A Demon Control Architecture with Off-Policy Learning and Flexible Behavior Policy

Shangtong Zhang
Dept. of Computing Science
University of Alberta
shangtong.zhang@ualberta.ca

Richard S. Sutton
Dept. of Computing Science
University of Alberta
rsutton@ualberta.ca

Abstract

Auxiliary tasks can help learn a better feature representation. The tasks are often learned simultaneously via off-policy learning. However, state of the art techniques usually derive the behavior policy from one fixed task. In this paper, we propose an architecture combining flexible behavior policy, multi-task learning and off-policy methods. We showcase our proposed architecture can achieve faster learning speed and learn a better feature representation.

1 Introduction

In complex environment, a good feature representation is crucial to learning a good policy. Recently multi-task learning via off-policy methods is shown to be able to help learn better feature representations. However, state of the art techniques, e.g. Cabi et al. (2017) and Jaderberg et al. (2016), often derive the behavior policy from one fixed task. In this paper, we explore the possibility to have a flexible behavior policy, i.e. deriving the behavior policy from different tasks at different time steps, which is similar to the option framework (Sutton et al. 1999). In the meantime, we also integrate the multi-task learning and off-policy methods. We adopt the one-step option execution model (Sutton et al. 1999), which is actually a polling execution. Thus we refer our policy over options as the polling policy, and we call our architecture the Demon Control Architecture (DCA). Further more, we showcase DCA achieves faster learning speed and learns a better feature representation.

2 Related Work

Flexible behavior policy, multi-task learning and off-policy methods are three important components of DCA, and we are the first to combine all the three together.

Many previous work derived the behavior policy from one fixed task. Horde architecture was introduced by Sutton et al. (2011) for knowledge representation. Each *demon* in Horde was associated with an auxiliary task and had corresponding pseudo reward. It was trained to learn the general value function (GVF). The behavior policy was derived from the main task. Hybrid Reward Architecture (HRA, van Seijen et al. 2017) extended Horde to control problems by decomposing the reward function into sub-reward functions. The behavior policy was derived via linear combination of the learned Q-values under the sub-reward functions and pseudo reward functions. UNsupervised REinforcement and Auxiliary Learning (UNREAL) agent (Jaderberg et al. 2016) was the first paradigm applying unsupervised auxiliary tasks, where the behavior policy was always derived from the main task. The main feature of the unsupervised auxiliary task is that it does not involve the human-designed pseudo reward. Auxiliary tasks in Horde, in contrast, are mainly derived from human-designed pseudo rewards. Following the UNREAL agent, Cabi et al. (2017) proposed the Intentional Unintentional (IU) Agent, where the behavior policy was always from the hardest task.

Hierarchical Reinforcement Learning Workshop at the 31st Conference on Neural Information Processing Systems (HRL@NIPS 2017), Long Beach, CA, USA.

Some previous work did not involve the off-policy learning. Option framework actually had flexible behavior policy, it committed to different intra-option policies at different time steps. With each option as a task, it also involved multi-task learning. However, its multi-task learning is sequentially rather than simultaneously. At any time step, only one intra-option policy was trained. Off-policy learning, which can significantly boost the training progress, was not included. Although there has been recently huge progress in option discovery, e.g Bacon et al. (2017) and Machado et al. (2017), the option framework still suffers from the lack of off-policy learning. The main difference between our work and the Option-Critic Architecture (OCA, Bacon et al. 2017) is that we involve the off-policy learning. Moreover, we eliminate the need for the termination function via the one-step option execution model, resulting in a simpler update rule without losing expressivity.

