Developmental Reinforcement Learning through Sensorimotor Space Enlargement

Matthieu Zimmer, Yann Boniface, Alain Dutech

Developmental Reinforcement Learning through Sensorimotor Space Enlargement

Matthieu Zimmer*, Yann Boniface* and Alain Dutech†
*University of Lorraine, LORIA, UMR 7503, Nancy, France
†INRIA, LORIA, UMR 7503, Nancy, France
{firstname}.{lastname}@loria.fr

***Abstract*—In the framework of model-free deep reinforcement learning with continuous sensorimotor space, we propose a new type of transfer learning, inspired by the child development, where the sensorimotor space of an agent grows while it is learning a policy. To decide how the dimensions grow in our neural network based actor-critic, we add new developmental layers to the neural networks which progressively uncover some dimensions of the sensorimotor space following an Intrinsic Motivation heuristic. To mitigate the catastrophic forgetting problem, we take inspiration from the Elastic Weight Constraint to regulate the learning of the neural controller. We validate our approach using two state-of-the-art algorithms (*DDPG* and *NFAC*) on two high-dimensional environment benchmarks (*Half-Cheetah* and *Humanoid*). We show that searching first for a suboptimal solution in a subset of the parameter space, and then in the full space, is helpful to bootstrap learning algorithms, and thus reach better performances in fewer episodes.**

I. Introduction

Reinforcement learning (RL) is a framework for framing sequential decision problems, in which an agent interacts with its environment and adapts its policy based on a scalar reward signal [1]. RL agents can autonomously learn difficult tasks, like playing video games [2]. While RL techniques for discrete problems are well studied, fully continuous environments for both state and action spaces still need new algorithms to address more complex real-world problems, as well as problems where the reward is sparse or delayed [3].

State of the art algorithms for continuous settings or large discrete spaces rely on artificial neural networks because they show good generalization and scalability properties. However, in a very high-dimensional space, searching from scratch is difficult, and generally inefficient. A successful research direction consist in using task of increasing difficulty to facilitate learning, as done, for example, in curriculum learning [4]. We follow that trend by proposing an original developmental RL approach where the sensorimotor space of the agent is enlarged while the agent learns its task.

Thus, our approach consists of searching first in a smaller space to find a local optimum, then increasing the search space by adding new perception and action dimensions. This is a type of transfer learning [5], compatible with curriculum learning, where an agent learns a source task and transfers its knowledge to a target task. Since the state and action spaces grow over time, agents are able to gradually increase the difficulty of their task as in developmental robotics [6]. Moreover, as stated by Guerin [7], learning basic physical concepts is crucial in a developmental approach which is more natural in a fully continuous environment where no discrete symbol is infused. For those reasons, we categorize our work into developmental reinforcement learning.

We test this approach using two different state-of-the-art model-free RL algorithms. We show that our approach could lead to higher policy quality and increases the learning speed on two high-dimensional environments with continuous sensorimotor space. Unlike Atari games [8] for discrete action RL, benchmarking reinforcement learning with continuous actions in high-dimensional sensorimotor space is difficult because of the lack of free and open simulators for those domains[1]. Inspired by the models proposed within OpenAI Gym [9], we developed two open-source environments within Open Dynamic Engine (ODE) [10] trying to reproduce them from Mujoco [11]: Half-Cheetah and Humanoid.

II. Background

We are interested in RL problems, modeled as *Markov Decision Processes* (MDP) $\langle S, A, T, R\rangle$ where S is the state space, A is the action space, T is the transition function, and R is the reward function. We are especially interested in MDP where the state space and action space are continuous. The goal of RL is to seek for an *optimal* policy π^* (a mapping from S to A) maximizing the expected discounted cumulative reward:

$$\pi^* \in \arg\max_{\pi} J(\pi) = \arg\max_{\pi} \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t r_t | \pi\Big], \qquad (1)$$

where r_t denotes the reward at time step t and $0 \leq \gamma < 1$ is the discount factor.

