Journal of Machine Learning Research 1 (2021) 1-48 Submitted 4/00; Published 10/00

Hyper-archical Reinforcement Learning and its Applications to Transfer Learning

Benjamin Thérien BTHERIEN@UWATERLOO.CA
Cheriton School of Computer Science
University of Waterloo
Waterloo, ON N2L 3G1, Canada

Editor: Benjamin Thérien

Abstract

Hierarchical reinforcement learning (HRL) is a subfield of reinforcement learning which deliberately instills a hierarchical structure into its *agent(s)*. These techniques frequently involve a high-level policy (HLP) and one or many low-level policies (LLPs). Indeed, many classical HRL techniques train their LLPs distinctly and select which to execute using a HLP. While these methods can be effective in certain settings, the classical paradigm is inherently restrictive as it prevents the HLP from dynamically modifying the behaviour of the LLPs. Moreover, this rigid framework disallows parameter sharing between related lower level policies by design. Many recent works have set out to improve upon these shortcomings by developing HRL techniques which allow more flexibility for controlling the lower level policy(Barreto et al.; Haarnoja et al., 2018; Nachum et al.). In our work, we add to this growing body of literature by proposing an alternative approach with applications to transfer learning. In particular, we pre-train one lower level policy on one or many source tasks and train a hypernetwork to guide its behaviour on the target task. This hypernetwork is in fact our higher level policy, whose action space corresponds to the parameters it generates. To our knowledge, our approach, which we denominate hyper-archical reinforcement learning, is the first work to apply hypernetworks to hierarchical reinforcement learning. Our contributions are twofold, we provide a novel approach to HRL and empirically evaluate its applicability to transfer learning.

Keywords: Hierarchical reinforcement learning, Hypernetwork, Transfer learning

1. Introduction

While transfer learning, especially from large pre-trained models, has become ubiquitous in supervised learning today(Devlin et al., 2019; Zhuang et al., 2020), it is comparatively seldom used in reinforcement learning. In general, transfer learning corresponds to leveraging knowledge from one or many source tasks to improve performance or efficiency of learning on a target task (Zhu et al., 2020). Developing principled approaches to transfer learning in deep reinforcement learning (DRL) is critical to the deployment of DRL algorithms in real world contexts, where inefficient algorithms can incur large costs. A recent survey categorizes current approaches to transfer learning in DRL into the following categories: reward shaping, learning from demonstrations, policy transfer, inter-task mapping, and representation transfer (Zhu et al., 2020). While many of these approaches do present effective techniques to speed up learning, most of the techniques work by transferring knowledge from agents trained on closely related tasks without providing a mechanism for their agent

to compose pre-trained knowledge in a new ways. For instance, policy transfer methods learn effective policies on a number source tasks and attempt to distill their knowledge into a policy for the target task. In contrast, HRL approaches allow the higher level policy to leverage the lower level polcy's pre-training by influencing its actions. In this way, many different higher level policies can leverage the same pre-trained lower level policy, allowing for transfer to many different tasks.

Most techniques for hierarchical reinforcement learning which could be applied to transfer learning fall into two broad categories: those which explicitly exploit the option formalism (Sutton et al., 1999; Bacon et al.; Barreto et al.) and those who explicitly leverage hierarchy by composing policies via other means (Haarnoja et al., 2018; Nachum et al.). Approaches leveraging the option formalism learn many distinct lower level policies and select among them with one higher level policy, while composable HRL approaches learn a cascade of policies where the compositing of these distinct components can be seen as one policy. Neither of these approaches take full advantage of parameter sharing or recently proposed conditioning, hypernetwork, and adapter techniques form the supervised deep learning literature (Perez et al., 2018; Mahabadi et al., 2021; Houlsby et al., 2019; Bagaria et al., 2021).

