cos(\alpha)$. (3) velocity $\dot{\alpha}$ of the pole.
control action:	(1) motor voltage u.
α range:	$[-\pi, \pi]$.
$\dot{\alpha}$ range:	$[-12\pi, 12\pi]$.
u range:	$[-3, 3]$ V.

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

Figure 2. Illustration of the inverted pendulum swing-up model.

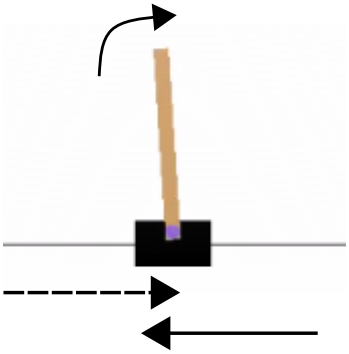

Figure 3. Illustration of the cart pole model.

Cart Pole (CP) The car moves back and forth along a no friction track with the task of balancing a pole attached by an un-actuated joint as shown in Figure 3. The goal is to learn how to swing and balance the pole just by moving the car around the track (Barto et al., 1983; Nagendra et al., 2017). The observation variable, as in inverted pendulum swing-up environment, considers derived information of the angle, below has more details of training and variables characteristics in Table 4. The reward function is represented in Equation (8).

Table 4. Variables Cart pole training.

condition	**values**
init:	car in middle and pole in a random position.
goal:	swing pole up and maintain it upright.
observation variables:	(1) position of the cart on the track (x). (2) angle of the pole with the vertical (θ). (3) cart velocity ($\dot{x}$). (4) angular velocity ($\dot{\theta}$).
control action:	(1) force u applied to the cart.
α range:	$[-\pi, \pi]$.
$\dot{\alpha}$ range:	$[-5\pi, 5\pi]$.
x range:	$[-2.4, 2.4]$.
$\dot{x}$ range:	$[-10, 10]$.
u range:	$[-15, 15]$ V.

$$\rho(x, \theta, \dot{x}, \dot{\theta}) = -2x^2 - 0.1\dot{x}^2 - \theta - 0.1\dot{\theta}^2 \qquad (8)$$

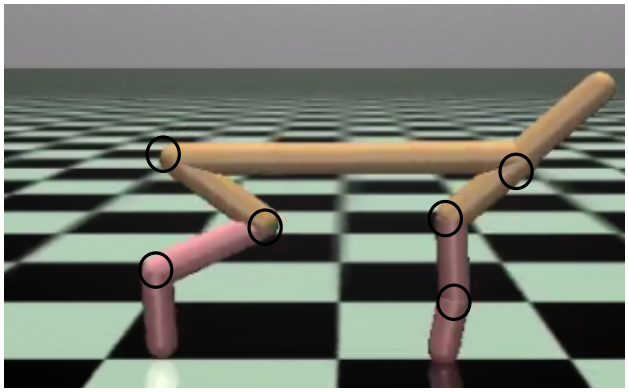

Figure 4. Illustration of the half cheetah model.

Half Cheetah (HC) Half cheetah is a walking animal in 2D environment (Todorov et al., 2012; Wawrzynski, 2007). Figure 4 shows the six joint point of half cheetah which receive torque to control a two-legged walking robot. For each leg, the three degrees of freedom correspond to thighs, shins and feet. The seventeen observation variables and six actions are better described below in Table 5.

Table 5. Variables of Half Cheetah training.

condition	**values**
init:	random variables.
goal:	learn to walk alone.
observation variables:	(1) x position, range $[0, 1]$ meter. (2) y angle, range $[-\pi, \pi]$. (3) hind thigh (bthigh) angle, range $[-\pi, \pi]$. (4) hind shin (bshin) angle, range $[-\pi, \pi]$. (5) hind foot (bfoot) angle, range $[-\pi, \pi]$. (6) front thigh (fthigh) angle, range $[-\pi, \pi]$. (7) front shin (fshin) angle, range $[-\pi, \pi]$. (8) front foot (ffoot) angle, range $[-\pi, \pi]$. (9) x velocity, range $[-1, 1]$ m/s. (10) y angular velocity, range $[-10, 10]$ m/s. (11) z velocity, range $[-1, 1]$ m/s. (12) bthigh angular velocity, range $[-10, 10]$ m/s. (13) bshin angular velocity, range $[-10, 10]$ m/s. (14) bfoot angular velocity, range $[-10, 10]$ m/s. (15) fthigh angular velocity, range $[-10, 10]$ m/s. (16) fshin angular velocity, range $[-10, 10]$ m/s. (17) ffoot angular velocity, range $[-10, 10]$ m/s.
control action:	(1) torque hind thigh, range $[-1, 1]$ N m. (2) torque hind shin, range $[-1, 1]$ N m (3) torque hind foot, range $[-1, 1]$ N m (4) torque front thigh, range $[-1, 1]$ N m (5) torque front shin, range $[-1, 1]$ N m (6) torque front foot, range $[-1, 1]$ N m

5. RESULTS AND DISCUSSION

The execution performance of all the experiments proposed in this study are presented Table 6. The performance of each experiment represents the average of the last 10 accumulated rewards in the learning runs. In table, the final result is presented with an average performance of 30 runs and its confidence interval with 95% confiability, except the half cheetah environment that always uses a 10 run average. The best DDPG algorithm is used as a benchmark for performance comparison. So, the executions of the ensemble, which showed an intersection of the

Table 6. Action Ensemble performances.

