Sample-Efficient Reinforcement Learning for Continuous Actions with Continuous BDPI

Denis Steckelmacher
Vrije Universiteit Brussel
Brussels, Belgium
denis.steckelmacher@vub.be

Hélène Plisnier
Vrije Universiteit Brussel
Brussels, Belgium
helene.plisnier@vub.be

Ann Nowé
Vrije Universiteit Brussel
Brussels, Belgium
ann.nowe@vub.be

ABSTRACT

Bootstrapped Dual Policy Iteration [23, BDPI] is a model-free Reinforcement Learning algorithm that combines several off-policy critics with an actor robust to off-policy critics. The critics are trained using a variant of Q-Learning, and the actor imitates their average greedy policies with Conservative Policy Iteration. BDPI achieves state-of-the-art sample-efficiency in discrete-action domains, but is inapplicable to continuous-action domains, as both the actor and critic update rules rely on the ability to enumerate the actions. In this paper, we present a novel implementation of the BDPI ideas, off-policy critics and an actor, for continuous actions. Our actor is built around a single discriminator network, easy to train to imitate greedy policies, and the critics take inspiration from the off-line batch RL literature to allow off-policy learning. In this early work (visionary) paper, we show that our Continuous BDPI is several times more sample-efficient than the Soft Actor-Critic on BipedalWalker, using naive hyper-parameters. We explain in our experimental section how future work will allow to characterize the behavior of Continuous BDPI regarding its hyper-parameters, which will allow it to be applied to more environments.

KEYWORDS

Model-Free Reinforcement Learning, Continuous Actions, Sample-Efficiency

1 INTRODUCTION

Real-world Reinforcement Learning problems, such as advanced legged locomotion [13] or the control of industrial machines [12], rely on the agent being able to learn a good policy with few interactions in the environment, and few mistakes. The currently most prevalent approach to ensure learning with few interactions is the use of simulators, in which the agent learns before being deployed in the real world. Because designing accurate simulators for real-world tasks is challenging, most approaches rely on an approximate but noisy simulator, in which the agent learns a policy that acts "carefully", and does not rely on precise dynamics to achieve its goal [2, 8, 30].

Another research direction does not rely on simulators, but instead focuses on designing agents that quickly learn directly in the real world [14]. The advantage of these methods is that no simulator needs to be designed at all, which reduces the engineering effort of applying Reinforcement Learning to a task. An example of such sample-efficient model-free (and simulator-free) Reinforcement Learning algorithm, that requires no domain-specific knowledge, is Bootstrapped Dual Policy Iteration (BDPI) [23].

While BDPI is several times more sample-efficient than PPO and ACKTR in various discrete-action Reinforcement Learning environments [23], its formalism strongly relies on an action set that can be enumerated, which prevents its application to continuous-action tasks, more common in the real world. In this paper, we propose a continuous-action model-free Reinforcement Learning algorithm that builds on the core ideas of BDPI, but completely changes how its actor and critics are implemented, to allow for continuous actions. We then demonstrate that our Continuous BDPI algorithm achieves sample-efficiencies several times higher than the Soft Actor-Critic in BipedalWalker, a challenging physics-based task available in the OpenAI Gym [3], with a high-dimensional state-space and a 4-dimensional action space. This result is highly encouraging, as it has been obtained using non-tuned hyper-parameters, and most current work at improving sample-efficiency with continuous actions usually obtains gains of about 10% to 30% (not 200% as in this paper). As corollary, our results also show that the high sample-efficiency of BDPI comes from its core ideas (an actor and several off-policy critics), not its particular discrete-action implementation.

2 BACKGROUND

Our Continuous BDPI algorithm builds on the BDPI core ideas, along with several theoretical and algorithmic foundations that we review in this section.

2.1 Discrete-action BDPI

Bootstrapped Dual Policy Iteration is an actor-critic algorithm, in which the agent trains one stochastic actor and several critics. The use of several critics has been shown to be beneficial to stability and exploration in [19]. Every time-step t, the actor observes a state s_t, selects an action a_t that is executed in the environment, leading to a reward r_t and next-state s_{t+1}. This (s_t, a_t, r_t, s_{t+1}) experience tuple is added to an experience buffer.