3 Algorithm

Considering a discounted and episodic setting with a discount ratio γ and a *Markov Decision Process* with a state set $\mathcal{S}$, an action set $\mathcal{A}$, a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and a transition function $p : \mathcal{S}\times\mathcal{A} \to (\mathcal{S} \to [0, 1])$, we introduce the *control demon*. Similarly to the demon in Horde, each control demon $o \in \mathcal{O}$ is associated with a pseudo reward function $r_o : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and has its demon policy $\pi_o : \mathcal{S}\times\mathcal{A} \to [0, 1]$, where $\mathcal{O} = \{o_1, \dots, o_n\}$ is the set of all the control demons. Each demon policy π_o has corresponding demon value function $v_{\pi_o}(s) = \mathbb{E}_{\pi_0}[\sum_{k=t+1}^{T} \gamma^{k-t-1} R_{o,k} \mid S_t = s]$, where T is a random variable denoting the time entering a corresponding pseudo terminal state following π_o and $R_{o,k}$ is a random variable denoting the pseudo reward for the demon o at time k. Considering $\mu : \mathcal{S}\times\mathcal{O} \to [0, 1]$ the polling policy, its value function is $v_\mu(s) = \mathbb{E}_\mu[\sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \mid S_t = s]$, where T is a random variable denoting the time entering a terminal state following μ and R_k is a random variable denoting the reward from the environment at time k. We can flatten μ into a policy $b : \mathcal{S} \times \mathcal{A} \to [0, 1]$ over actions, i.e. for $s \in \mathcal{S}$ and $a \in \mathcal{A}$, $b(a|s) = \sum_{o\in\mathcal{O}} \mu(o|s)\pi_o(a|s)$. Actually b is the behavior policy which can be interpreted as a linear combination of the demon policies similarly to the linear combination of the GVFs in HRA. In DCA, however, the weight $\mu(o|s)$ is learned rather than pre-defined like HRA. Staring from S_t, the agent selects a demon $O_t \sim \mu$ and an action $A_t \sim \pi_{O_t}$, after which the environment gives reward R_{t+1} and leads the agent to S_{t+1}.

Assuming π_o is parameterized by θ_o and v_{π_o} is parameterized by ϕ_o, the goal is to maximize $J(\theta_o) = \sum_{s\in\mathcal{S}} d^b(s) v_{\pi_o}(s)$, where $d^b(s)$ is the limiting distribution of states under b. According to the off-policy actor-critic algorithm (Degris et al. 2012) with $\lambda = 0$,

$$\nabla J(\theta_o) \approx \mathbb{E}_b[\rho_o(S_t, A_t)\nabla \log \pi_o(A_t|S_t)(R_{o,t+1} + \gamma v_{\pi_o}(S_{t+1}) - v_{\pi_o}(S_t))]$$

where $\rho_o(S_t, A_t) = \frac{\pi_o(A_t|S_t)}{b(A_t|S_t)}$ is the importance sampling ratio. Thus our update rules for θ_o and ϕ_o are:

$$\theta_o \leftarrow \theta_o + \alpha\rho_o(S_t, A_t)\nabla \log \pi_o(A_t|S_t)(R_{o,t+1} + \gamma v_{\pi_o}(S_{t+1}) - v_{\pi_o}(S_t))$$
$$\phi_o \leftarrow \phi_o + \alpha(R_{o,t+1} + \gamma v_{\pi_o}(S_{t+1}) - v_{\pi_o}(S_t))\nabla v_{\pi_o}(S_t)$$

where α is the shared step size for updating all the demon policies.

Let the polling policy μ and its value function v_μ be parameterized by θ_μ and ϕ_μ. With all the demon policies as a part of the environment, the goal therefore is to maximize $J(\theta_\mu) = \sum_{s\in\mathcal{S}} d^\mu(s) v_\mu(s)$, where $d^\mu(s)$ is the limiting distribution of states under μ. Following the actor-critic algorithm,

$$\nabla J(\theta_\mu) = \mathbb{E}_\mu[\nabla \log \mu(O_t|S_t)(R_{t+1} + \gamma v_\mu(S_{t+1}) - v_\mu(S_t))]$$

Thus, our update rules for θ_μ and ϕ_μ are

$$\theta_\mu \leftarrow \theta_\mu + \beta\nabla \log \mu(O_t|S_t)(R_{t+1} + \gamma v_\mu(S_{t+1}) - v_\mu(S_t))$$
$$\phi_\mu \leftarrow \phi_\mu + \beta(R_{t+1} + \gamma v_\mu(S_{t+1}) - v_\mu(S_t))\nabla v_\mu(S_t)$$

where β is the step size.