When the state space S and action space A are continuous, the use of an *actor* (*i.e.* a parametric policy) becomes crucial to approximate the policy and to overcome the complexity of the argmax search to select an action. This often leads to *actor-only* methods or model-free policy search [12] like REINFORCE [13] or Covariance Matrix Adaptation Evolution

[1]The free license of Mujoco is not sufficient to perform an optimization of the hyperparameters on a cluster which is almost always required for deep actor-critic algorithms.

Strategy (CMA-ES) [14]. The major drawbacks of *actor-only* methods are the high variability of the sampling of the objective J and their low data efficiency. On the other hand, *actor-critic* algorithms [15] try to mitigate those drawbacks with a *critic*. The *critic* learns a value function thus reducing the variability of the sampling of J by estimating $Q : S \times A \to \mathbb{R}$, the sequential values of actions in each state: (or $V : S \to \mathbb{R}$ the sequential value of each state) :

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\Big[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \Big| s_t = s, a_t = a\Big], \qquad (2)$$

where r_t is the reward obtained at time t from R following π.

We now present two state-of-the-art *actor-critic* algorithms that we will use. To focus on the contributions of our developmental approach, these algorithms will be used to learn deterministic policies as it is difficult to clearly evaluate algorithms learning stochastic policies [16].

Deep Deterministic Policy Gradient (DDPG) is an *off-policy*, *online* algorithm [17]. It learns the Q value function, and it reuses previous samples through its *off-policy* update with experience replay. Based on Neural Fitted Q with Continuous Actions [18], DDPG is more scalable due to online updates, target networks [2] and batch normalization [19]. The target networks serve to slow down the weight updates to increase the stability of learning, by soft updating a copy of the policy and the value function. The actor is updated according to the backpropagation of the gradient of the critic.

Neural Fitted Actor Critic (NFAC) is an *on-policy*, *offline* algorithm [20]. It splits reinforcement learning problems into two distinct supervised learning problems. The critic update relies on Fitted-Q updates [21] applied to a V function. The actor update relies on Continuous Actor Critic Learning Automaton (CACLA) algorithm [22] that reinforces actions according to Temporal Difference errors. The samples are forgotten after each episode because the actor features *on-policy* update. In this work, we use the extended version of NFAC with batch normalization and eligibility traces [23].

III. Sensorimotor Space Enlargement Without Environment Modification

The main objective behind our idea of developmental RL is to progressively increase the dimensionality of the tackled problem. Like curriculum learning [4], we suppose that learning with gradual difficulty is helpful to explore the search space in complex continuous environments.

With MDPs, the growth in problem dimensionality translates to growing the sensorimotor space of the agent. In this work, we address the more specific problem of the sensorimotor space enlarging without modifications to the environment. Modifications are considered as being "internal" to the agent. At first, the agent learns to solve the problem with fewer perceptions and controls : this is the *source* task. Then comes a *transfer moment* where the agent increases its sensorimotor space, thus changing tasks. In this work, the time unit for a transfer moment is the episode. As such, a transfer never occurs online during one episode, but between two episodes (offline). After the transfer, the agent is now solving the *target* task with full perceptions and controls. Recall that for $s_t, s_{t+1} \in S^2, a_t \in A, r_{t+1} \in \mathbb{R}$, the system is governed by the following equations :

$$\begin{cases} a_t & \sim \pi(\cdot|s_t), \\ s_{t+1} & \sim T(\cdot|s_t, a_t), \\ r_{t+1} & \sim R(s_t, a_t, s_{t+1}). \end{cases}$$