Periodically (every time-step in the BDPI paper, but the frequency can change), $N > 0$ batches of $M > 0$ experiences are sampled uniformly at random in the experience buffer. Each batch is then used to train one of the critics, selected at random, by producing target Q-Values with Equation 1. These target Q-Values are used to optimize a critic network on the Mean Squared Error. Sequentially after each critic is trained, the same batches of experiences, along with their updated Q-Values, are used to train the actor with Equation 2. Updated action probabilities, produced by Equation 2, are also used to optimize a neural network on the Mean Squared Error:

$$Q^{i,A}_{k+1}(s_t, a_t) = Q^{i,A}_k(s_t, a_t) + \alpha\big(r_t + \gamma V_k(s_{t+1}) - Q^{i,A}_k(s_t, a_t)\big) \quad (1)$$

$$V_k(s_{t+1}) \equiv \min_{l=A,B} Q^{i,l}_k\big(s_{t+1}, \text{argmax}_{a'} Q^{i,A}_k(s_{t+1}, a')\big)$$

$$\pi_{k+1}(s) = (1-\lambda)\pi_k(s) + \lambda\Gamma(Q^{i,A}_{k+1}(s,\cdot)) \quad (2)$$

with $Q^{i,A}$ and $Q^{i,B}$ the two Q-networks of critic number i, following the Clipped DQN formalism [4], $\alpha > 0$ the critic learning rate, λ the actor learning rate, and Γ the greedy function, that returns a vector of one element per discrete action available to the agent, 0 for every action but the one having the largest Q-Value, that has its element set to 1.

BDPI relies on discrete action-spaces in its two main equations: Equation 1 represents an off-policy Q-Learning-like update rule, that needs to compute a maximum over actions; and Equation 2 uses the greedy function, that computes an arg-maximum over actions. We now review algorithms and approaches that will allow us, in Section 3, to implement a continuous-action version of BDPI.

2.2 Off-policy Critics with Continuous Actions

Almost every Reinforcement Learning algorithm compatible with continuous actions either uses no critic, such as vanilla Policy Gradient [24], or combines an actor with a critic that evaluates the actor, such as A3C [16], PPO [20], ACKTR [28], TD3 [4] or the Soft Actor-Critic [7]. Some algorithms, such as ACER [26] and the Soft Actor-Critic, allow the behavior of the agent to be distinct from its actor, but the critic must still evaluate the actor (not the behavior policy). They use importance sampling, off-policy corrections [18], or *soft* update rules in which the actor intervenes, to ensure that the critic learns Q-Values that evaluate the actor.

Training a critic with continuous actions, in an off-policy way (thus without requiring an actor or tying the update rule to a behavior policy), seems highly challenging. One of the only algorithms, to our knowledge, that manages to get close to this objective is BCQ, for Batch-Constrained Deep Q-Learning [5]. BCQ builds on Q-Learning, but replaces the maximum over actions $\max_{a' \in A} Q(s_{t+1}, a')$ with a maximum over sampled actions, $\max_{a' \sim P(a|s_t)} Q(s_{t+1}, a')$. Fujimoto et al. [5] show that uniformly sampling the continuous action space would lead to unstable learning, as the Q-Function would be sampled in regions in which it has never been trained, and in which it is highly inaccurate. To overcome this limitation, they introduce a Conditional Variational Auto-Encoder [1, VAE], a neural network that is able to learn distributions of data. BCQ trains a VAE on the distribution of actions visited by the agent, conditioned on states. The VAE is then used to implement $P(a|s_t)$ such that only actions present in the experience buffer, for states around s_t, are being sampled when computing Q-Values.

We point out that the main contribution of BCQ is a method that allows to query a critic only in areas where it should have high accuracy. Other approaches that fulfil this objective exist. For instance, uncertainty estimation such as proposed in [25] could allow to restrict $P(a|s_t)$ to actions for which the critic has high certainty. The Behavior-Regularized Actor-Critic framework [29] introduces a penalty term, that decreases the Q-Values of actions that are not often executed by the agent, and biases learning towards policies close to the behavior policy of the agent. BEAR [11] uses the Maximum Mean Discrepency (MMD) to only query the critic on actions for which it has support, so for which it has been "sufficiently" trained. The main difference between BEAR and BCQ is that BCQ samples actions proportionally to how often they are executed by the agent, while BEAR considers a harder threshold between executed-enough and not-executed-enough actions.

In this paper, we use the BCQ approach, with a Variational Autoencoder, as it is both simple and makes few assumptions about the rest of the agent. Other approaches, such as BRAC or BEAR, assume that the critic is able to query an actor for actions, which is not the case in BDPI (the actor imitates the greedy policies of the critics, but the critics must not know that the actor even exists).

2.3 Maximizing Q-Values

In continuous BDPI, we need to compute which action in a particular state has the largest Q-Value: $max_{a'} Q(s, a')$. With discrete actions, this simply requires enumerating every possible action. With continuous actions, and without making any assumption about the Q-Function, this requires a form of non-convex function optimization.

Several function optimization approaches exist. In the Reinforcement Learning literature, BCQ samples actions from a Variational Auto-Encoder (see Section 2.2). [15] propose the use of the Cross-Entropy Method, an iterative method that repeatedly samples actions, keeps the top K ones, and samples a new batch of actions from a normal distribution defined by the mean and variance of the top K actions. Other function optimization algorithms exist, such as Genetic Algorithms [21], Ant Colony optimization [22], or *gradient descent*, reviewed and explained in a recent book by Kochenderfer and Wheeler [10].