We follow the training protocol of the Asynchronous Advantage Actor-Critic paradigm (A3C, Mnih et al. 2016), where multiple workers are involved to generate uncorrelated on-policy data. In this instantiation, we also adopted n-step return to train the critic. The whole algorithm is formalized in **Appendix A**.

For simplicity, in the following sections we refer the control demon as the demon.

4 Experiments

4.1 Fruit Collection Domain

The fruit collection task was first introduced by van Seijen et al. (2017). It is actually an instance of the traveling salesman problem, which is known to be NP-complete. We also see similar tasks in Jaderberg et al. (2016), however they used image input while we encoded the information into a vector for simplicity. The agent has to collect fruits as quickly as possible in a 10×10 grid. There are 10 possible fruit locations, spread out across the grid. For each episode, a fruit is randomly placed on 5 of those 10 locations. The agent starts at a random position and the episode ends after 100 steps or when all 5 fruits have been eaten, whichever comes first. Reward is 0 except for reaching a fruit, where reward is $+1$. The agent has 4 possible actions and if bumping into the wall, it will just stay in same location. [1]

Our goal is to answer the following questions:

- Does DCA outperform baseline algorithms, e.g. primitive actor-critic method auxiliary tasks?
- Does DCA really learn a good feature representation?

Experiment Setup. We parameterized μ, v_μ, $\{\pi_{o_i}\}$, $\{v_{\pi_{o_i}}\}$ by single $ReLU$ (Nair and Hinton 2010) hidden layer neural networks with 250 hidden units. μ and v_μ shared a common hidden layer. All demon policies $\{\pi_{o_i}\}$ and demon value functions $\{v_{\pi_{o_i}}\}$ also shared a common hidden layer. The network had a binary input layer of length 98, encoding the location of the agent, the locations of the fruits and whether there was a fruit on each location. Although part of our encoded state vector is constant, it is actually useful information for solving the task and we expect the agent to make use of it. We had 10 demons, with each demon representing a solution about how to reach a possible fruit location. The pseudo reward for the demon is 0 except the agent reaches corresponding fruit location, where the pseudo reward is $+1$ no matter whether there is a fruit or not, and this state is set to be a pseudo terminal state for that demon. We set the discount ratio $\gamma = 0.95$ and the architecture of the networks is elaborated in **Figure 1**.

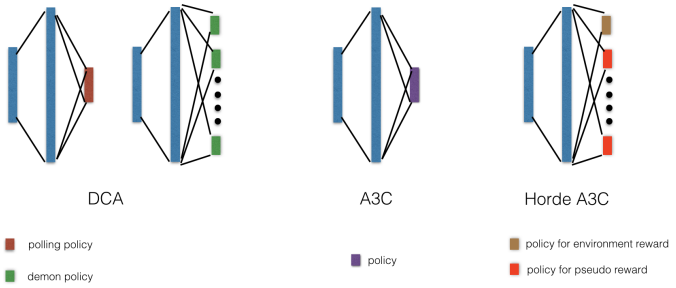
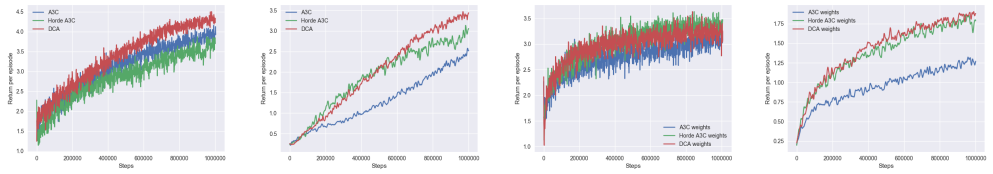


Figure 1: Architectures for the algorithms. Only policy functions are shown. Each policy has its corresponding value function, which is not displayed.