		DDPG	Alternately Persistent			Online Learning		
			3 best	*mostly good*	*mostly bad*	*3 best*	*mostly good*	*mostly bad*
PD	Best	-792 ± 14						
	Mean		**-839**±43	−1613±174	−4041±118	−1786±369	−2836±280	−4252±109
	Data Center		**-795**±11	**-804**±10	−1784±406	**-794**±11	−854±24	−2221±326
	Density		**-922**±136	−1080±161	−2361±259	−1412±229	−1607±266	−2834±249
CP	Best	-344 ± 86						
	Mean		**-416**±154	−962±273	−2550±334	−1333±336	−2103±243	−3409±290
	Data Center		**-573**±312	−901±228	−2472±347	**-291**±58	**-484**±167	−1760±345
	Density		**-477**±172	−1551±205	−2770±370	**-344**±120	−1110±193	−1378±194
HC	Best	1419 ± 73						
	Mean		**3474**±822	**2205**±634	551±412	314±302	−53±17	−100±20
	Data Center		**2321**±618	**2040**±423	1332±83	**2159**±342	**1942**±198	**1284**±294
	Density		**2205**±459	**1441**±195	1041±110	**1424**±415	613±392	555±267

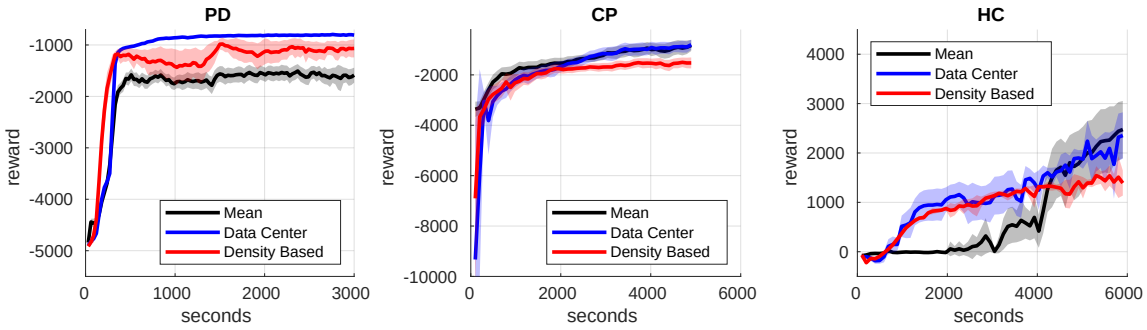

Figure 5. Alternately Persistent learning the *mostly good* hyperparametrizations.

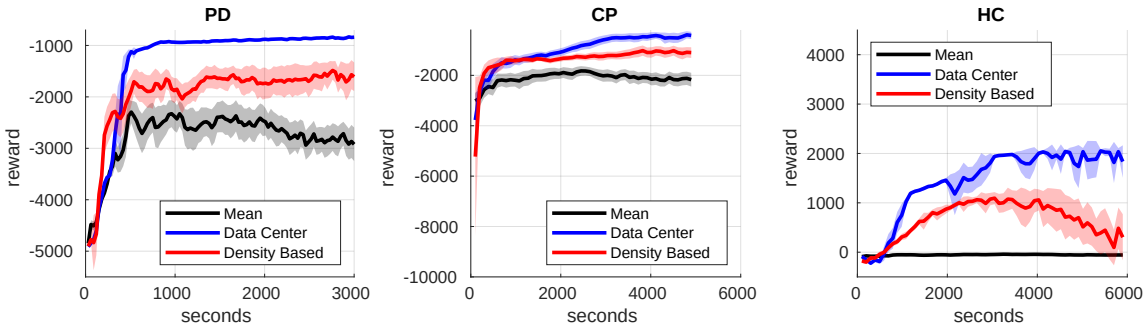

Figure 6. Online learning the *mostly good* hyperparametrizations.

mean and its confidence interval with *Best* DDPG, are highlighted in bold. The number of runs of the experiments seeks to achieve an expected reliability for comparison purposes (Triola, 2015).

Analysing Table 6, the *3 best* in Alternately Persistent performs very well in any chosen strategy. The mean strategy works very well in this learning mode, but it does not work for Online Learning. Note that the *3 best* with the Data Center strategy in Online Learning performs equal to or better than Alternately Persistent, but with a narrower confidence interval. This leads to more stable learning. This stability becomes interesting when we compare the *mostly* good and its results. For this case, the Data Center Online Learning always converges, although the ensemble has algorithms that are not capable of learning.

Besides that, the Data Center strategy in the *3 best* and *mostly good* ensembles shows very good results when compared with *best* DDPG, the exception is *mostly good* of Cart Pole in Alternately Persistent. Data Center strategy good surprise is that the *mostly bad* of the half cheetah environment has shown good results when compared to the Best. On other side, Density based strategy seems to be a good strategy, however it does not stand out in comparison to the others, so some cases that have poor performance struggle with the need for parameterization.