In Continuous BDPI, we use the VAE approach of BCQ when computing the value of a next state, $V(s_{t+1}) = \max_{a' \sim D(s_{t+1})} Q(s_{t+1}, a')$. When computing the greedy function, used to train the actor, we use simple rejection sampling: we sample a large amount of actions uniformly at random in the space of actions allowed by the environment, and select the one with the largest Q-Value. We evaluated other optimization approaches (gradient ascent, the cross-entropy method), that did not work for our agent. We believe that anything more advanced than sampling random actions and keeping the best one somehow biases the output of the greedy function, which reduced the exploration of the actor in our experiments.

2.4 Generative Models

Most function approximators, such as neural networks, by default produce deterministic functions: given an input, they will always produce the same output. However, discrete-action BDPI relies on a stochastic policy, that maps a state to a probability distribution over actions [23]. State-of-the-art continuous-action Reinforcement Learning algorithms usually implement their stochastic policy as a parameterized Gaussian distribution [20]: the actor network produces a mean and a standard deviation, used to sample actions from the Gaussian distribution these values define. However, another approach is possible: generative models.

Generative models are able to produce samples generated from an *arbitrary* distribution (not just a Gaussian, or some other designer-specified distribution). Both families of generative models, Generative Adversarial Networks [6] and Variational Autoencoders [1], learn how to map uniform noise to outputs that follow some training distribution.

On-line model-free Reinforcement Learning, as considered in this paper, brings an important challenge for generative models: the agent must progressively learn, which means that it has to explore, and that the set of states and actions visited by the agent changes over time. Conventional GANs and Variational Autoencoders are able to learn good approximations of arbitrary distributions, *when presented with a full dataset at once*. An un-trained Variational Autoencoder has a tendency of mapping every possible input to the same output, close to the vector of all zeroes. This means that using a VAE as a policy, in a Reinforcement Learning setting, would lead to an agent that does not explore.

In the next section, we introduce Continuous BDPI. We use a VAE as part of the critic learning procedure, taking inspiration from BCQ (see Section 2.2). For the actor, we use neither a VAE nor a GAN, as they are unable to explore in the early stages of learning. Instead, we introduce a pure discriminator network, that we initialize and train to ensure that the resulting policy maximally explores at the early stages of learning, and becomes more deterministic as time passes.

3 CONTINUOUS BDPI

Implementing a continuous-actions version of BDPI is not as simple as replacing discrete actions with vectors of floating-point values. Learning Q-Values with an off-policy update rule, akin to Q-Learning, requires a solution to the problem of having an infinite amount of possible action in every state. In Section 2.2, we review several algorithms that tackle this challenge. However, simply combining these algorithms with components from discrete-action BDPI would not lead to a working algorithm. Before presenting our main contribution, Continuous BDPI, we first review the main high-level properties of discrete-action BDPI, point out where current algorithms fail to meet these properties, and illustrate what our solutions are:

Critics unaware of the actor
In discrete-action BDPI, the critics are trained with a Q-Learning-family algorithm, that learns from experiences from an experience buffer, without relying on the presence of an explicit actor or model of the behavior policy. In Continuous BDPI, we modify Batch-Constrained Deep Q-Learning [5] to use no actor in its update rules.

Stable Q-Values for non-executed actions
Most Reinforcement Learning algorithms designed after DQN [17] represent discrete-action Q-Values with a single neural network, that takes a state s as input and produces one output per action a, for $Q(s, a)$. Almost every implementation of these algorithms[1] computes the loss of the neural network by *only considering executed actions*. When updating its weights, the neural network will only try to fit the updated Q-Values of these actions, while allowing the Q-Values of the other actions to drift unsupervised. BDPI is different: the executed actions will be fitted towards an updated Q-Value, and the other ones will be fitted *to their original values*, to ensure stability. Obtaining this behavior in Continuous BDPI requires much attention to how the critics are trained.

An actor that starts stochastic
Discrete-action BDPI assumes that, when the agent has just started learning, the still-untrained actor outputs a uniform probability distribution over actions, that becomes more deterministic only as learning progresses. For Continuous BDPI, the main challenge is that generative models *do not* exhibit this property of high stochasticity when untrained. An untrained VAE, for instance, will map any noise vector to an output close to all-zeroes. We introduce a discriminator-based actor that ensures that the actions taken by the agent have high stochasticity in states that are not often visited.

[1]Such as https://github.com/sweetice/Deep-reinforcement-learning-with-pytorch/blob/master/Char01%20DQN/DQN.py#L92 for a clear example.