The baseline algorithms are the primitive A3C algorithm and the A3C algorithm with auxiliary tasks (referred to as Horde A3C). In Horde A3C, we had 11 demons for the 10 pseudo rewards and the original environment reward, where we always derived the behavior policy from the demon associated with the original reward. We trained all the demons simultaneously via the off-policy actor-critic method. The architectures of A3C and Horde A3C are elaborated in **Figure 1**. We compared different optimizers, e.g. *SGD* (Bottou 2010) with $momentum = 0.9$, *Adam* (Kingma and Ba 2014) and *RMSProp* (Tieleman and Hinton 2012), and found *SGD* gave best performance. We applied same learning rate for updating policy function and value function. We selected the best learning rate for each algorithm from $\{2^{-1}, \ldots, 2^{-15}\}$, and we found $2^{-12}$was the best for all the three algorithms. Alongside the training workers, we added an extra test worker to evaluate current learned policy.

[1]The task description is adapted from van Seijen et al. (2017).



(a) (b) (c) (d)

Figure 2: All the curves are averaged over 30 independent runs, and different runs have different random possible fruit locations. (a) training progression (b) test progression (c) training progression for A3C agents with fixed bottom layers (d) test progression for A3C agents with fixed bottom layers. A3C, Horde A3C and DCA are plotted in blue, green and red.

The test worker continuously conducted 30 deterministic episodes and used the mean return as the metric. Note that in the fruit collection domain the optimal policy can simply be deterministic. But in practice a stochastic policy often performs better than a deterministic policy. Assume the agent is near the wall and will bump into the wall following the current policy. If the policy is deterministic, the agent will just get stuck there until timeout. If the policy is stochastic, however, the agent can still be able to get out of the dilemma. In this sense, the test process is more difficult that the training process. We observed that DCA can significantly outperform the baseline algorithsm in both training time and test time. The curves for training and test progression are shown in **Figure 2a** and **Figure 2b**. The training curve was plotted with the statistics from a randomly selected training worker and the test curve was from the test worker. All the curves were average over 30 independent runs. As our x-axis is steps, which can not be aligned among multiple runs by nature, we adopted the linear interpolation to align the statistics.

DCA actually learned better feature representations. We showcased this by the following experiment. First we trained three agents with DCA, A3C and Horde A3C until convergence. In neural networks, the bottom layers are usually responsible for learning the feature representation. So we stored the weights of the bottom layers of the converged networks. Then we started three new A3C agents, and their bottom layers were initialized with those stored weights respectively. When training the three new agents, we froze the the bottom layers. So actually only the weights of the last layers were learned in the new agents and the first hidden layers remained unchanged as initialized. In this setting, the agents initialized with weights from DCA and Horde A3C learned significantly faster than the agent with weights from primitive A3C, demonstrating that auxiliary tasks do help find a better feature representation. The curves for training and test progression are shown in **Figure 2c** and **Figure 2d**.

5 Discussion

DCA is the first framework combining flexible behavior policy, multi-task learning and off-policy methods. We showcase that DCA can accelerate learning process and gain a better feature representation. However, until now we only investigated cooperative demons, future work will be evaluating DCA with competitive demons. Moreover, DCA now is limited to pseudo rewards, which are more like supervised tasks compared with the UNREAL agent. Future work will involve unsupervised tasks in DCA.

6 Acknowledgment

The authors thank Kristopher De Asis, Huizhen Yu and anonymous reviewers for their insightful comments.

References

Bacon, P.-L., Harb, J., and Precup, D. (2017). The option-critic architecture. In *AAAI*, pages 1726–1734.

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer.

Cabi, S., Colmenarejo, S. G., Hoffman, M. W., Denil, M., Wang, Z., and de Freitas, N. (2017). The intentional unintentional agent: Learning to solve many continuous control tasks simultaneously. *arXiv preprint arXiv:1707.03300*.

Degris, T., White, M., and Sutton, R. S. (2012). Off-policy actor-critic. *arXiv preprint arXiv:1205.4839*.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. (2016). Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*.