To simulate the sensorimotor space increase, we decompose $\pi(a_t|s_t)$ into two parts: $\pi_t^+({a_t}^+|s_t) \cup \pi_t^-({a_t}^-|h(s_t))$ where ${a_t}^+ \in A^+$, ${a_t}^- \in A^-$ such that $A^+ \cup A^- = A$, $A^+ \cap A^- = \emptyset$ and $a_t = {a_t}^+ \cup {a_t}^-$. While π^- is learned by the agent to solve the source task, the functions π^+ and h are provided by the designer. The policy π^+ controls the dimensions that are not yet controlled by the agent. The function $h : S \to S^- \subseteq S$ is used to reduce the perception over the state space. Thus, the policy π^- learned by the agent is defined in a smaller space $S^- \times A^- \subseteq S \times A$. Once π^- is learned, it is used to learn the full policy π without decomposition in the full sensorimotor space. An example is given in Figure 1. Instead of having only one transfer, such decomposition can be easily generalized to more numerous moments.

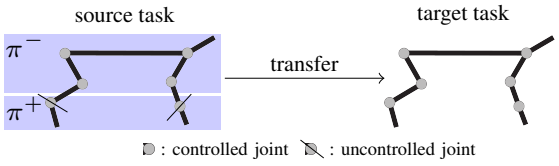

Fig. 1. The Half-Cheetah body is composed of several degrees of freedom to learn how to run. In this example, S^- is equivalent to S minus the perception over the ankles, π^+ controls the ankles (for instance by applying a constant torque) and π^- controls and perceives the rest of the body.

A crucial question concerns the order in which the sensorimotor space is incremented. In this work, this question is left unanswered and considered as *a priori* knowledge. We concentrate on the design of developmental reinforcement learning for neural control architectures with *actor-critic* methods and two other questions : when to transfer and how to transfer.

IV. Developmental Layer

We focus on an approach where we seek for generality and try to minimize modifications of the environment and actor-critic learning algorithms by working mostly on the neural network architecture. We create a new neural layer filtering inputs that we called developmental layer (DL). Similar to a dropout layer [24], each input is connected to one output where the associated weight control if the information passes (see Figure 2).

Given an input vector x and parameters ϑ of the DL, the deterministic activation function to compute the y output is :

$$y_i = \begin{cases} x_i & \text{if } \vartheta_i \geq \text{threshold}, \\ 0 & \text{otherwise.} \end{cases}$$

Placing DL at strategic locations of the neural network architecture, as shown in Figure 3, makes the transfer more

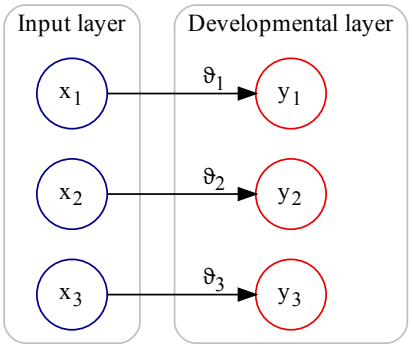

Fig. 2. The developmental layer (DL) controls which input x will be forward to the next layer through ϑ.

transparent for the designer. With *actor-critic* methods, a DL is added at the beginning of the policy and the value function to hide some dimensions of the state according to h. It simulates the reduction of S to S^-. Another DL is added at the end of the policy to disable control over some dimension according to π^-. It simulates the reduction of A to A^-.

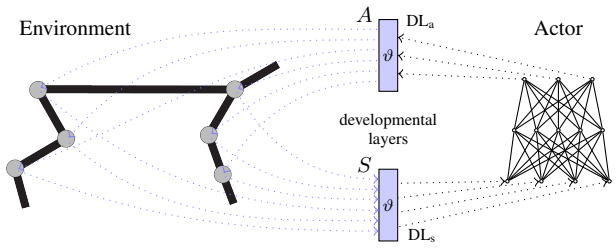

Fig. 3. The lower developmental layer DL_s is used to filter dimensions of the state and the higher developmental layer DL_a to filter some action dimensions. As the activation of disabled neurons will be 0, the corresponding weights (within the inner product) will not change during backpropagation, so the search space is smaller. Thereby, it is not necessary to explicitly define a neural network over A^- and S^- when using DL.