3.1 General design of the critics

Continuous BDPI trains several critics and one actor, based on experiences collected by a behavior policy (the actor), and stored in an experience buffer. Taking inspiration from BCQ [5], each critic of Continuous BDPI uses three neural networks: Q^A and Q^B, that learn Q-Values for state-action pairs in a way comparable to Clipped DQN [4], and G, a Conditional Variational Autoencoder that learns the distribution of actions executed by the agent, conditioned on a state. Each critic has its own Q^A and Q^B networks, that share no weights. However, a single VAE G is shared across all the critics of the agent, and is only trained by the first critic. This dramatically increases the compute-efficiency of the algorithm, at no cost in learning performance, as G learns a quite simple supervised task.

3.1.1 Conditional VAE. The Conditional VAE network G consists of an encoder $E(s, a)$ and a decoder $D(s, z)$. The encoder concatenates the state and action, then passes them through feed-forward hidden layers with the ReLU activation function. Finally, the last layer produces an embedded representation of the state-action tuple, a vector of 8 values for the *mean* μ, and a vector of 8 values for the *variance* σ^2 of a Gaussian distribution. These two outputs of the encoder have a linear activation function. When training, the mean and standard deviations predicted by the encoder are used to sample a noise element $z \sim \mathcal{N}(\mu, \sigma^2)$. The decoder concatenates a state s and noise element z, then passes the result through feed-forward hidden layers with the ReLU activation function. Finally, the last layer produces an action a, with a linear activation function.

$$L^{VAE} = L_r^{VAE} + \frac{1}{2} L_{KL}^{VAE} \tag{3}$$

$$L_r^{VAE} = \frac{1}{N} \sum_{i=0}^{N} (D(s_i, z \sim E(s_i, a_i)) - a_i)^2 \tag{Recon}$$

$$L_{KL}^{VAE} = -0.5 \frac{1}{N} (1 + \log(\sigma_i^2) - \mu_i^2 - \sigma_i^2) \quad \mu_i, \sigma_i^2 = E(s_i, a_i) \tag{KL}$$

When training, the VAE loss [1] shown in Equation 3 is optimized on batches of states and actions. The loss encourages the VAE to output predicted actions as close as possible to input actions (the *reconstruction* loss), and an additional term, built on the Kullback-Leibler (KL) divergence, encourages the encoder to produce noise means and variances close to a normal distribution $\mathcal{N}(0, 1)$. In prediction mode, a state and a randomly-sampled noise vector $z \sim \mathcal{N}(0, 1)$ is given to the decoder, that predicts an action. Predicting

several actions, with the same state but different z noise vectors, allows to obtain actions drawn from a distribution that closely resembles the distribution of actions executed by the agent in the state.

3.1.2 Q^A and Q^B networks. The main goal of the Conditional VAE is to have a way to produce sets of actions, given a state, that are close to actions that have historically been executed by the agent around that state. BCQ relies on these action samples to implement a Q-Learning-inspired update rule, that we use in Continuous BDPI:

$$Q(s_t, a_t) = Q^A(s_t, a_t) + \alpha(r_t + \gamma V(s_{t+1}) - Q^A(s_t, a_t)) \quad (4)$$
$$V(s_{t+1}) \equiv \max_{a' \sim D(s_{t+1}, z \sim \mathcal{N}(0,1))} \min_{l=A,B} Q^l(s_{t+1}, a')$$

with (s_t, a_t, r_t, s_{t+1}) an experience tuple uniformly drawn from the experience buffer. Q^A and Q^B are neural networks that take as input a state and an action, and produce a single output, the Q-Value of the action in the state. The exact architecture of these neural networks is detailed in the next section. Every critic of Continuous BDPI samples a batch B of N experiences from the buffer, then computes Equation 4 for each of these samples. Then, the Q^A neural network is trained to minimize $\sum_{s,a \in B}(Q^A(s,a) - Q(s,a))^2$.

As in discrete-action BDPI, every time-step, several batches of experiences are sampled from the experience buffer, and used to train several critics. Each critic will itself perform several *training iterations*, that each consist of computing Equation 4 and fitting Q^A, then swapping the roles of Q^A and Q^B. When training Q^A on the mean-squared-error loss described above, several gradient steps are performed.

3.2 Special aspects of the critics

The previous section introduces the parts of the Continuous BDPI critics that closely resemble the BCQ algorithm. We now detail which components of our critic are significantly different from BCQ.

3.2.1 Stabilizing non-executed actions. As mentioned in the beginning of Section 3, discrete-action BDPI has several key properties that we aim at replicating in Continuous BDPI. One of them is that the Q-Values of actions that are not the one executed in a given state need to be explicitly kept stable. We achieve that by tuning how the Q^A network of each critic is trained: the set S of states sampled from the experience buffer is duplicated, to form $S' = \{S, S\}$. The set A' of actions consists of the actions A sampled from the experience buffer, and for which updated Q-Values have been computed, concatenated with the same number of actions $\tilde{A}$, uniformly sampled in the range of actions allowed by the environment, to form $A' = \{A, \tilde{A}\}$. We stress that $\tilde{A}$ is *uniformly sampled*, not sampled using the VAE. In expectation, this allows the actions often executed by the agent to have updated Q-Values, while actions not often executed by the agent have their Q-Values fixed. Finally, the set Q' of target Q-Values is formed by concatenating Q, the set of updated Q-Values for the executed actions, and $Q^A(S, \tilde{A})$, a snapshot of what Q^A predicts before being updated, for the randomly-sampled actions. When fitting Q^A on $(S', A') \rightarrow Q'$, this forces Q^A to maintain stable predictions for non-executed actions.

