A Deep Hierarchical Approach to Lifelong Learning in Minecraft

Chen Tessler ∗  
Shahar Givony ∗  
Tom Zahavy ∗  
Daniel J. Mankowitz ∗  
Shie Mannor  

Technion Israel Institute of Technology, Haifa, Israel

Abstract

The ability to reuse or transfer knowledge from one task to another in lifelong learning problems, such as Minecraft, is one of the major challenges faced in AI. Reusing knowledge across tasks is crucial to solving tasks efficiently with lower sample complexity. We provide a Reinforcement Learning agent with the ability to transfer knowledge by learning reusable skills, a type of temporally extended action (also known as Options (Sutton et. al. 1999)). The agent learns reusable skills to solve tasks in Minecraft, a popular video game which is an unsolved and high-dimensional lifelong learning problem. These reusable skills, which we refer to as Deep Skill Networks (DSNs), are then incorporated into our novel Hierarchical Deep Reinforcement Learning Network (H-DRLN) architecture. The H-DRLN, a hierarchical extension of Deep Q-Networks, learns to efficiently solve tasks by reusing knowledge from previously learned DSNs. The DSNs are incorporated into the H-DRLN using two techniques: (1) a DSN array and (2) skill distillation, our novel variation of policy distillation (Rusu et al., 2015) for learning skills. Skill distillation enables the H-DRLN to scale in lifelong learning, by accumulating knowledge and encapsulating multiple reusable skills into a single distilled network. The H-DRLN exhibits superior performance and lower learning sample complexity (by taking advantage of temporally extended actions) compared to the regular Deep Q Network (Mnih et. al. 2015) in sub-domains of Minecraft. We also show the potential to transfer knowledge between related Minecraft tasks without any additional learning.

1. Introduction

Lifelong learning is defined as the ability to accumulate knowledge across multiple tasks and then reuse or transfer this knowledge in order to solve subsequent tasks (Eaton and Ruvolo, 2013). This is one of the fundamental learning problems in AI (Thrun and Mitchell, 1995; Eaton and Ruvolo, 2013). Lifelong learning in real-world domains suffers from the curse of dimensionality. That is, as the state and action spaces increase, it becomes more and more difficult to model and solve new tasks as they are encountered. In addition, planning over potentially infinite time-horizons and efficiently accumulating and reusing knowledge pose non-trivial challenges. A challenging, high-dimensional domain that incorporates many of the elements found in life-long learning is Minecraft. Minecraft is a popular video game whose goal is to build structures, travel on adventures, hunt for food and avoid zombies. Minecraft is an open research problem as it is impossible to solve the entire game using a single AI technique (Smith and Aha, Oh et al., 2016). Instead, the solution to Minecraft may lie in solving sub-problems, using a divide-and-conquer approach, and then providing a synergy between

∗. The paper was contributed to equally by these authors.
the various solutions. Once an agent learns to solve a sub-problem, it has acquired a skill that can then be reused when a similar sub-problem is subsequently encountered.

Many of the tasks that are encountered by an agent in a lifelong learning setting can be naturally decomposed into skill hierarchies (Stone et al., 2000, 2005; Bai et al., 2015). In Minecraft, tasks, which can also be interpreted as skills include, building houses, finding objects and navigating to various locations in the game. In a high-dimensional, lifelong learning setting such as Minecraft, learning skills and when to reuse the skills is non-trivial. This is key to accumulating knowledge, increasing exploration, efficiently solving tasks and ultimately advancing the capabilities of the Minecraft agent. Reinforcement Learning (RL) provides a generalized approach to skill learning through the options framework (Sutton et al., 1999). Options are Temporally Extended Actions (TEAs) and are also referred to as skills (da Silva et al., 2012) and macro-actions (Hauskrecht et al., 1998). Options have been shown both theoretically (Precup and Sutton, 1997, Sutton et al., 1999) and experimentally (Mann and Mannor, 2013; Mankowitz et al., 2014) to speed up the convergence rate of RL planning algorithms. From here on in, we will refer to options as skills.