In the following, we apply a HRL aproach to the transfer learning problem which leverages recent advances in parameter generation for deep networks. Specifically, we train a lower level policy on a continuous control task with deep deterministic policy gradient (DDPG). We subsequently freeze its weights and train a higher level policy to generate parameters which modulate the LLP's behaviour on a target task. In theory, the higher level policy should benefit from the LLP's pre-training in that it need not produce low level control behaviours itself. Instead, the HLP should be able to direct the LLP to perform a desirable sequence of lower level behaviours which lead the agent to solving the target task. In some sense, the LLP's pre-training should enable the HLP to reason at higher levels of abstraction. Moreover, our technique is, in theory, inherently scalable in depth and in breadth. Although we only evaluate a hierarchy of two policies, it is possible to create as cascade of policies of arbitrary depth, where each higher level policy generates parameters to influence the layer below it. Additionally, we can scale in width by increasing the number of low level policies in the hierarchy and expanding the actions space of the higher level policy to generate parameters for the chosen number of lower level policies. While these theoretical properties would certainly be interesting to explore, we leave them to future work, instead limiting ourselves to explore a rudimentary to hyper-archical reinforcement learning approach.

The rest of the paper is divided into four main components. Firstly, we review relevant literature to our approach. Secondly, we outline our approach in detail providing relevant background for clarity. Thirdly, we explain our empirical evaluation and the method we compare to. Finally, we discuss the main findings and provide suggestions for future work.

2. Literature Review

Transfer learning in DRL A large number of different techniques exists of which we will only survey a few important works. We refer the interested reader to (Zhu et al., 2020) who survey the topic in depth. (Rusu et al., 2016b) introduce progressive neural networks, a

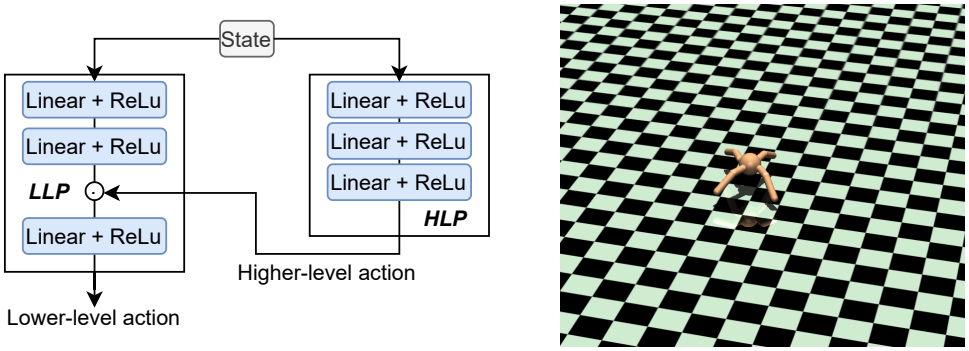
neural network architecture which laterally expands for each new task it is trained on. The network is designed to enable progressive transfer via connections to newly added parts. (Rusu et al., 2016a) and (Parisotto et al., 2016) both publish related work simultaneously, the former distill knowledge from Q networks trained on distinct tasks into one large policy network, while the latter shows how the information from one Q networks can effectively be distilled into a smaller network.

Hypernetworks and conditioning Hypernetworks are defined to be any neural network which is used to generate parameters for another neural network. In their incipient work (Ha et al., 2017) show that a smaller RNN can effectively generate the parameters of a larger RNN for downstream tasks and obtain bettwer performance than if the RNN was trained solely on the downstream task. Since then, different works have adopted the hypernetwork paradigm as a method for conditioning networks. At a large scale, (Mahabadi et al., 2021) use a hypernetwork to generate the parameters of adapter layers (Houlsby et al., 2019) used for parameter efficient transfer learning in NLP. In an application to a smaller model, (Perez et al., 2018) use a GRU to generate the affine parameters of batch norm layers in a convolutional neural network to achieve state of the art results on a visual quesitons answering task. In contrast to the original hypernetworks paper which generates all the parameters of the downstream network, each of these works uses a hypernetwork to generate a small number of parameters in the downstream network. This can be seen as conditioning the downstream network on the generated parameters.