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

Machado, M. C., Bellemare, M. G., and Bowling, M. (2017). A laplacian framework for option discovery in reinforcement learning. *arXiv preprint arXiv:1703.00956*.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., and Precup, D. (2011). Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 761–768. International Foundation for Autonomous Agents and Multiagent Systems.

Sutton, R. S., Precup, D., and Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31.

van Seijen, H., Fatemi, M., Romoff, J., Laroche, R., Barnes, T., and Tsang, J. (2017). Hybrid reward architecture for reinforcement learning. *arXiv preprint arXiv:1706.04208*.

A

Algorithm 1: An episode of a DCA worker

Input:
n: number of demons
l: maximum rollout length
α: learning rate for demon policies
β: learning rate for the polling policy
r_o for all $o \in \mathcal{O}$: pseudo reward functions
$\theta_\mu^-, \phi_\mu^-, \{\theta_{o_i}^-\}_{i=1\ldots n}, \{\phi_{o_i}^-\}_{i=1\ldots n}$: global shared parameters

Initialize step counter $t \leftarrow 0$
Get initial state s_0 from environment
while *True* **do**
  Sync worker-specific parameters $\theta_\mu, \phi_\mu, \{\theta_{o_i}\}_{i=1\ldots n}, \{\phi_{o_i}\}_{i=1\ldots n}$ with the global shared parameters. For clarity, we refer $\theta_{o_i}^-, \phi_{o_i}^-, \theta_{o_i}, \phi_{o_i}$ as $\theta_i^-, \phi_i^-, \theta_i, \phi_i$
  Reset gradients $d\theta_\mu, d\phi_\mu, \{d\theta_i\}, \{d\phi_i\}$ to 0
  repeat
    Select a demon $o_t \sim \mu(\cdot\,|s_t, \theta_\mu)$
    Select an action $a_t \sim \pi_{o_t}(\cdot\,|s_t, \theta_{o_t})$
    Execute a_t, get reward r_t, pseudo rewards $r_{t,1}, \ldots, r_{t,n}$ and next state s_{t+1}
    $t \leftarrow t + 1$
  until *terminal* s_t *or* $t - t_{start} == l$
  if *terminal* s_t **then** $G \leftarrow 0$
  else $G \leftarrow v_\mu(s_t|\phi_\mu)$
  for $i = 1, \ldots, n$ **do**
    if s_t *is terminal w.r.t.* o_i **then** $G_i \leftarrow 0$
    else $G_i \leftarrow v_{\pi_{o_i}}(s_t|\phi_i)$
  end
  for $k = t-1, \ldots, t_{start}$ **do**
    $G \leftarrow r_k + \gamma G$
    $d\theta_\mu \leftarrow d\theta_\mu + \beta \nabla \log \mu(o_k|s_k, \theta_\mu)(G - v_\mu(s_k|\phi_\mu))$
    $d\phi_\mu \leftarrow d\phi_\mu + \beta(G - v_\mu(s_k|\phi_\mu))\nabla v_\mu(s_k|\phi_\mu)$
    $b_k = flatten(o_k)$
    for $i = 1, \ldots, n$ **do**
      $G_i \leftarrow r_{k,i} + \gamma G_i$
      $d\theta_i \leftarrow d\theta_i + \alpha \frac{\pi_{o_i}(a_k|s_k,\theta_i)}{b_k(a_k|s_k)} \nabla \log \pi_{o_i}(a_k|s_k, \theta_i)(G_i - v_{\pi_{o_i}}(s_k|\phi_i))$
      $d\phi_i \leftarrow d\phi_i + \alpha(G_i - v_{\pi_{o_i}}(s_k|\phi_i))\nabla v_{\pi_{o_i}}(s_k|\phi_i)$
    end
  end
  Perform asynchronous update for $\theta_\mu^-, \phi_\mu^-, \{\theta_i^-\}_{i=1\ldots n}, \{\phi_i^-\}_{i=1\ldots n}$ with $d\theta_\mu, d\phi_\mu, \{d\theta_i\}_{i=1\ldots n}, \{d\phi_i\}_{i=1\ldots n}$
  if *terminal* s_t **then**
    break
  end
end