With such architecture, the learning algorithm needs only a very slight modification when it calls π to take into account the non-controlled dimensions of the action. Because the DL will set disabled dimensions of the action to zero, it is necessary to replace them according to π^+ during the training on the source task.

A. How to transfer

Solutions found in the source task should be exploited at the best in the target task. However, it is not straightforward with neural networks because of the catastrophic forgetting problem [25]. In this paper, we are going to compare two methods to perform such a transfer. In the first method, the solution found in the sub-space is only used as a starting point in the full space. In the general case (without DL), the number of parameters between the source task and the target should be different. Suppose that $\theta^{source} \in \mathbb{R}^k$ are the parameters found once the first optimization is done and $\theta^{target} \in \mathbb{R}^K$ are the parameters for the full problem. When using DL, the number of parameters of neural networks (without taking into account DL parameters ϑ) does not change ($k = K$) but a part of θ^{source} is not used. This part is composed of weights linked to input and output neurons disabled by DL, and these weights thus keep their initial random value. The method we call "initialization" will start learning the target task with the same weights θ^{source}.

The second method, called Elastic Weight Constraint (EWC), use the same initialization but will limit the update of important weights, using the Fisher Information Matrix (FIM) to decide on the importance of weights [26]. A regularization constraint $\mathcal{R}(\theta)$ is added to the cost function (denoted $\mathcal{L}(\theta)$) optimized by the network, leading to a new cost $\mathcal{L}_{final}(\theta)$:

$$\mathcal{L}_{final}(\theta) = \mathcal{L}(\theta) + \beta\eta^{t-t_0}\mathcal{R}(\theta),$$

where β controls the importance of the constraint, η is a decay parameter to reduce the importance of the constraint over time, t is the current episode, t_0 is the episode where a transfer occurred last and $\mathcal{R}(\theta) = \sum_{i=1}^{K} F_i(\theta_i - \theta_i^{source})$ with F being the FIM. Fisher weights of disabled dimensions will be zero because of the backpropagation algorithm properties. This constraint is added only to the actor because a good critic should evaluate the current actor accurately, and for that he has to adapt quickly.

B. When to transfer?

A key question is to decide *when* to enlarge the sensorimotor space (through changing the weights ϑ of the developmental layer). If the transfer occurs too lately, then, in most problems, there will be no gain at using DL because all the time spent would be enough to solve the task without DL. On the other hand, if the transfer occurs too soon, the "boostrap effect" will be negligible as the agent will not have the time to extract interesting solutions from the source task.

Accurately answering this question *a priori* seems impossible and highly task dependent. In this work, we use a heuristic solution inspired by *intrinsic motivation* [27]. When the increase in performance stagnates during the source task, the agent triggers a transfer by modifying the weights of developmental layers according to a given development plan. Formally, the agent triggers a transfer when :

$$\sum_{i=0}^{sm} \hat{J}(\pi_{t-i}) - \sum_{i=0}^{sm} \hat{J}(\pi_{t-i-wi}) \leq 0,$$

where t is the current episode, $\hat{J}(\pi)$ is an evaluation of the policy π, sm is a smoothing parameter and wi is a time window parameter. Depending on the task and the actor-critic algorithm, those parameters should be tuned carefully. Moreover, actor-critic algorithms rely on exploration during the training phase, so the agent can only access a sampling evaluation $\hat{J}$ of the true performance J.

V. Experimental Setup

The experiments are designed to show that adding our developmental layer (DL) to the neural network architecture of DDPG and NFAC could increase the learning speed or the policy quality on two high-dimensional environments.

A. Environments

The experiments are performed on environments with continuous state and action spaces in a time-discretized simulation. The first environment is Half-Cheetah (6 degrees of freedom) [28] and the second is Humanoid (17 degrees of freedom) [29].