Hierarchical Reinforcement leanring HRL has been an idea of interest to the RL community since the end of the 20th century. Foundational works include (Dayan and Hinton) who introduce the idea of a manager worker hierarchy in reinforcement learning and (Sutton et al., 1999) who introduce the options framework. Many papers have since attempted to design reinforcement learning algorithms which follow from these ideas (Nachum et al.; Vezhnevets et al., 2017; Bacon et al.; Silver and Ciosek, 2012). Of the papers exploiting the option framework, the most closely related to our work is, perhaps, (Bacon et al.) who design an end-to-end differentiable architecture for learning a fixed number of options, controlled by an option policy. Another related work learns a number of options from different cumulants and takes linear combinations of these cumulants to induce new options by combining learned value functions of existing options (Barreto et al.). Other works leverage hierarchies of reinforcement learning agents without explicitly defining options. Indeed, a recent work attempts to leverage invertible neural networks along with maximum entropy reinforcement learning to build a hierarchy of policies whose continuous action space is fed as input to the next layer in the hierarchy (Haarnoja et al., 2018). Their framework is novel and scalable, allowing higher level policies to directly influence any action executed by the lower levels at every time step. As such, it has many similarities to our work which we discuss in the next section.

3. Hyper-archical Reinforcement Learning: Our Proposed Approach

Closely related works The hypernetwork HRL technique we develop is closely related to (Haarnoja et al., 2018) in its unlimited composability and training scheme. In theory, both approaches can be scaled to arbitrary depth. Moreover, we directly adopt the author's training technique. I.E. we freeze the lower level policy's weights before training our



(a) Hyper-archical RL architecture (b) Hyper-archical RL Agent

Figure 1: Figure (a) depicts the 2-level hierarchical reinforcement learning policy formed by the LLP (left) and the HLP (right). We see that the parameters generated by the HLP modulate the forward pass of the LLP network in order to influence its downstream behaviour. Figure (b) on the left shows the ant agent in movement after 4 Million training steps (3M pre-training LLP on a source task and 1M training HLP on the target task)

higher level policy on top. Our approach differs in that our higher level policy acts as a hypernetwork whose continuous actions correspond to the values of parameters it generates in the lower level policy. In contrast, (Haarnoja et al., 2018) use the output of the higher level policy as input to the lower level policy. In this way, the actions of the higher level policy become part of the state that the lower level policy receives at each timestep. Our hypernetwork based approach draws inspiration from the NLP transfer learning litterature (Mahabadi et al., 2021) who also generate parameters to condition a frozen underlying network. Yet our method differ from (Mahabadi et al., 2021) whose hypernetworks consume a task embedding for a limited number of tasks. Our hypernetworks are more similar to the GRU encoder from (Perez et al., 2018) in that their inputs do not come from a fixed set of embeddings and they generate affine parameters of batch norm layers. Specifically, our hypernetworks generate parameters which modulate the latent representation of the lower level policy via a linear transformation. This linear transformation correspond to a Hadamard product between the higher level policy's actions and the lower level policy's latent representation.

3.1 Background

MDPs A Markov Decision Process (Bellman) can be defined as a set $\{\mathcal{A}, \mathcal{S}, \mathcal{T}, \mathcal{R}, \gamma\}$, where $\mathcal{A}$ is the set of actions, $\mathcal{S}$ is the set of states, $\mathcal{T}(s'|s,) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ defines the transition dynamics from states s to s' upon taking actions a, and $\mathcal{R}(s, a) : \mathcal{S} \rightarrow \mathbb{R}$ defines the reward function associated to taking action a in state s. The techniques we propose are based on the assumption that the tasks and environment can effectively be modeled by a markov decision process.