In order to learn reusable skills in a lifelong learning setting, the framework needs to be able to (1) learn skills, (2) learn a controller which determines when a skill should be used and reused and (3) be able to efficiently accumulate reusable skills. There are recent works that perform skill learning (Mankowitz et al., 2016a, b; Mnih et al., 2016a, Bacon and Precup, 2015), but these works have focused on learning good skills and have not explicitly shown the ability to reuse skills nor scale with respect to the number of skills in lifelong learning domains. Deep approaches exist for sub-goal learning (Rusu et al., 2016; Kulkarni et al., 2016), but either manually construct sub-goals a-priori or provide intrinsic motivation which may be problematic for complicated problems where designing good intrinsic motivations is not clear and non-trivial.

**Contributions:** (1) We train reusable skills which we refer to as reusable Deep Skill Networks (DSNs) using Deep Q Networks (DQNs) (Mnih, 2015), a well-known Deep RL algorithm, in Minecraft. We perform knowledge transfer of the learned DSNs to new tasks to obtain an optimal solution. (2) Our novel Hierarchical Deep Reinforcement Learning Network (H-DRLN) architecture shown in Figure 1. The H-DRLN controller learns to solve more complicated tasks by reusing the pre-trained DSNs. We incorporate the DSN’s into the H-DRLN via (3) a Deep Skill Module (see Figure 1). There are two types of Deep Skill Modules: (i) a DSN array (Figure 1, Module A) and (ii) a multi-skill distillation network (Figure 1, Module B), our novel variation of policy distillation (Rusu 2015) applied to learning skills. Skill distillation enables the H-DRLN to scale in lifelong learning, by accumulating knowledge and encapsulating multiple reusable skills into a single distilled network. (4) Empirical results for learning a H-DRLN in sub-domains of Minecraft with a DSN array and a distilled skill network. We also verify the improved convergence guarantees for utilizing reusable DSNs (a.k.a options) within the H-DRLN, compared to the vanilla DQN.

### 2. Hierarchical Deep RL Network

**Deep Skill Module:** The pre-learned skills are represented as deep networks and are referred to as Deep Skill Networks (DSNs). They are trained a-priori on various sub-tasks using our version of the DQN algorithm and the regular Experience Replay (ER). Note that the DQN is one choice of architecture and, in principal, other suitable networks may be used in its place. The Deep Skill Module represents a set of N DSNs. Given an input state $s \in S$ and a skill index $i$, it outputs an action $a$ according to the corresponding DSN policy $\pi_{DSN}$. We propose two different Deep Skill Module
architectures: (1) The DSN Array (Figure 1, module A): an array of pre-trained DSNs where each DSN is represented by a separate DQN. (2) The Distilled Multi-Skill Network (Figure 1, module B), a single deep network that represents multiple DSNs. Here, the different DSNs share all of the hidden layers while a separate output layer is trained for each DSN via policy distillation (Rusu, 2015). The Distilled skill network allows us to incorporate multiple skills into a single network, making our architecture scalable to lifelong learning with respect to the number of skills.

**H-DRLN architecture:** A diagram of the H-DRLN architecture is presented in Figure 1 (top). Here, the outputs of the H-DRLN consist of primitive actions as well as skills. The H-DRLN learns a policy that determines when to execute primitive actions and when to reuse pre-learned skills. If the H-DRLN chooses to execute a primitive action \( a_t \) at time \( t \), then the action is executed for a single timestep. If the H-DRLN chooses to execute a skill \( \sigma_t \) (DSN \( i \) as shown in Figure 1), then DSN \( i \) executes its policy, \( \pi_{DSN_i}(s) \) until it terminates and then gives control back to the H-DRLN. This gives rise to two necessary modifications that we needed to make in order to incorporate skills into the learning procedure and generate a hierarchical deep network:

1. **Skill Objective Function:** The H-DRLN loss function has the same structure as the DQN loss function, however instead of minimizing the standard Bellman equation, we minimize the Skill Bellman equation. More specifically, for a skill \( \sigma_t \) initiated in state \( s_t \) at time \( t \) that has executed for a duration \( k \), the H-DRLN target function is given by: 
   \[
y_t = \sum_{j=0}^{k-1} [\gamma^j r_{j+t}] + \gamma^k \max_{\sigma'} Q_{\theta_{\text{target}}}(s_{t+k}, \sigma') \text{ otherwise.}
\]

2. **Skill - Experience Replay:** We extend the regular ER (Lin, 1993) to incorporate skills and term this the Skill Experience Replay (S-ER). There are two differences between the standard ER and our S-ER. Firstly, for each sampled skill tuple, we calculate the sum of discounted cumulative rewards, \( \tilde{r} = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'} \), generated whilst executing the skill. Second, since the skill is executed for \( k \) timesteps, we store the transition to state \( s_{t+k} \) rather than \( s_{t+1} \). This yields the skill tuple \((s_t, \sigma_t, \tilde{r}_t, s_{t+k})\) where \( \sigma_t \) is the skill executed at time \( t \).
3. Experiments

In our experiments, we show (1) the ability of the Minecraft agent to learn DSNs on sub domains of Minecraft (shown in Figure 2a–d). (2) The ability of the agent to reuse a DSN from the navigation 1 domain (Figure 2a) to solve a new and more complex task, termed the two-room domain (Figure 2e). (3) The potential to transfer knowledge between related tasks without any additional learning. (4) We demonstrate the ability of the agent to reuse multiple DSNs to solve the complex-domain (Figure 2f). (5) We use two different Deep Skill Modules and demonstrate that our architecture scales for lifelong learning. State space - As in Mnih (2015), the state space is represented as raw image pixels from the last four image frames which are combined and down-sampled into an 84 × 84 pixel image. Actions - The primitive action space for the DSN consists of six actions: (1) Move forward, (2) Rotate left by 30°, (3) Rotate right by 30°, (4) Break a block, (5) Pick up an item and (6) Place it. Rewards - In all domains, the agent gets a small negative reward signal after each step and a non-negative reward upon reaching the final goal (See Figure 2 for the different domain goals). Training - Episode lengths are 30, 60 and 100 steps for single DSNs, the two room domain and the complex domain respectively. The agent is initialized in a random location in each DSN and in the first room for the two room and complex domains. Evaluation - the agent is evaluated during training using the current learned architecture every 20k (5k) optimization steps (a single epoch) for the DSNs (two room and complex room domains). During evaluation, we averaged the agent’s performance over 500 (1000) steps for the DSNs (two room and complex room domains). Success percentage: The % of successful task completions during evaluation.

3.1 Training a DSN

Our first experiment involved training DSNs in sub-domains of Minecraft (Figure 2a–d), including two navigation domains, a pickup domain and a placement domain respectively. The break domain is the same as the placement domain, except it ends with the break action. Each of these domains come with different learning challenges. The Navigation 1 domain has identical walls, providing a significant learning challenge since there are visual ambiguities with respect to the agent’s location (see Figure 2a). The Navigation 2 domain provides a different learning challenge since there are obstacles that occlude the agent’s view of the exit from different regions in the room (Figure 2b). The pick up (Figure 2c), break and placement (Figure 2d) domains require navigating to a specific location and ending with the execution of a primitive action (Pickup, Break or Place respectively). In order to train the different DSNs, we use the Vanilla DQN architecture (Mnih. 2015) and performed a grid search to find the optimal hyper parameters for learning DSNs in Minecraft. After we tuned the hyper parameters, all the DSNs managed to solve the corresponding sub-domains with 100% success as shown in Figure 3. Table (c).

