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
chen.tessler, shahargiv, tomzahavy {@campus.technion.ac.il }, danielm@tx.technion.ac.il, shie@ee.technion.ac.il

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 using Deep Q Networks (Mnih et. al. 2015) 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 is a hierarchical version of Deep Q-Networks and learns to efficiently solve tasks by reusing knowledge from previously learned DSNs. The H-DRLN exhibits superior performance and lower learning sample complexity (by taking advantage of temporal extension) 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). There are different ways to performing knowledge transfer across tasks. For example, Ammar et al. (2014) and Ammar et al. (2015) transfer knowledge via a latent basis whereas Brunskill and Li (2014) perform batch optimization across all encountered tasks.

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. A challenging, high-dimensional domain that incorporates many of the elements found in lifelong learning is Minecraft. Minecraft is a popular video game whose goal is to build structures, travel on adventures, hunt for food and avoid zombies. An example screenshot from the game is seen in Figure 1. Minecraft is an open research problem as it is impossible to solve the entire game using a single AI technique. Instead, the solution to Minecraft may lie in solving sub-problems, using a divide-and-conquer approach, and then providing a synergy between 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; Stone et al., 2005; Bai et al., 2015). In Minecraft for example, consider building a wooden house as seen in Figure 1. This task can be decomposed into sub-tasks (a.k.a skills) such as chopping trees, sanding the wood, cutting the wood into boards and finally nailing the boards together. Here, the knowledge gained from chopping trees can also be partially reused when cutting the wood into boards. In addition, if the agent receives a new task to build a small city, then the agent can reuse the skills it acquired during the ‘building a house’ task.

In a lifelong learning setting such as Minecraft, learning skills and when to reuse the skills are key to increasing explo-
ration, efficiently solving tasks and advancing the capabilities of the Minecraft agent. As mentioned previously, Minecraft and other lifelong learning problems suffer from the curse of dimensionality. Therefore as the dimensionality of the problem increases, it becomes increasingly non-trivial to learn reasonable skills as well as when to reuse these skills.

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] to speed up the convergence rate of RL algorithms. From here on in, we will refer to options as skills.

Recent work in RL has provided insights into learning reusable skills [Mankowitz et al., 2016a; Mankowitz et al., 2016b], but this has been limited to low dimensional problems. In high-dimensional lifelong learning settings (E.g. Minecraft), learning from visual experiences provides a potential solution to learning reusable skills. With the emergence of Deep Reinforcement Learning, specifically Deep Q-Networks (DQNs) [Mnih et al., 2015], RL agents are now equipped with a powerful non-linear function approximator that can learn rich and complex policies. Using these networks the agent learns policies (or skills) from raw image pixels, requiring less domain specific knowledge to solve complicated tasks (E.g Atari video games [Mnih et al., 2015]). While different variations of the DQN algorithm exist [Van Hasselt et al., 2015; Schaul et al., 2015; Wang et al., 2015; Bellemare et al., 2015], we will refer to the Vanilla DQN [Mnih et al., 2015] unless otherwise stated.

In our paper, we focus on learning reusable RL skills using Deep Q Networks [Mnih et al., 2015], by solving subproblems in Minecraft. These reusable skills, which we refer to as Deep Skill Networks (DSNs) are then incorporated into our novel Hierarchical Deep Reinforcement Learning (RL) Network (H-DRLN) architecture. The H-DRLN, which is a hierarchical version of the DQN, learns to solve more complicated tasks by reusing knowledge from the pre-learned DSNs. By taking advantage of temporal extension, the H-DRLN learns to solve tasks with lower sample complexity and superior performance compared to vanilla DQNs.

**Contributions:** (1) A novel Hierarchical Deep Reinforcement Learning Network (H-DRLN) architecture. (2) We show the potential to learn reusable Deep Skill Networks (DSNs) and perform knowledge transfer of the learned DSNs to a new task to obtain an optimal solution. (3) Empirical results for learning a H-DRLN in a sub-domain of Minecraft which outperforms the vanilla DQN. (4) We verify empirically the improved convergence guarantees for utilizing reusable DSNs (aka options) within the H-DRLN, compared to the vanilla DQN. (5) The potential to transfer knowledge between related tasks without any additional learning.