Deep deterministic policy gradient Our agents are trained using DDPG (Lillicrap et al., 2016), an actor critic algorithm whose actor $\pi(s|\theta_\pi) : \mathcal{S} \rightarrow \mathcal{A}$ and critic $Q(s, a|\theta_Q) : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ are represented as multilayer perceptrons parameterized by θ_π and θ_Q respectively. We maintain two critic networks $(Q^{(1)}, Q^{(2)})$ and respective target networks

$(\bar{Q}^{(1)}, \bar{Q}^{(2)})$in order to reduce detrimental overestimations of our q values (van Hasselt et al., 2015). We use a replay buffer to improve sample complexity via off-policy training. At each step we sample a mini-batch from the replay buffer and minimize the mean squared temporal difference error between our current critic's prediction and the one sample approximation of the target critic's predictions. In the double Q-network setting, the target critics predicitons corresponds to $\bar{Q}_{\min}(s,a) = \min(\bar{Q}^{(1)}(s,a|\theta_{\bar{Q}^{(1)}}), \bar{Q}^{(2)}(s,a|\theta_{\bar{Q}^{(2)}}))$. To train our critic networks, we take gradient descent steps to minimize the following loss function.

$$\mathcal{L}(\theta_Q^{(j)}) = \frac{1}{N}\sum_{i}^{N}(Q^{(j)}(s_i,a_i|\theta_Q) - r_i + \gamma \bar{Q}_{\min}(s_{i+1},a_{i+1}))^2 \tag{1}$$

Where $j \in \{1,2\}$ identifies the critic network whose loss function we are computing. The parameters of the actor network are updated by descending the negative of the mean policy gradient in equation 2, where $Q_{\min_i}$ designates the minimum q value of the two critic networks for the ith sample.

$$\nabla_{\theta_\pi} \approx \frac{1}{N}\sum_{i}^{N} \nabla_{\theta_\pi} Q(s_i, \pi(a_i|\theta_\pi)|\theta_Q) \tag{2}$$

DDPG for training higher level policies We generalize DDPG to the hypernetwork setting where our HLP generates parameters to modulate the behaviour of the LLP. We note taht the LLP's weights are frozen when training the higher level policy. DDPG is straightforward to adapt in this setting. We again use two critic networks $(\mathbf{Q}^{(1)}, \mathbf{Q}^{(2)})$ to mitigate the overestimation bias and two corresponding target networks $(\bar{\mathbf{Q}}^{(1)}, \bar{\mathbf{Q}}^{(2)})$. We make an important design choice here: instead of critiquing the higher level policies actions, we critique the lower level policy's modulated actions allowing the HLP's policy gradients to carry signal from the lower level policy. We also use a replay buffer in this setting. Therefore, training the critic networks is identical to equation 1, except that the samples in the replay buffer are generated from the modulated lower level policy. In contrast, training the higher level policy is slighlty different as it now requires backpropagating through the lower level policy. To train the higher level policy $(\mu(s|\theta_\mu) : \mathcal{S} \rightarrow \mathcal{A}'$, where $\mathcal{A}'$ is the action space for the HLP) we descent the negative of its policy gradient given in equation 3.

$$\nabla_{\theta_\mu} \approx \frac{1}{N}\sum_{i}^{N} \nabla_{\theta_\mu} \mathbf{Q}(s_i, \pi(a_i|\theta_\pi, \mu(s|\theta_\mu))|\theta_{\mathbf{Q}}) \tag{3}$$

Transfer Learning In supervised learning, transfer learning can broadly be defined as training a learner on a set of source tasks $T_s \subset \mathbf{T}$ so as to increase its performance or efficiency on a target task $t_t \in \mathbf{T}$, where $\mathbf{T}$ is the set of all tasks. In reinforcement learning, we wish to learn a better policy for our target domain by leveraging information obtained from training on the source task(s) and information from the target task. In our approach, we propose to encode source task information exclusively with the lower level policy (π) and to maximize reward on the target task with the higher level policy (μ). Consequently, training the HLP with DDPG on the target task, we are conducting a search for the optimal HLP μ parameterized by θ_μ as shown in the following equation

$$\arg\max_{\theta_\mu \sim t_t} Q(s, \pi(a|\theta_\pi \sim \mathbf{T}_s, \mu(s|\theta_\mu \sim t_t))|\theta_\mathbf{Q}) \tag{4}$$

In the empirical evaluation that follows we consider the case where $\mathbf{T}_s = \{t_{s_1}\}$. I.E. there is only one source task. However, in general there are often multiple source tasks.