3.2 Training an H-DRLN with a DSN

In this experiment, we train the H-DRLN agent to solve a more complicated task, the two-room domain, by reusing a single DSN (pre-trained on the navigation 1 domain). This domain consists of two-rooms (Figure 2e(iii)). The first room is shown in Figure 2e(i) with its corresponding exit (Figure 2e(ii)). Note that the exit of the first room is not identical to the exit of the navigation 1 domain (Figure 2a). The second room contains a goal (Figure 2e(iii)) that is the same as the goal.
Figure 2: The domains: (a)-(d) are screenshots for each of the domains we used to train the DSNs. (e) The two-room domain and (f) the complex domain with three different tasks, (i) navigation, (ii) pickup and (iii) placement.

of the navigation 1 domain (Figure 2a). The agent’s available action set consists of the primitive movement actions and the Navigate 1 DSN.

Skill Reusability/Knowledge Transfer: We trained the H-DRLN architecture as well as the vanilla DQN on the two-room domain. The H-DRLN architecture solves the task after a single epoch (76% success) and generates significantly higher reward compared to the vanilla DQN (which only manages 50% success after 39 epochs). This sub-optimal performance is due to wall ambiguities, causing the agent to get stuck in sub-optimal local minima. The H-DRLN makes use of knowledge transfer by reusing the DSN trained on the one-room domain to solve the two-room domain. This DSN is able to identify the exit of the first room (which is different from the exit on which the DSN was trained) and navigates the agent to this exit and then subsequently reuses the DSN to reach the second room exit (see video 1).

Knowledge Transfer without Learning: We then decided to evaluate the DSN (which we trained on the navigation 1 domain) in the two-room domain without performing any additional learning on this network. We found it surprising that the DSN, without any training on the two-room domain, generated a higher reward compared to the vanilla DQN which was specifically trained on the two-room domain for 39 epochs. Figure 3a summarizes the success percentage comparison between the different architectures in the two-room domain. The vanilla DQN, DSN, H-DRLN START and H-DRLN END had average success percentages of 50%, 67.65%, 73.08% and 76% respectively. The DSN performance is sub-optimal compared to the H-DRLN architecture but still manages to solve the two-room domain. This is an exciting result as it shows the potential for DSNs to identify and solve related tasks without performing any additional learning.

3.3 Training an H-DRLN with a Deep Skill Module

In this section, we discuss our results for training and utilizing the H-DRLN with a Deep Skill Module to solve the complex Minecraft domain. In each of the experiments in this section, we utilized DDQN to train the H-DRLN and the DDQN baseline unless otherwise stated.

Complex Minecraft Domain: This domain (Figure 2f) consists of three rooms. Within each room, the agent is required to perform a specific task. Room 1 (Figure 2f(i)) is a navigation task, where the agent needs to navigate around the obstacles to reach the exit. Room 2 (Figure 2f(ii))
Figure 3: (a) Two room domain success percentages for the vanilla DQN, the single DSN, the H-DRLN after a single epoch (START) and in the last epoch (END). (b) The success % learning curves for the (1) H-DRLN with a DSN array, (2) H-DRLN with DDQN and a DSN array, (3) H-DRLN with DDQN and multi-skill distillation, and (4) the DDQN baseline. (c) The success % performance of the original DSNs and distilled multi-skill network on each of the four tasks (Figures 2b − d).

contains two tasks. A pickup task whereby the agent is required to navigate to and collect a block in the center of the room. And a break task, where the agent needs to navigate to the exit and break a door. Finally, Room 3 (Figure 2f (iii)) is a placement task whereby the agent needs to place the block that it collected in the goal location. The agent receives a non-negative reward if it successfully navigates through room 1, collects the block and breaks the door in room 2 and places the block in the goal location in room 3 (Arrow path in Figure 2f). Otherwise the agent receives a small negative reward at each timestep. Note that the agent needs to complete three separate tasks before receiving a sparse, non-negative reward. The agent’s available action set are the original primitive actions as well as the DSN’s: (1) Navigate 2, (2) Pickup, (3) Break and (4) Placement.