3.2.2 Dueling architecture. Training a critic off-policy, especially with continuous actions, is extremely challenging regarding the stability of the Q-Values. The use of Q^A and Q^B in a Clipped DQN fashion helps, but, with continuous actions, is not enough to prevent the Q-Values to degenerate to large positive values (in some runs) or very negative values (in other runs, at random). To address this problem, we draw inspiration from Dueling DQN [27]: the Q-Value of an action in a state is $Q(s,a) = V(s) + A(s,a)$, with $A(s,a) \leq 0$ the *advantage value* of an action in a state. The advantage is always negative, and zero for the greedy action.

[27] use a special loss, in which the average advantage value of the (discrete) actions in a state s intervenes, to force the Q-network to learn a meaningful V function in addition to advantage values. In Continuous BDPI, we instead rely on the negative nature of $A(s,a)$. We implement the Q^A and Q^B networks of the critic so that two sets of layers, that share no weights, separately produce $V(s)$, a scalar, with a linear activation function, and $\bar{A}(s,a)$, a scalar too, with the sigmoid activation function. Then, the final output of the network is computed as $Q(s,a) = V(s) - \rho\bar{A}(s,a)$. The sigmoid activation function forces $\bar{A}$ to be positive, so $-\bar{A}$ to be negative. This forces the network to learn a meaningful $V(s)$ function, that does not depend on the action, and advantage values. ρ is a hyper-parameter, that represents the maximum difference between the Q-Values of any two actions in a given state, the *range* of the advantage values. We set ρ to 4 in our experiments.

3.3 The actor

After being trained, each critic produces greedy actions: they map every state s in their batch of experiences to an action $a^* = \max_{a'} \min_{l=A,B} Q^l(s,a')$ that has the largest Q-Value, among a sample of actions *uniformly sampled* in the range of actions allowed by the environment. The actor is then trained to imitate these actions. We do not use the VAE for this sampling operation, to ensure that the set of greedy actions sent to the actor is not biased towards what

Algorithm 1 Continuous Bootstrapped Dual Policy Iteration

for every critic $i \in [1, N_c]$ in random order **do**
  $E \leftarrow$ N experiences sampled from the buffer
  if $i = 0$ **then**
    Trained the shared critic VAE on (s,a) tuples from E
  end if
  for N_t training iterations **do**
    Swap $Q^{A,i}$ and $Q^{B,i}$
    Compute Q (updated Q-Values) with Equation 4
    S and A are the sets of states and actions from E
    $S' \leftarrow \{S, S\}$, $A' \leftarrow \{A, \tilde{A}\}$, with $\tilde{A}$ uniformly sampled
    $Q' \leftarrow \{Q, Q^{A,i}(S, A')\}$
    Minimize $\langle (Q^{A,i}(S', A') - Q')^2 \rangle$ for several gradient steps
  end for
  $A^* \leftarrow \{\text{argmax}_{a'} Q^{A,i}(s,a')\}, \forall s \in E, a'$ uniformly sampled
  $S' \leftarrow \{S, S\}, A' \leftarrow \{A, \tilde{A}\}, P' \leftarrow \pi(S', A')$
  Update P' with Equation 5
  Minimize $\langle (\pi(s,a) - P'(s,a))^2 \rangle \quad \forall s \in S', a \in A'$
end for

the agent usually executes (what the VAE learns to predict). This dramatically improves exploration, stability and final performance.

3.3.1 The problem with VAEs. In discrete-action BDPI, the actor outputs a discrete probability distribution over actions. When untrained, and also in states that are not visited often, the actor predicts a probability distribution that is *close to uniform*. Then, as learning progresses, Equation 2 progressively increases the probabilities of actions that are often identified as greedy by the critics, and decreases (because a discrete probability distribution sums to 1) the probabilities of actions that are not often greedy. This results in an actor that progressively becomes more deterministic as the agent learns.

A simple method of replicating the BDPI architecture in Continuous BDPI would be to use a VAE for the actor. The VAE would then be trained to imitate the distribution of greedy actions as generated by the critics. However, while this would work given a large and diverse experience buffer, using a VAE as actor degenerates in practice: when not yet trained, a VAE produces outputs drawn from a very narrow distribution centered around all-zero vectors. This prevents the agent from exploring actions of large magnitudes. This degenerate output when untrained is not a problem when using VAEs in a supervised learning setting, as a big dataset is available from the get go. But for a Reinforcement Learning actor, exploration is key and *how the actor progressively learns* matters just as much as *how well it performs at the end.*

3.3.2 Discriminator-based Actor. Instead of a VAE, we therefore introduce a discriminator-based actor. The actor $\pi(s, a)$ is a neural network that takes a state and action as input, concatenates them, and passes them through feed-forward hidden layers with the ReLU activation function. Then, a last layer produces one single output, with the sigmoid activation function, $P(a|s, \pi)$, the probability that the actor accepts an action a in a state s.