4. Empirical Evaluation

4.1 The Ant Environments

'Ant-V3' is an environment from the popular gym framework (Brockman et al., 2016) which leverages mujoco (Todorov et al., 2012), an open source physics simulator, to obtain realistic samples. This so called environment defines an markov decision process, where a quadrupedal agent receives reward proportional to its forward velocity and certain measures of health (I.E. ensuring the positions of its joints are acceptable). In this MDP the state is a 111 dimensional vector with components corresponding to the agents position, the forces applied to its different joints, external forces, etc... and each action is an 8 dimensional vector of torque values in the range [-1,1]. An agent is considered to have solved the 'Ant-V3' environment if it obtains more than 6000 reward on average.

4.2 The Task Evaluated

The source task For training our lower level policy, we define a customized extension of the 'ant-v3' envrionment where the quadrupedal ant receives reward proportional to its velocity in any direction. This corresponds to the source task $t_{s_1} \in \mathbf{T}_s$.To extract knoledge from this task, we train our lower level policy for 3 million steps (10 hours of training) as shown in figure 2.

The target task We also create different extension of the 'ant-v3' envrionment for our target tasks. In each of these tasks, the agent reward proportional to its forward velocity in a particular direction. To keep things simple we chose directions which align with the coordinate axes in the x-y plane: forward (x), backward (-x), right (y), and left (-y) directions. We note that solving these tasks also corresponds to obtaining 6000 reward or greater.

Comparison baseline To evaluate the effectiveness of modulating the lower level policy with a higher level hypernetwork, we compare it to simply fine tuning the lower level policy on the task target task.

4.3 Results

Figure 2 plots the average test reward of the lower level policy when trained on the source task. Datapoints are taken every 10 training episodes and correspond to the average reward over 10 testing episodes. The line in this plot is in fact an average over two trials, each of which took more than 10 hours to train. The light blue outline corresponds to the standard error between both trials. Both agents manage to solve the task after about 2.5 Million steps of training and produce little standard error indicating that DDPG trained on this task is quite stable. We note that one of the two agents depicted in this curve corresponds to the lower level policy that re-use to solve the target tasks.

Figure 2 plots the performance of our hyper-archical RL agent (called HLP in the figure) compared to a baseline which simply fine tunes the pre-trained lower level policy. We note that the same hyperparameters are used for all target tasks and that the pre-trained lower level policy used is the same in each case. We see that in each figure except for (c) the

fine tuned baseline outperforms our hierarchical approach. Moreover, both agents seem to perform differently depending on the subtask. We hypothesize that these differences in performance may be caused by the direction the agent learns to move in during pre-training. Although the source task rewards velocity in any direction, the agent will end up picking a specific direction at random by virtue of its exploration. Therefore, since each target task rewards the agent for walking along a coordinate axis, there will certainly be some target tasks which reward the agent for moving in a direction withing 45 degrees to the one it saw during pre-training. This explains the differences in performances within the same methods. However the good performance of the higher level policy in figure (c) is more difficult to interpret as it should does not follow the other trends. It seems to indicate that by modulating the computation of the lower level policy, our higher level policy is able to achieve behaviours which the lower level policy cannot produce on its own.