In Half-Cheetah, the reward function is $R(s,a) = v_x(s) - 0.1 \cdot \|a\|_2^2 - 1000 \cdot g(s)$ where $v_x(s)$ is the speed of the cheetah on x axis and $g(s)$ is 1 if the head touches the ground and 0 otherwise. Absorbing states correspond to $S^* = \{s \in S \mid g(s) = 1\}$. The cheetah must move as fast as possible interacting with an immobile ground in a 2D plane while being subjected to realistic physical constraints.

In Humanoid, the reward function is $R(s,a) = 3 + 5v_x(s) - 0.05 \cdot \|a\|_2^2$ where $v_x(s)$ is the speed of the humanoid on the x-axis. It also has to move forward but without falling. States become absorbing if the height of the torso is lower than 0.8 or higher than 1.7.

States are composed of the joint positions/angles and joint position/angle velocities. Dimensions of $S \times A$ are 18×6 (Half-Cheetah), and 45×17 (Humanoid). The full description (body, mass, friction, inertia, ...) and source code of those environments can be found at https://github.com/matthieu637/ddrl.

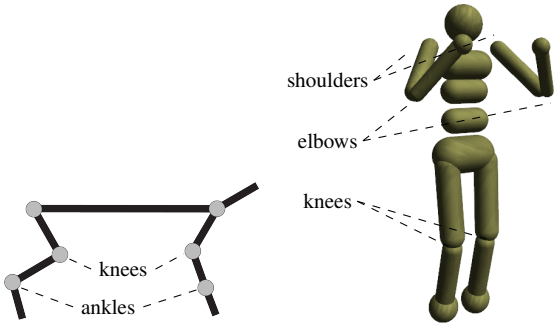

Fig. 4. Illustration of Half Cheetah (left) and Humanoid (right) environments. The labeled limbs will potentially be controlled by π^+ during a source task.

B. Grid Search

In order to select a good development plan (i.e. in what order the dimensions of the sensorimotor space are taken into account by switching their DL ϑ from 0 to 1), we performed an offline grid search over several possibilities. This is more a proof of concept than a general methodology that should be used. We follow a general guideline for the two environments: the agent cannot control a limb that it does not yet perceive. For instance, the action dimension of controlling a knee is never accessible before the respective state dimension.

For Half Cheetah, the selected limbs for the grid search are the ankles and the knees. An additional constraint links the two ankles and the two knees so they cannot be developed at a different time. For the humanoid, the selected limbs of the grid search are the elbows, shoulders and the knees.

C. Learning algorithms

For both actor-critic algorithms, the optimization method used to update the weights was Adam algorithm [30]. For NFAC, the exploration strategy is a truncated Gaussian law inside $[-1;1]$ with an independent variance $\sigma = 10^{-1}$. For DDPG, the exploration strategy is an Ornstein-Uhlenbeck process with $\sigma = 10^-3$, $\theta = 0.15$ and $dt = 10^-2$. During the testing phase (displayed in the following curves) this noise is disabled. The discount factor is fixed to $\gamma = 0.99$ for both algorithms. The source code of every algorithm and hyperparameter used can be found at http://drl.gforge.inria.fr/.

D. Neural networks representation

Policy and value function networks are composed of 2 hidden layers (inner product) of $50, 25$ units. Developmental layers are added in front (for the state) and back (for the action) of policy networks and in front of critic networks (see Figure 3). For the critic network of DDPG, actions and states are given as input at the same level. Each weight is initialized from a normal distribution $\mathcal{N}(0, 0.01)$. It is important to initialize the weights at a small value so when the transfer occurs the solution is not too much altered. The last layer of the critic and actor networks is linear. The developmental layers share parameters between critic and actor. Batch normalization is only applied to the first layer of the actor for both algorithms.

E. Results

For each experimental setup without DL, we optimize every hyperparameter through another grid search for each algorithm on each environment. Then we used the same hyperparameters when DL are added. Each experiment is been run 60 times for statistical results. In figures 5 and 6, only the best developmental plans are shown (selected according to the best median of J over testing phase). The policy π^+ that controls the disabled action dimension is constant and independent of the state. It simply applies a constant torque that doesn't disrupt too much the task.