Actions are selected by the actor by uniformly sampling a large number of candidate actions $\hat{A}$ (4096 in our experiments), then querying the actor for their probabilities $P = \pi(s, A)$. Then, actions and their probabilities are sequentially enumerated, and each action $a \in \hat{A}$ is *accepted* with a probability P_a. As soon as an action is accepted, the actor returns it and the action is executed in the environment. This Monte-Carlo sampling method allows to draw actions $a \sim P(a|s, \pi)$ from an arbitrary policy learned by the actor.

When the actor network is still untrained, and in states that are not often visited, the output of the network tends to be the same for every action. No action therefore has preference, and the sampling method described above leads to actions selected uniformly at random in the set of actions allowed by the environment. When the actor learns, it progressively gives preference to some actions, which implement the BDPI property of an actor that gets more deterministic over time.

3.3.3 Training the Actor. The actor is trained as follows: as with the critics, the set S of states sampled from the experience buffer is duplicated, to form $S' = \{S, S\}$; then, random actions $\tilde{A}$ are uniformly sampled in the range of actions accepted by the environment, and combined with the greedy actions A^* produced by the critic, to form $A' = \{A^*, \tilde{A}\}$. The actor then predicts its current probabilities for these states and actions, to produce $P' = \{P^+, P^-\} = \pi(S', A')$. Note that the first $|S|$ outputs of the actor correspond to actions that are identified as greedy by the critic, while the remaining $|S|$ outputs correspond to uniformly-sampled counter-example actions. Following the discrete BDPI Equation 2, P' is adjusted as follows:

$$P'_i \leftarrow \begin{cases} P'_i + \lambda & \text{if } i < |S| \quad (P'_i \in P^+) \\ P'_i - \lambda & \text{if } i \geq |S| \quad (P'_i \in P^-) \end{cases} \tag{5}$$

with λ the actor learning rate. This update equation leads to the probability given to actions that are often greedy to increase, while the other actions see their probabilities decrease. This implements the BDPI actor, that starts fully uniform over the whole action space, and becomes more deterministic as it is trained on greedy actions produced by the critics.

Pseudocode that summarizes how to train the actor and critics of Continuous BDPI is given in Algorithm 1. We now provide empirical evidence that Continuous BDPI allows to learn high-quality policies, and achieves a sample-efficiency that is several times higher than state-of-the-art model-free Reinforcement Learning algorithms.

4 EXPERIMENTS

We evaluate Continuous BDPI in BipedalWalker [3], an environment that we choose because it is one of the most complex continuous-action environments in the Gym, that does not require a Mujoco license. It is also an environment for which it is easy to find high-quality baselines online, the one we report (SAC) being the best we found.

4.1 Environment

BipedalWalker-v3, available in the OpenAI Gym [3], is a 2D physics-based environment in which a small agent with two articulated legs has to learn to walk without falling. The agent observes vectors of 24 readings, consisting on various joint angle and velocities, along with 10 LIDAR readings that allow the agent to sense the terrain around it. The action is a vector of 4 floating-point values, all in the $[-1, 1]$ interval, that control the torque applied to the hip joints of the two legs, and their knee joints. The reward is -100 if the agent falls on the floor (its body hits the floor), and positive as the agent advances towards the right. Reaching the "end" of the environment leads to a total return of +300. Some negative rewards are also given proportionately to the energy consumed by the agent, so smoother movement leads to higher rewards. The episode ends when the agent falls on the floor, reaches the end of the environment, or after 1600 time-steps.

4.2 Algorithms

We compare Continuous BDPI to the currently state-of-the-art Soft Actor-Critic [7], for which we use a high-quality implementation.[2] This implementation performs the best on BipedalWalker, compared to about a dozen other implementations of the Soft Actor-Critic and other algorithms.

The Soft Actor-Critic uses hyper-parameters as found in the literature, with some of them tuned for BipedalWalker. Its neural networks have 2 hidden layers of 256 neurons. The batch size is set

[2] https://github.com/Rafael1s/Deep-Reinforcement-Learning-Algorithms/blob/master/Ant-PyBulletEnv-Soft-Actor-Critic/

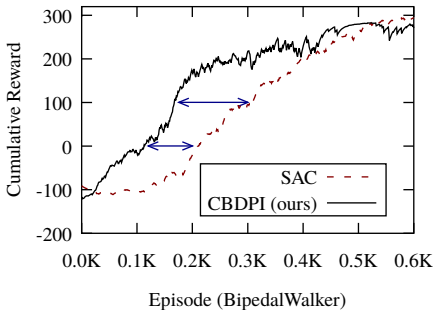

Figure 1: Returns per episode for Continuous BDPI (CBDPI) and the Soft Actor-Critic (SAC), on BipedalWalker. Continuous BDPI is almost two times more sample-efficient than the Soft Actor-Critic.

to 256. The replay buffer size contains 100K samples. We use the Adam optimizer [9], with a learning rate of 0.0001. Larger learning rates led to unstable learning.

