Guiding exploration via invariant representations

Aleksa Sukovic
Saarland University
alsu00001@stud.uni-saarland.de
7023848

***Abstract*—Modern reinforcement learning (RL) methods are often performant only on a particular task they were trained on. Moreover, such methods require immense number of trials as well as data observations. In this work, I try to empirically evaluate an approach to lifelong-learning that leverages abstract representations of previously learned tasks to guide exploration while learning a new task.**

I. Introduction

The long accepted norm of the modern machine learning systems is that they are good at one thing, namely the one they have been initially trained on. Some of the more recent developments [1] have expanded the capabilities of a single agent to a multitude of tasks, but a more general approach of leveraging past knowledge is still in its infancy. The sub-field of machine learning known as *lifelong learning* aims to build and analyze systems that continuously learn by accumulating past knowledge, that is then used in future learning and problem solving [2]. The aim of this work is to explore a particular application of lifelong learning paradigm to reinforcement learning. More specifically, my aim is to test a new policy, that I called *Lifelong-Learning Deep Q-Network (LLDQN)*, and evaluate agent's behavior, stability of learning and obtained reward compared to a regular DDQN policy [3] [4] [5].

II. Problem Setup

I use a standard definition of the continuous learning process, namely *Lifelong Learning* (LL) [2]. At any particular point in time, the agent is considered to have a *knowledge base* (KB) that consists of $\mathcal{N}$ tasks $\mathcal{T}_1, ..., \mathcal{T}_N$, each of which is associated with a specific *environment* [6]. Upon observing a new task $\mathcal{T}_{N+1}$, the goal of an agent is to learn a policy $\mathcal{P}_{N+1}$ by (partially) leveraging its current KB. In addition, the agent creates an abstract representation of its environment, by learning two separate autoencoders [7], one for the environment's observations and one for environment's action distribution. Thus, the knowledge gained from a particular task $\mathcal{T}$ is contained within (i) task policy, (ii) observation autoencoder and (iii) action autoencoder.

III. Learning Process

When faced with a new task, the first job of an agent is to train two different autoencoders, and thus obtain an abstract representation of the problem. During this time, the agent is not learning a policy in any way and is only allowed to randomly explore the environment. Then, the agent starts learning its policy by leveraging both, its KB, and trained abstract representations. I now describe the training process for autoencoders as well as for two different policies, a baseline policy (i.e., the one not using the knowledge gained from previous tasks) and the LLDQN policy (i.e., the one that leverages the KB).

A. Autoencoders

An (undercomplete) autoencoder is a type of neural network architecture that learns to best represent the input data-set via the (lower dimensional) code. The idea of seeking invariant representations of tasks comes from a working principle of the primate neocortex, which converts sensory inputs, on their way through the cortical layers, to ever-so-increasing abstract representations [8]. Furthermore, autoencoders have already been a tool used in various implementations of lifelong learning [9], albeit in a context different to RL. Moreover, each task encounter by the agent has a different observation and action spaces (i.e., dimensions): interaction between environments wasn't possible without relying on encoded entities.

To train the observation encoder, an agent first samples a series of episodes, each of which consists of multiple observations that constitute the input data-set, which is then used to learn the observation encoding scheme. Agent uses encoded (thus, not raw) observations to learn its policy. Moreover, encoded observations are used to query policies learned on previous tasks (see Figure 1 (top) and [10] script *train_autoencoder* function *train_observation_representation*).

The training of action encoder is rather simplistic, as it only learns to encode a one-hot vector, representing an action to be taken. This allows an action, yielded from a policy of a previously learned task, to be encoded and used in the current task (see Figure 1 (bottom) and [10] script *train_autoencoder* function *train_action_representation*).

B. Baseline Policy

The baseline policy represents our reference agent, which does not use any prior knowledge contained in the KB, and learns to solve the task at hand from scratch. To represent a policy I have chosen to approximate the Q-Value function [11] via the method known as Deep Q-Network [4] [3], more specifically Double Q-Learning [5]. The important thing to point out is that the policy is trained on the encoded observations to output a distribution over the allowed task actions. I have used the network with an input layer ($code\ dim \times 128$),