In figures 5 and 6, we observe that, before any transfer, the performance of algorithms with DL is always better than the performance without DL. This validates our assumption that learning first in a smaller space simplifies ulterior learning. After the transfer, both algorithms are able to take advantage of the previous learned policies.

In figures 5 and 6, we see that EWC does not always lead to significant improvements. EWC requires optimizing several additional hyperparameters (β, η and a decay for FIM estimation). Besides, a proper way to choose the θ^{source} involved in the computation of important weights would require to periodically stop the exploratory behavior of the algorithm and run several (costly) simulations to identify the best current parameters of the actor.

VI. Discussion and related work

Planning a good sensorimotor growth is highly dependent on the problem to solve. This is where a designer can provide *a priori* knowledge by giving either constraints or the full

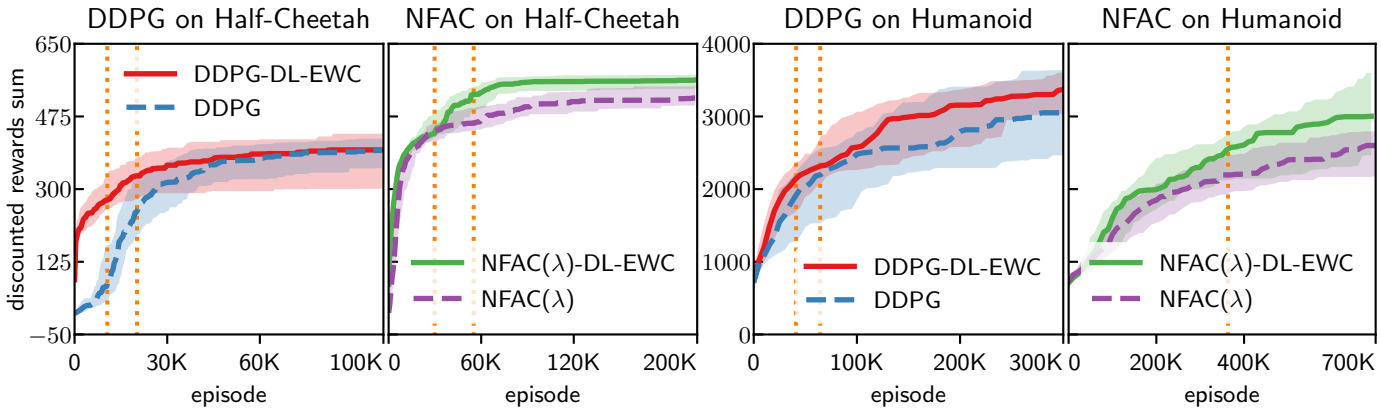

Fig. 5. Median and quartiles of the best registered performance during the testing phase with NFAC(λ) and DDPG on Half-Cheetah and Humanoid environments. When the DL are added, EWC is used to help the transfer. The averages of the transfer moments are displayed vertically (dashed orange).