Continuous BDPI has been configured taking inspiration from the discrete-action BDPI paper, along with some tuning to maximize *compute-efficiency*, not sample-efficiency. We therefore compare a sample-efficiency-optimized SAC with a Continuous BDPI agent that, due to limited compute power available, does not reach its full potential. Because Continuous BDPI trains many critics on large batch sizes, for several gradient steps every environment time-step, compute-efficiency becomes a critical concern in a simulated environment. We therefore use 16 critics, all of them being trained every time-step for a single training iteration ($N_t = 1$ in Algorithm 1) on a batch of 512 experiences sampled from an experience buffer of at most 1M experiences. The actor and critics are neural networks with one hidden layer of 256 neurons, instead of 2 hidden layers for the Soft Actor-Critic. The VAE network has an encoder and a decoder that each have one hidden layer of 256 neurons, and a latent dimension of 8. When training the actor and critics, 10 gradient steps are performed per training iteration, with the Adam optimizer as for the Soft Actor-Critic, and a learning rate of 0.0001. The actor learning rate of Equation 5, and the critic learning rate of Equation 4, are both set to 0.1. When sampling the actor for an action, 4096 uniformly-sampled actions are drawn and submitted to the discriminator for acceptance (or not). When computing the greedy function, 200 actions are sampled uniformly and queried for their Q-Values. When computing the maximum of Equation 4, 50 actions are sampled from the VAE's generator and queried for Q-Values. Finally, the maximum difference of Q-Values between actions in a state, ρ in Section 3.2.2, is set to 4.

4.3 Results

Figure 1 compares Continuous BDPI to the Soft Actor-Critic. The lines are averages of 8 runs for the Soft Actor-Critic, and 4 for Continuous BDPI, that has far less run-to-run variance. The main takeaway is that the approach we propose, with several off-policy critics and a discriminator-based actor, leads to a significant improvement in sample-efficiency compared to the Soft Actor-Critic, at no cost to final policy quality (SAC stabilizes at returns around 300, Continuous BDPI, *with its much smaller neural networks*, stabilizes at around 290). Continuous BDPI is almost twice as sample-efficient as SAC overall, and several times more sample-efficient than SAC in the early stages of learning. For real-world tasks, this means that a Continuous BDPI agent quickly stops being very bad, which may significantly decrease the risk and cost of training a Reinforcement Learning agent with continuous actions.

The high sample-efficiency of Continuous BDPI also has much theoretical significance, as the discrete-action BDPI paper [23] also shows several-times gains in sample-efficiency compared to state-of-the-art algorithms. To us, this demonstrates that the theoretical foundations and properties of BDPI, as listed in Section 3, are the main source of sample-efficiency, be it with discrete or continuous actions.

For improved clarity, we omit from Figure 1 several ablations of Continuous BDPI, that led to flat lines at about -110 (no learning at all). Removing the dueling architecture, not ensuring that the Q-Values of non-executed actions remain stable, or using a VAE for the actor instead of our discriminator, all prevent learning. This shows that all the components of Continuous BDPI are necessary for learning, and rely on each other.

5 CONCLUSION

In this visionary paper, we presented a highly sample-efficient model-free Reinforcement Learning algorithm for continuous actions. We observe that a highly sample-efficient algorithm for *discrete actions*, BDPI, exhibits some key theoretical properties, such as the use of several off-policy critics. We then propose a completely original algorithm for continuous actions, Continuous BDPI, that exhibits the same theoretical properties. Finally, we empirically demonstrate that Continuous BDPI achieves a sample-efficiency almost twice higher than the Soft Actor-Critic, even though Continuous BDPI uses much smaller neural networks, and has not enjoyed hyper-parameter tuning. We stress the importance of this result, as current research with continuous actions usually leads to much smaller increases in sample-efficiency of about 10%. Moreover, this visionary paper proposes early results, with no tuning to Continuous BDPI yet, and still largely outperforms SAC. Because Continuous BDPI is built on so many original components, identifying the importance of hyper-parameters will require investigation.

We also hope that the work we present in this paper will attract interest on algorithms with off-policy critics, and an actor that does not depend on the Policy Gradient loss, which Continuous BDPI is an example of. We hope that future research will further push sample-efficiency, to allow complex continuous control problems to leverage Reinforcement Learning, and its ability to automatically learn state representations and learn long-horizon policies.