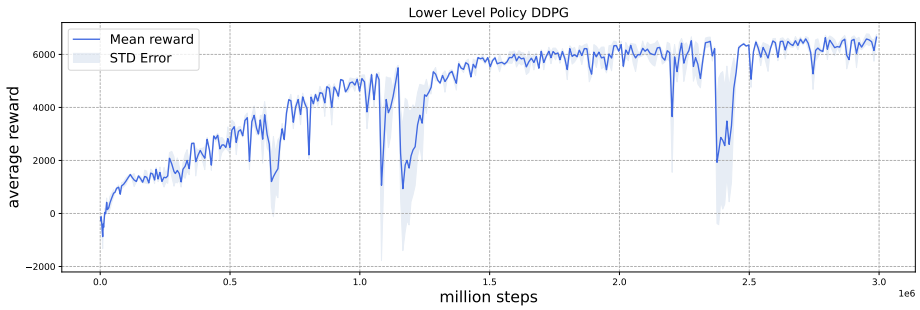
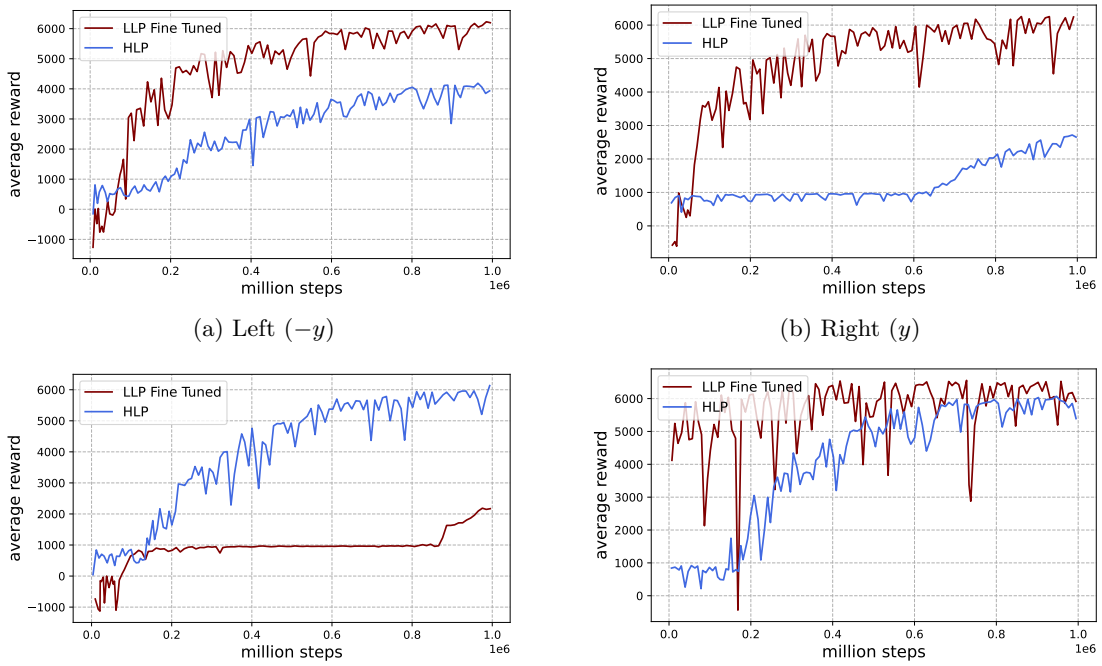


Figure 2: The figure shows the lower lever policy's performance on the source task as it is being pre-trained. This plot is an average over two trials where the light blue region denotes the standard error. Each datapoint corresponds to the average reward over 10 test episodes, taken every 10 training episodes.

4.4 Discussion

Effectiveness of proposed methods In its current state, our proposed hyper-archical reinforcement learning technique is not an effective solution to the transfer learning problem. In the particular settings we use to evaluate our approach it is not exactly clear that the task can benefit from a hypernetwork modulating the lower level policy. The difficulty of transfer learning in this task lies in correctly mapping states to actions based on the target tasks new reward function. The theoretical hope for our hyper-archical reinforcement learning algorithm was for it to guide the lower-level policy to take a desireable sequence of actions. I.E. we hope that it would simply be able to indicate which direction the lower level agent should now move in. The difficulty of accomplishing this is that the lower level agent must first be able to produce the variety of actions required. For instance, if we would like to modulate the behaviour of a lower level agent so that it moves in a different direction, that agent must first be able to move in that different direction to begin with. In this case, it is not clear that the lower level agent has the variety of behaviour needed. Moreover, in the reinforcement learning setting, learning policies which correspond to distinct actions sequences, such as forward locomotion can be produced by maximizing the cumulative reward in an environment under a certain reward function. Therefore, to produce desirable action sequences by modulating the parameters of a lower level agent, we would need to alter the mapping of states to actions so that our new policy corresponds to the behaviours