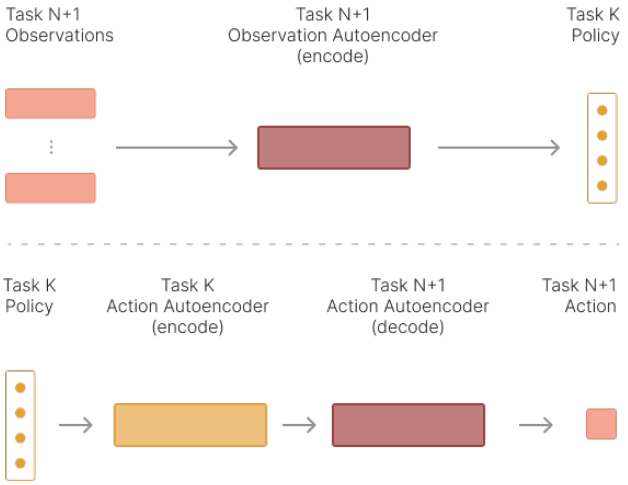


Fig. 1. (Top) Learning observation autoencoder on observations from a new task $\mathcal{T}_{N+1}$. Encoded observations can be passed to previously learned policies to obtain their "opinions". (Bottom) Decoding an "opinion" of previously learned policy of some task $\mathcal{T}_K$ to an action that can be executed in current task's environment.

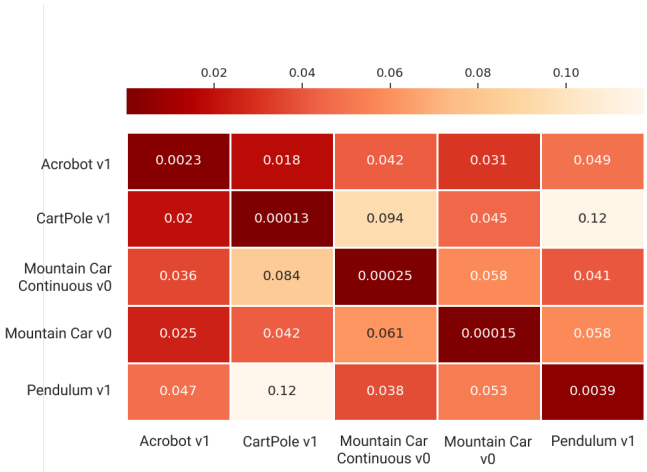


Fig. 2. Task similarity for the *Classic Control* [12] environments. Darker cells correspond to higher degree of similarity.

two fully-connected layers (128×128) and a final output layer ($128 \times num\ actions$). I have trained the network over 3 different runs for 30 epochs, each of which consisted of 4096 environment steps. The target Q-Network was updated every 1024 steps. Finally, I have used the *epsilon-greedy* exploration strategy [11] [3], with an exponentially decaying epsilon over the complete course of the training.

C. LLDQN Policy

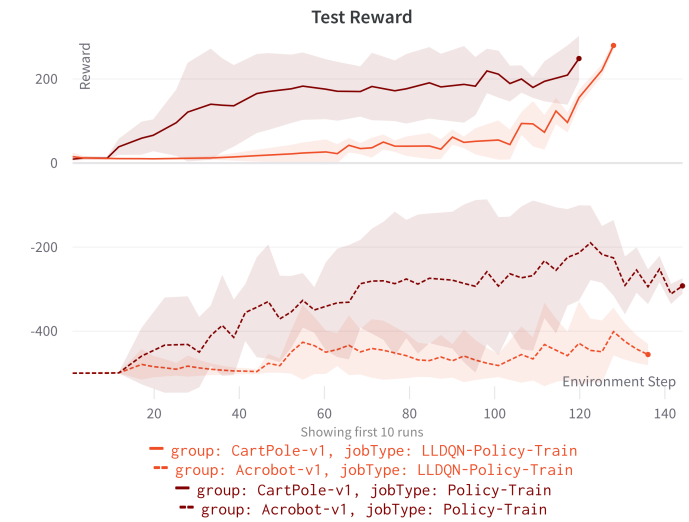
The LLDQN policy contains much of the same building blocks as our baseline policy, but with an important distinction: the *epsilon-greedy* exploration method is replaced with a custom exploration method that leverages prior knowledge (see [10], module *lldqn*). This custom exploration uses the *behaviour policy*, that is simply a distribution over the most relevant tasks from the agent's KB (see [10], module *behavior*). The most relevant tasks are determined by comparing the encoded observations of a new task $\mathcal{T}_{N+1}$, with every other task in the KB $\mathcal{T}_1, ..., \mathcal{T}_N$ (see [10], module *task*, method *get_similarity_index*). Similarity of two tasks is determined by the similarity of their lower-dimensional representations (see Figure 2 and [10] script *evaluate_autoencoder*). My approach differs (and is potentially worse) than [9] as it compares the mean square error (MSE) between *encoded* observations, which may or may not correspond to the real-world similarity of tasks. However, this approach was the only one that I could utilize and that could handle differently-shaped observations coming from different tasks.