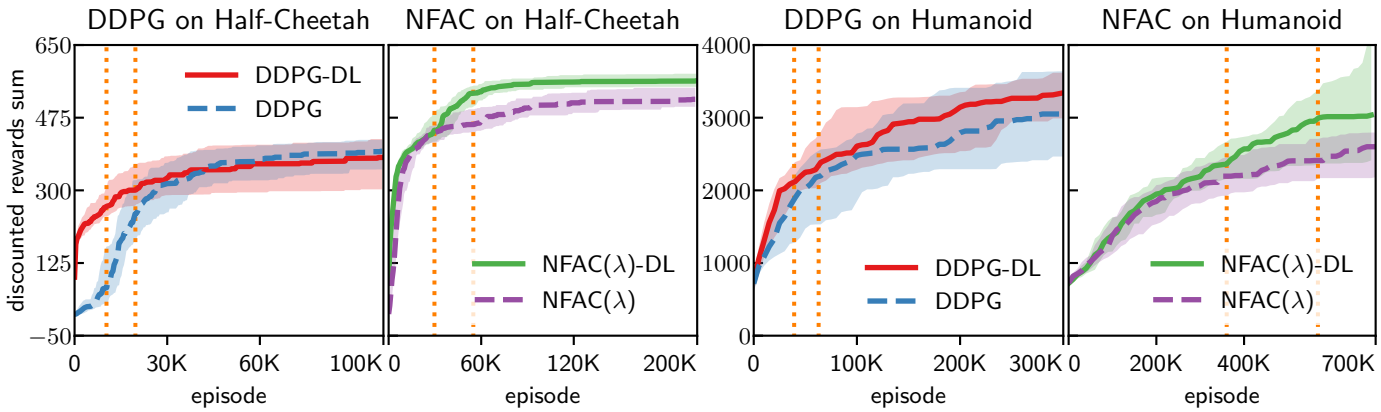

Fig. 6. Median and quartiles of the best registered performance during the testing phase with NFAC(λ) and DDPG on Half-Cheetah and Humanoid environments. When the DL are added, only the "initialization" method is used to transfer. The averages of the transfer moments are displayed vertically (dashed orange).

development order. In further works, it should be interesting to design heuristics to find and try different developmental plan online.

The environments we tried here are probably not the best suited to show how DL can improve the learning, but they are part of the usual benchmarks in deep RL and easily accessible (especially since our release of an open source version). We think that alternative environments where disabled dimensions are not crucial in the first stage might benefit more from our approach (for instance in a ball collecting task [31]). Sensorimotor space enlargement could also be more interesting in environments where the reward function is more sparse but still defined in the source task.

The developmental layer we propose is very generic because its parameters can be controlled in several ways. They can be updated online if one wants to simulate a morphological development over one episode. It allows the agent to control what is to be perceived on the environment and eventually facilitates a return to the source task for a while. A stochastic activation of DL could simulate a continuous transfer instead of the current instantaneous switching of tasks. DL may also be inserted between hidden layers to permit a higher level growth of the parameter space. Sensorimotor enlargement shares similarities with pre-training [32] in deep learning in the sense that it initializes the weights of the model to a "valid" and interesting solution. However, where unsupervised pre-training is done blindly, our sensorimotor enlargement approach seems guided because it relies on the reward function to find this initial solution.

Several previous works deal with improving the neural network representation within reinforcement learning. *Progressive neural networks* [33] add new sets of layers connected to the previous network for each new target task, showing a performance gain in the transfer and a way to prevent forgetting. Even though the work on *progressive neural networks* did not involve a change in input or output spaces and lead to really heavy computations, our sensorimotor space enlargement could use the same method in replacement of DL. *PathNet* [34] evolves the structure of the neural networks online with a genetic algorithm to take advantage of the previous knowledge learned in other tasks by avoiding catastrophic forgetting.

VII. Conclusions and further work

To facilitate reinforcement learning in high-dimensional continuous domains, we introduced the developmental paradigm of sensorimotor space enlargement and proposed a simple and non-invasive solution based on a new neural layer: the developmental layer. We proved the validity of the concept by showing that simply adding developmental layers to the neural networks could help the agent to learn better policies faster on two high-dimensional continuous environments with different algorithms.

We relied on four hypotheses: the global task must be decomposable so that the source task still make sense when input or output dimensions are removed. The designer must provide a default policy for the dimensions that are not controlled during the source task. The reward function must contain information on the source task, otherwise the solution transferred will act randomly in the full problem. And finally, the initial values of parameters that were not present in the source task should not disrupt too much the solution found.

An interesting perspective would be for the agent to automatically adapt its development plan (i.e what is the next sensorimotor dimension to consider). It would also be interesting to complement our approach with other curriculum learning techniques (reward shaping, task simplification, etc.).

Acknowledgment

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. A Bradford Book, 1998.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, and Others, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[3] M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation," in *Advances in Neural Information Processing Systems*, 2016, pp. 1471–1479.