To show the learning behaviour, the *mostly good* group is illustrated in all environments with the aggregation techniques. The Alternately Persistent learning is presented in Figure 5, the inverted pendulum presents distinct performance for each aggregation, cart pole struggles to reach a better performance and half cheetah showed a huge performance leap in early seconds between the Data Center and Mean, yet both ending with the equal performance. Figure 6 presents the Online learning, Data Center presents a better performance in all environments. Note that, the other aggregations mostly present the worst behavior compared to Figure 5.

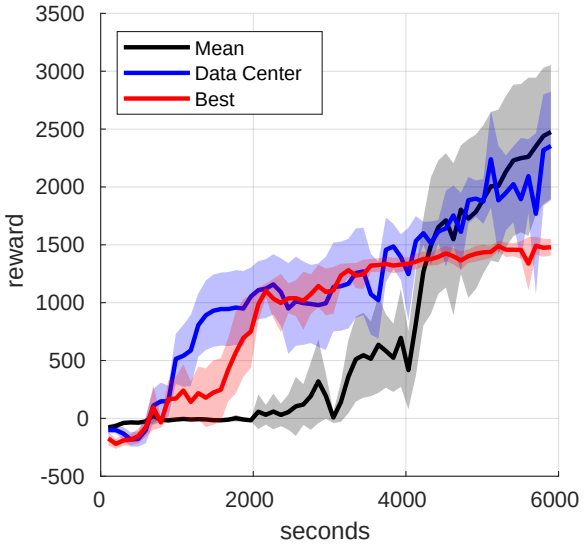

Figure 7. Half Cheetah performance for the *mostly good* hyperparametrizations in Alternately Persistent.

Figure 7 compares the Best algorithm learning curve with *mostly good* group trained with Alternately Persistent learning in Half Cheetah environment. Data Center presents an early learning convergence than others and surpass the performance of Best algorithms. The mean aggregation works well at the end, even though it presents low capacity in the initial stage of learning.

6. CONCLUSION

This article proposes a comparison with the continuous action ensemble techniques used in the literature. It demonstrates that it is possible to improve the grid search performance with the use of DDPG ensemble. When testing an ensemble with *3 best* DDPG it is not possible to highlight the best strategy, however when we use *mostly good* we realize that the Data Center strategy presents good performance and the narrowest confidence interval. When using algorithms with unreliable hyperparameter tuning, *mostly bad* groups, the Data Center demonstrated capability to learn as a fine-tuning single agent. Using Alternately Persistent learning, with fine-tuning algorithms, the Mean aggregation is an obvious choice for 3-algorithm ensemble. As future work, other aggregation techniques can be used in addition to other environments.

REFERENCES

Anschel, O., Baram, N., and Shimkin, N. (2017). Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning. 176–185.

Barto, A.G., Sutton, R.S., and Anderson, C.W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5), 834–846. doi: 10.1109/TSMC.1983.6313077.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *CoRR*.

Busoniu, L., Babuska, R., De Schutter, B., and Ernst, D. (2017). *Reinforcement learning and dynamic programming using function approximators*. CRC press.

Duell, S. and Udluft, S. (2013). Ensembles for continuous actions in reinforcement learning. In *ESANN*.

Faußer, S. and Schwenker, F. (2011). Ensemble methods for reinforcement learning with function approximation. In *International Workshop on Multiple Classifier Systems*. Springer.

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

Gu, S., Holly, E., Lillicrap, T., and Levine, S. (2017). Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 3389–3396.

Ha, S., Xu, P., Tan, Z., Levine, S., and Tan, J. (2020). Learning to walk in the real world with minimal human effort. *arXiv preprint arXiv:2002.08550*.

Kingma, D.P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. *Proceedings of International Conference on Learning Representations*.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M.A. (2013). Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602. URL `http://arxiv.org/abs/1312.5602`.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529–533.

Nagendra, S., Podila, N., Ugarakhod, R., and George, K. (2017). Comparison of reinforcement learning algorithms applied to the cart-pole problem. In *International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE.

Popov, I., Heess, N., Lillicrap, T.P., Hafner, R., Barth-Maron, G., Vecerík, M., Lampe, T., Tassa, Y., Erez, T., and Riedmiller, M.A. (2017). Data-efficient deep reinforcement learning for dexterous manipulation. *CoRR*, abs/1704.03073. URL `http://arxiv.org/abs/1704.03073`.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on International Conference on Machine Learning*, volume 32, 387–395. JMLR.org, Bejing, China.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent*

Robots and Systems, 5026–5033. IEEE.

Triola, M.F. (2015). *Elementary Statistics Technology Update*. Pearson, 11th edition.

Uhlenbeck, G.E. and Ornstein, L.S. (1930). On the theory of the brownian motion. *Physical review*, 36, 823.

Wawrzynski, P. (2007). Learning to control a 6-degree-of-freedom walking robot. In *EUROCON 2007-The International Conference on "Computer as a Tool"*, 698–705. IEEE, Warsaw, Poland.

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

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