Upon determining the similarity of the current task to tasks found in the KB, the LLDQN policy takes all of the tasks whose similarity is under a given threshold (I used 0.05) and constructs a behavioral policy by passing the similarity values through the *Softmax* function (see [10], module *lldqn* function *init_behavior_policy*). Finally, during training, we follow the principle of $\epsilon - greedy$ strategy: with the probability of $1 - \epsilon$, we choose the greedy action, but with probability of ϵ we now choose the action sampled from our behavior policy, instead of acting randomly.

IV. Results

During my testing, I have struggled to obtain consistent results, both with baseline, but also with LLDQN method. I have tried several different environments, including different types of Atari games (see commit 224de91560), as well as *FOREX* and *Stock* trading environments (see commit 1831b08c48). After these failures, I have then decided to focus my evaluation to the *OpenAI Gym Classic Control* [12]. Given the amount of time needed to train deep reinforcement learning, I have opted to test the method only on two environments, namely *CartPole-v1* and *Acrobot-v1* [12]. For each task, I have run the training procedure for both, baseline (see [10] script *train_baseline*) and LLDQN (see [10] script *train_lldqn*) methods, 3 times and tracked training and test reward, standard deviation as well as network update loss. For LLDQN policy, I have trained the *Acrobot-v1* task by leveraging the baseline policy of *CartPole-v1* task and vice versa. The original idea was to train the network until either (i) maximum number of epochs was reached or (ii) the given task environment threshold reward was reached. Unfortunately, the agent failed to (consistently) reach the environment threshold reward, so I opted to train for the full number of epochs, and empirically observe if LLDQN policy achieves any notable difference, w.r.t. any of the mentioned metrics (see Figures 3 and 4).

The training process proved to be rather unstable and highly dependent on the environment seed, indicated by the high standard deviation. In both environments, the LLDQN method behaved worse, achieving lower average test reward of -458.546 as oppose to -309.396 and 57.797 as oppose to 166.78 for *Acrobot-v1* and *CartPole-v1* respectively. As it



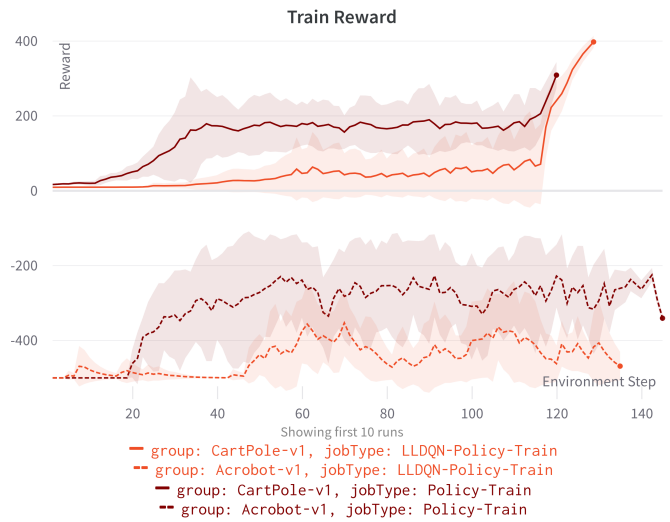
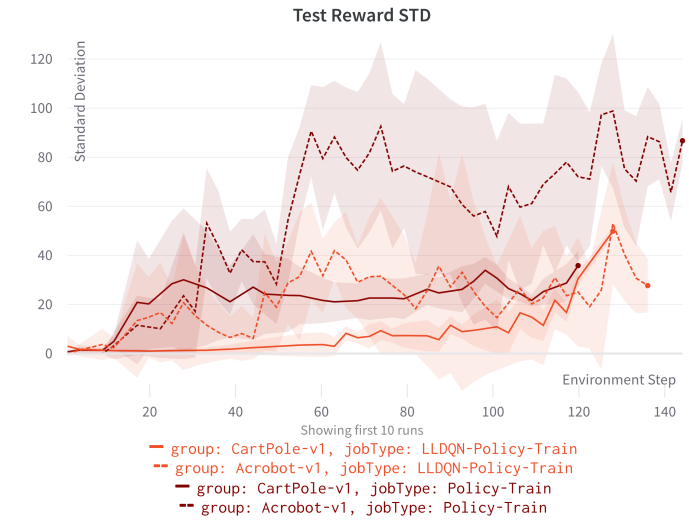


Fig. 3. Test reward and standard deviation for baseline and LLDQN policies. The environment thresholds for solving *CartPole-v1* and *Acrobot-v1* environments are 475.0 and −100 respectively.