[4] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*. ACM, 2009, pp. 41–48.

[5] M. E. Taylor and P. Stone, "Transfer Learning for Reinforcement Learning Domains : A Survey," *Journal of Machine Learning Research*, vol. 10, pp. 1633–1685, 2009.

[6] M. Asada, K. Hosoda, Y. Kuniyoshi, H. Ishiguro, T. Inui, Y. Yoshikawa, M. Ogino, and C. Yoshida, "Cognitive Developmental Robotics : A Survey," *IEEE Transactions on Autonomous Mental Development*, vol. 1, no. 1, pp. 1–44, 2009.

[7] F. Guerin, "Learning like a baby: a survey of artificial intelligence approaches," *The Knowledge Engineering Review*, vol. 26, no. 02, pp. 209–236, 2011.

[8] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," in *International Joint Conference on Artificial Intelligence*, vol. 47, 2013, pp. 253—-279.

[9] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," 2016.

[10] R. Smith, "Open dynamics engine," 2005.

[11] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.

[12] J. Peters, K. Mülling, and Y. Altun, "Relative Entropy Policy Search." in *Association for the Advancement of Artificial Intelligence*. Atlanta, 2010, pp. 1607–1612.

[13] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229—-256, 1992.

[14] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies." *Evolutionary computation*, vol. 9, no. 2, pp. 159–195, 2001.

[15] V. R. Konda and J. N. Tsitsiklis, "Actor-Critic Algorithms," *Neural Information Processing Systems*, vol. 13, pp. 1008–1014, 1999.

[16] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep Reinforcement Learning that Matters," *ArXiv e-prints*, 2017.

[17] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[18] R. Hafner and M. Riedmiller, "Reinforcement learning in feedback control," *Machine Learning*, vol. 84, no. 1-2, pp. 137–169, 2011.

[19] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *arXiv preprint arXiv:1502.03167*, 2015.

[20] M. Zimmer, Y. Boniface, and A. Dutech, "Neural Fitted Actor-Critic," in *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2016.

[21] M. Riedmiller, "Neural fitted Q iteration - First experiences with a data efficient neural Reinforcement Learning method," in *Lecture Notes in Computer Science*, vol. 3720 LNAI, 2005, pp. 317–328.

[22] H. Van Hasselt, "Reinforcement Learning in Continuous State and Action Spaces," in *Reinforcement Learning*. Springer Berlin Heidelberg, 2012, pp. 207–251.

[23] M. Zimmer, "Developmental reinforcement learning," Ph.D. dissertation, University of Lorraine, 2018.

[24] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[25] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends in Cognitive Sciences*, vol. 3, no. 4, pp. 128–135, 1999.

[26] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, and A. Grabska-Barwinska, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the National Academy of Sciences*, p. 201611835, 2017.

[27] P.-Y. Oudeyer and F. Kaplan, "How can we define intrinsic motivation?" *8th Conf on Epigenetic Robotics*, no. July, pp. 93–101, 2008.

[28] P. Wawrzyński, "Learning to control a 6-degree-of-freedom walking robot," in *International Conference on Computer as a Tool*, 2007, pp. 698–705.

[29] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 4906–4913.

[30] D. P. Kingma and J. L. Ba, "Adam: a Method for Stochastic Optimization," *International Conference on Learning Representations*, pp. 1–13, 2015.

[31] M. Zimmer and S. Doncieux, "Bootstrapping Q-Learning for Robotics from Neuro-Evolution Results," *IEEE Transactions on Cognitive and Developmental Systems*, 2017.

[32] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?" *Journal of Machine Learning Research*, vol. 11, no. Feb, pp. 625–660, 2010.

[33] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive Neural Networks," *arXiv preprint arXiv:1606.04671*, 2016.

[34] C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel, and D. Wierstra, "Pathnet: Evolution channels gradient descent in super neural networks," *arXiv preprint arXiv:1701.08734*, 2017.