ACKNOWLEDGMENTS

This research received funding from the Flemish Government (Belgium), the first author being sponsored by the Flemish AI Research Program. The second author is funded by the Science Foundation of Flanders (FWO, Belgium), with Applied Research grant number 1SA6619N.

REFERENCES

[1] Jinwon An and Sungzoon Cho. 2015. Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE* 2, 1 (2015), 1–18.

[2] Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt Konolige, et al. 2018. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 4243–4250.

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. arXiv:arXiv:1606.01540

[4] Scott Fujimoto, Herke Van Hoof, and David Meger. 2018. Addressing Function Approximation Error in Actor-Critic Methods. In *International Conference on Machine Learning (ICML)*. 1582–1591.

[5] Scott Fujimoto, David Meger, and Doina Precup. 2019. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*. 2052–2062.

[6] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. *Advances in neural information processing systems* 27 (2014), 2672–2680.

[7] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv* abs/1801.01290 (2018).

[8] Janne Karttunen, Anssi Kanervisto, Ville Hautamäki, and Ville Kyrki. 2019. From Video Game to Real Robot: The Transfer between Action Spaces. *arXiv* abs/1905.00741 (2019).

[9] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[10] Mykel J. Kochenderfer and Tim A. Wheeler. 2019. *Algorithms for Optimization*. MIT Press.

[11] Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. 2019. Stabilizing off-policy q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems*. 11784–11794.

[12] Nevena Lazic, Craig Boutilier, Tyler Lu, Eehern Wong, Binz Roy, MK Ryu, and Greg Imwalle. 2018. Data center cooling using model-predictive control. In *Advances in Neural Information Processing Systems (NIPS)*. 3814–3823.

[13] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. 2020. Learning quadrupedal locomotion over challenging terrain. *Science robotics* 5, 47 (2020).

[14] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. 2016. End-to-End Training of Deep Visuomotor Policies. *Journal of Machine Learning Research* 17 (2016), 39:1–39:40.

[15] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. 2018. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research* 37, 4-5 (2018), 421–436.

[16] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous Methods for Deep Reinforcement Learning. In *International Conference on Machine Learning (ICML)*. 10.

[17] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei a Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533. https://doi.org/10.1038/nature14236 arXiv:1312.5602

[18] Rémi Munos, Thomas Stepleton, Anna Harutyunyan, and Marc G. Bellemare. 2016. Safe and efficient off-policy reinforcement learning. In *Neural Information Processing Systems (NIPS)*.

[19] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. 2016. Deep Exploration via Bootstrapped DQN. In *Advances in Neural Information Processing Systems (NIPS)*.

[20] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *Arxiv* abs/1707.06347 (2017). arXiv:1707.06347

[21] Randall S Sexton, Robert E Dorsey, and John D Johnson. 1998. Toward global optimization of neural networks: a comparison of the genetic algorithm and backpropagation. *Decision Support Systems* 22, 2 (1998), 171–185.

[22] Krzysztof Socha and Christian Blum. 2007. An ant colony optimization algorithm for continuous optimization: application to feed-forward neural network training. *Neural Computing and Applications* 16, 3 (2007), 235–247.

[23] Denis Steckelmacher, Hélène Plisnier, Diederik M Roijers, and Ann Nowé. 2019. Sample-Efficient Model-Free Reinforcement Learning with Off-Policy Critics. *arXiv* abs/1903.04193 (2019).

[24] Richard Sutton, D McAllester, Satinder Singh, and Yishay Mansour. 2000. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Neural Information Processing Systems (NIPS)*. 7.

[25] Joost Van Amersfoort, Lewis Smith, Yee Whye Teh, and Yarin Gal. 2020. Uncertainty estimation using a single deep deterministic neural network. In *International Conference on Machine Learning*. 9690–9700.

[26] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Rémi Munos, Koray Kavukcuoglu, and Nando de Freitas. 2016. *Sample Efficient Actor-Critic with Experience Replay*. Technical Report. 1–20 pages. arXiv:1611.01224

[27] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando de Freitas. 2016. *Dueling Network Architectures for Deep Reinforcement Learning*. Technical Report. 1–16 pages. https://doi.org/10.1109/MCOM.2016.7378425 arXiv:1511.06581

[28] Yuhuai Wu, Elman Mansimov, Roger B Grosse, Shun Liao, and Jimmy Ba. 2017. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in neural information processing systems (NIPS)*. 5279–5288.

[29] Yifan Wu, George Tucker, and Ofir Nachum. 2019. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361* (2019).

[30] Zhaoming Xie, Patrick Clary, Jeremy Dao, Pedro Morais, Jonathan Hurst, and Michiel van de Panne. 2019. Iterative Reinforcement Learning Based Design of Dynamic Locomotion Skills for Cassie. *arXiv* abs/1903.09537 (2019).