## 2 Background

### Reinforcement Learning

The goal of an RL agent is to maximize its expected return by learning a policy \( \pi : S \to \Delta_s \) which is a mapping from states \( s \in S \) to a probability distribution over the action space \( A \). At time \( t \) the agent observes a state \( s_t \in S \), selects an action \( a_t \in A \), and receives a bounded reward \( r_t \in [0, R_{\text{max}}] \) where \( R_{\text{max}} \) is the maximum attainable reward and \( \gamma \in [0,1] \) is the discount factor. Following the agents action choice, it transitions to the next state \( s_{t+1} \in S \). We consider infinite horizon problems where the cumulative return at time \( t \) is given by \( R_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_t \). The action-value function \( Q^\pi(s,a) = \mathbb{E}[R_t|s_t=s, a_t=a, \pi] \) represents the expected return after observing state \( s \) and taking an action \( a \) under a policy \( \pi \). The optimal action-value function obeys a fundamental recursion known as the Bellman equation,

\[
Q^*(s_t, a_t) = \mathbb{E} \left[ r_t + \gamma \max_{a'} Q^*(s_{t+1}, a') \right]
\]

### Deep Q Networks

The DQN algorithm [Mnih et al., 2015] approximates the optimal Q function with a Convolutional Neural Network (CNN) [Krizhevsky et al., 2012], by optimizing the network weights such that the expected Temporal Difference (TD) error of the optimal bellman equation (Equation 1) is minimized,

\[
\mathbb{E}_{s_t, a_t, r_t, s_{t+1}} \| Q_{\theta}(s_t, a_t) - y_t \|^2_2, \quad (1)
\]

where

\[
y_t = \begin{cases} 
  r_t & \text{if } s_{t+1} \text{ is terminal} \\
  r_t + \gamma \max_a Q_{\theta_{\text{target}}}(s_{t+1}, a') & \text{otherwise}
\end{cases}
\]

Notice that this is an off-line learning algorithm, meaning that the tuples \( \{s_t, a_t, r_t, s_{t+1}, \gamma\} \) are collected from the agents experience and are stored in the Experience Replay (ER) [Lin, 1993]. The ER is a buffer that stores the agents experiences at each time-step \( t \), for the purpose of ultimately training the DQN parameters to minimize the loss function. When we apply minibatch training updates, we sample tuples of experience at random from the pool of stored samples in the ER. The DQN maintains two separate Q-networks. The current Q-network with parameters \( \theta \), and the target Q-network with parameters \( \theta_{\text{target}} \). The parameters \( \theta_{\text{target}} \) are set to \( \theta \) every fixed number of iterations. In order to capture the game dynamics, the DQN represents the state by a sequence of image frames.

### Skills, Options, Macro-actions [Sutton et al., 1999]

A skill \( \sigma \) is a temporally extended control structure defined by a triple \( \sigma = \langle I, \pi, \beta \rangle \) where \( I \) is the set of states where the skill can be initiated, \( \pi \) is the intra-skill policy, which determines how the skill behaves in encountered states, and \( \beta \) is the set of termination probabilities determining when a skill will stop execution. \( \beta \) is typically either a function of state \( s \) or time \( t \).

### Semi-Markov Decision Process (SMDP)

Planning with skills can be performed using SMDP theory. For each state in the skill initiation set \( I \), the SMDP model predicts the state in which the skill will terminate and the total reward received
along the way. More formally, a SMDP can be defined by a five-tuple \( < S, \Sigma, R, \gamma > \) where \( S \) is a set of states, \( \Sigma \) is a set of skills, and \( P \) is the transition probability kernel. We assume rewards received at each timestep are bounded by \([0, R_{\text{max}}]\). \( R : S \times \sigma \rightarrow [0, \frac{R_{\text{max}}}{1-\gamma}] \) represents the expected discounted sum of rewards received during the execution of a skill \( \sigma \) initialized from a state \( s \).