Training and Distilling Multiple DSNs: As mentioned in the H-DRLN Section, there are two ways to incorporate skills into the Deep Skill Module: (1) DSN Array and (2) Multi-Skill Distillation. For both the DSN array and multi-skill distillation, we utilize four pre-trained DSNs (Navigate 2, Pickup, Break and Placement). These DSNs collectively form the DSN array. For the multi-skill distillation, we utilized the pre-trained DSNs as teachers and distill these skills directly into a single network (the student) using the distillation setup described in the Background Section. Once trained, we tested the distilled network separately in each of the three individual rooms (Figure 2b − d). The performance for each room is shown in Figure 3, Table(c) for temperatures $\tau = 0.1$ and $\tau = 1$. The high success percentages indicate that the agent is able to successfully complete each task using a single distilled network. In contrast to policy distillation, our novelty lies in the ability to, not only distil skills into a single network, but also learn a control rule (using the H-DRLN) that switches between the skills to solve a given task.

Training H-DRLN: We now show results for training the (1) H-DRLN with a DSN array, (2) H-DRLN with DDQN and a DSN array and (3) H-DRLN with DDQN and a distilled multi-skill network (with $\tau = 0.1$). This is compared to (4) a DDQN baseline. The learning curves can be seen in Figure 3(b). We performed these trials 5 times for each architecture and measured success rates of $85 \pm 10\%$, $91 \pm 4\%$ and $94 \pm 4\%$ (mean% ± std) for the H-DRLN, H-DRLN with DDQN DSN array and H-DRLN with a DDQN distilled multi-skill network respectively. To calculate these values we averaged the success percentages for the final 10 epochs. Note that the distilled H-DRLN has a higher average success rate and both H-DRLN’s with DDQN have lower variance. The DDQN was unable to solve the task. This is due to a combination of wall ambiguities (as in the two room domain) and requiring more time to learn. The H-DRLN is able to overcome ambiguities and also learns to reuse skills (see video 1).
4. Discussion

We have provided the first results for learning Deep Skill Networks (DSNs) in Minecraft, a lifelong learning domain. The DSNs are learned using a Minecraft-specific variation of the DQN (Mnih, 2015) algorithm. Our Minecraft agent also learns how to reuse these DSNs on new tasks by utilizing our novel Hierarchical Deep RL Network (H-DRLN) architecture. We incorporate multiple skills into the H-DRLN using (1) the DSN array and (2) the scalable distilled multi-skill network, our novel variation of policy distillation. In addition, we show that the H-DRLN provides superior learning performance and faster convergence compared to the DDQN, by making use of temporally extended actions (Sutton et al. 1999). Our work can also be interpreted as a form of curriculum learning (Bengio et al. 2009) for RL. Here, we first train the network to solve relatively simple sub-tasks and then use the knowledge it obtained to solve the composite overall task. We also show the potential to perform knowledge transfer between related tasks without any additional learning. We see this work as a building block towards truly general lifelong learning using hierarchical RL and Deep Networks. This architecture also has the potential to be utilized in other 3D domains such as Doom and Labyrinth (Mnih et al., 2016b). Recently, it has been shown that Deep Networks tend to implicitly capture the hierarchical composition of a given task (Zahavy et al., 2016). In future work we plan to utilize this implicit hierarchical composition to learn DSNs. We also aim to train the teacher networks (DSNs) (Suddarth and Kergosien, 1990), whilst simultaneously guiding learning in the student network (our H-DRLN).

Acknowledgement

This research was supported in part by the European Community’s Seventh Framework Programme (FP7/2007-2013) under grant agreement 306638 (SUPREL).

References


David Bonanno1 Mark Roberts2 Leslie Smith and David W Aha. Selecting subgoals using deep learning in minecraft: A preliminary report.