(a) Left ($-y$)

(b) Right (y)

(c) Forward (x)

(d) Backward ($-x$)

Figure 3: Figures (a), (b), (c), and (d) compare the training HLP to fine tuning the lower level policy on the different target tasks. Each network is trained using the same pre-trained initialization of LLP (trained for 3M steps 2). The same hyperparameters are used in all cases, so the only differences is the environment. In general we see that fine tuning the lower level policy outperforms HLP, however, in plot (c) the opposite behaviour is observed. This suggests that modulating a lower level policy with a hypernetwork can produce new behaviours.

we want. I.E. we would need to learn a mapping to different policies which must be induced under different reward functions. Finding such a mapping is highly non trivial and explains our difficulties here.

Future research While our proposed method is not an effective solution to solve the transfer learning problem we evaluate, the idea of using a higher level hypernetwork to modulate the behaviour of lower level policies is an interesting one. A simpler setting to explore might be to train many lower level controllers to produce distinct behaviours and then train a hypernetwork to softly index into the parameters of the lower level policies. A different idea might be to use a hypernetwork do generate entire policy networks which elicit different behaviours.

5. Conclusion

We proposed a novel approach to HRL and evaluated its performance on a simple transfer learning task. We find that training a higher level policy (hypernetwork) to modulate the parameters of a pre-trained LLP is does not improve upon a simple baseline which fine tunes the lower level policy. Nevertheless, the idea of a HRL agent whose higher level policy corresponds to a hypernetwork is an interesting one and warrants further exploration.

Acknowledgments

We would like to acknowledge support for this project from the University of Waterloo regarding their generous MS Funding.

References

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. 31.

Akhil Bagaria, Jason Senthil, Matthew Slivinski, and George Konidaris. Robustly learning composable options in deep reinforcement learning. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 2161–2169. International Joint Conferences on Artificial Intelligence Organization, 8 2021. Main Track.

Andre Barreto, Diana Borsa, Shaobo Hou, Gheorghe Comanici, Eser Aygün, Philippe Hamel, Daniel Toyama, Jonathan hunt, Shibl Mourad, David Silver, and Doina Precup. The option keyboard: Combining skills in reinforcement learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*. Curran Associates, Inc.

Richard Bellman. ISSN 00959057, 19435274.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In S. Hanson, J. Cowan, and C. Giles, editors, *Advances in Neural Information Processing Systems*. Morgan-Kaufmann.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1*, June 2019.

David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

Tuomas Haarnoja, Kristian Hartikainen, Pieter Abbeel, and Sergey Levine. Latent space policies for hierarchical reinforcement learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1851–1860. PMLR, 10–15 Jul 2018.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR, 09–15 Jun 2019.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. 2016.

Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. *arXiv preprint arXiv:2106.04489*, 2021.

Ofir Nachum, Shixiang (Shane) Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*. Curran Associates, Inc.

Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning, 2016.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron C. Courville. Film: Visual reasoning with a general conditioning layer. In *AAAI*, 2018.

Andrei A. Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation, 2016a.

Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks, 2016b.

David Silver and Kamil Ciosek. Compositional planning using optimal option models. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, page 1267–1274, Madison, WI, USA, 2012. Omnipress. ISBN 9781450312851.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112 (1):181–211, 1999. ISSN 0004-3702.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 3540–3549. JMLR.org, 2017.

Zhuangdi Zhu, Kaixiang Lin, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey. *arXiv preprint arXiv:2009.07888*, 2020.

Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning, 2020.