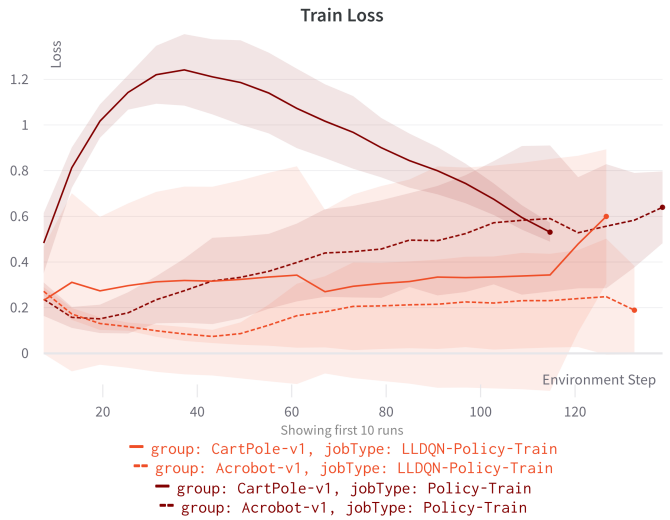


Fig. 4. Training reward and network loss for *CartPole-v1* and *Acrobot-v1* environment for baseline and LLDQN policies.

is sometimes the case in deep reinforcement learning, lower network loss does not directly imply better performance of the agent.

V. Conclusion

In this work I have evaluated my attempt to guide exploration of a RL agent by leveraging previously encountered tasks. Reporting failures is never an enjoyable, but often necessary, thing to do on a way to a functional idea. The method, as described here, failed to match, let alone surpass, the regular DDQN policy. I suspect there might be several reasons for this. First, used autoencoders were rather simple and thus not able to capture abstract enough dependencies, at least not in a way so that they become useful in other tasks. Furthermore, task similarity was done as a mean square error (MSE) comparison of encoded values, which may or may not correspond to a real-world similarity. Continuing, different action space dimension of tasks is indeed a problem, since the result of a policy (i.e., an action) cannot easily be applied in the context of another task, and using the action autoencoders proved to not faithfully transfer this information. Finally, I recall that the initial reasoning behind my idea was to avoid unnecessary exploration in new tasks, by using previous knowledge. However, to achieve this, I might have been better off approximating the State-Value function [11], which evaluates the ”goodness” of a state, and is action-independent. Therefore, while I do think that on a higher level the idea behind LLDQN policy and guided exploration is quite insightful, practically I have found rather little to corroborate it.

REFERENCES

[1] "A Generalist Agent," May 2022, number: arXiv:2205.06175 arXiv:2205.06175 [cs]. [Online]. Available: http://arxiv.org/abs/2205.06175

[2] Z. Chen and B. Liu, "Lifelong Machine Learning, Second Edition," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 12, no. 3, pp. 1–207, Aug. 2018. [Online]. Available: https://www.morganclaypool.com/doi/10.2200/S00832ED1V01Y201802AIM037

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," arXiv, Tech. Rep. arXiv:1312.5602, Dec. 2013, arXiv:1312.5602 [cs] type: article. [Online]. Available: http://arxiv.org/abs/1312.5602

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015. [Online]. Available: http://www.nature.com/articles/nature14236

[5] "Deep Reinforcement Learning with Double Q-learning," Dec. 2015, number: arXiv:1509.06461 arXiv:1509.06461 [cs]. [Online]. Available: http://arxiv.org/abs/1509.06461

[6] OpenAI, "OpenAI Gym," https://www.gymlibrary.ml.

[7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, 2016, oCLC: 1183962587. [Online]. Available: http://public.ebookcentral.proquest.com/choice/publicfullrecord.aspx?p=6287197

[8] J. Hawkins and S. Blakeslee, *On intelligence*, 1st ed. New York: Times Books, 2004.

[9] R. Aljundi, P. Chakravarty, and T. Tuytelaars, "Expert Gate: Lifelong Learning with a Network of Experts," arXiv, Tech. Rep. arXiv:1611.06194, Apr. 2017, arXiv:1611.06194 [cs, stat] type: article. [Online]. Available: http://arxiv.org/abs/1611.06194

[10] A. Sukovic, "Guiding exploration via invariant representations," https://github.com/aleksa-sukovic/lldqn, 2022.

[11] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*, second edition ed., ser. Adaptive computation and machine learning series. Cambridge, Massachusetts: The MIT Press, 2018.

[12] OpenAI, "Classic control," https://www.gymlibrary.ml/environments/classic_control.