**Skill Policy:** A skill policy \( \mu : S \rightarrow \Delta_{\Sigma} \) is a mapping from states to a probability distribution over skills \( \Sigma \). The action-value function \( Q : S \times \Sigma \rightarrow \mathbb{R} \) represents the long-term value of taking a skill \( \sigma \in \Sigma \) from a state \( s \in S \) and thereafter always selecting skills according to policy \( \mu \) and is defined by \( Q(s, \sigma) = \mathbb{E}[\sum_{i=0}^{\infty} \gamma^i R_i(s, \sigma), \mu] \). We denote the action-value function as \( Q^\pi(s, \sigma) \).

\[
Q^\pi(s, \sigma) = \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{k-1} r_{t+k} | s_t = s, \sigma] ,
\]

\[
P^\sigma_{s,s'} = \sum_{j=0}^{\infty} \gamma^j P_r[k = j, s_{t+j} = s' | s_t = s, \sigma]
\]

Under these definitions the optimal skill value function is given by the following equation [Stolle and Precup, 2015] as:

\[
Q^*_\sigma(s, \sigma) = \mathbb{E}[R^\sigma_s + \gamma^j \max_{\sigma' \in \Sigma} Q^*_\sigma(s', \sigma')] \quad (2)
\]

### 3 Hierarchical Deep RL Network

The Hierarchical Deep RL Network (H-DRLN) is a new architecture, based on the DQN, that facilitates skill reuse in lifelong learning. In this section, we provide an in-depth description of this network architecture as well as necessary modifications that we implemented in order to convert a vanilla DQN into its hierarchical counterpart.

**H-DRLN architecture:** A diagram of this architecture is presented in Figure 2. Here, the outputs of the H-DRLN consist of primitive actions (E.g. Left (L), Right (R) and Forward (F)) as well as skills. The H-DRLN learns a policy that determines when to execute primitive actions and when to reuse pre-learned skills. The pre-learned skills are represented with deep networks and are referred to as Deep Skill Networks (DSNs). They are trained a-priori on various sub-tasks using the vanilla DQN algorithm and the regular Experience Replay (ER) detailed in Section 2.

If the H-DRLN chooses to execute a primitive action \( a_t \) at time \( t \), then the action is executed for a single timestep. However, if the H-DRLN chooses to execute a skill \( \sigma_t \) (and therefore a DSN as shown in Figure 2), then the DSN executes its policy, \( \pi_{\text{DSN}}(s) \) for a duration of \( k_t \) timesteps 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 truly hierarchical deep network: (1) Optimize an objective function that incorporates skills; (2) Construct an ER that stores skill experiences.

**Skill Objective Function:** As mentioned previously, a H-DRLN extends the vanilla DQN architecture to learn control between primitive actions and skills. The H-DRLN loss function has the same structure as Equation [1] however instead of minimizing the standard Bellman equation, we minimize the Skill Bellman equation (Equation [2]). More specifically, for a skill \( \sigma_t \) initiated in state \( s_t \) that has a time duration \( k_t \), the H-DRLN target function is given by:

\[
y_t = \begin{cases} 
\sum_{k=0}^{k_t-1} \gamma^k r_{t+k} & \text{if } s_{t+k} \text{ is terminal} \\
\sum_{k=0}^{k_t-1} \gamma^k r_{t+k} + \gamma^k \max_{\sigma' \in \Sigma} Q^\sigma_{t+k+1}(s_{t+k}, \sigma') & \text{else}
\end{cases}
\]

ensuring that temporal extension is preserved.

**Skill - Experience Replay:** We extend the regular ER [Mnih et al., 2015] to incorporate skills and have termed this 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 generated while executing the skill and store this sum in the variable \( \tilde{r} \). Second, since the skill is executed for \( k_t \) timesteps, we store the transition to state \( s_{t+k} \) rather than \( s_{t+1} \). This yields the skill tuple \((s_t, \sigma_t, \tilde{r}, s_{t+k})\) where \( \sigma_t \) is the skill executed at time \( t \).

### 4 Experiments

In order to solve new tasks as they are encountered in a lifelong learning scenario, the agent needs to be able to adapt to new game dynamics as well as reuse skills that it has learned from solving previous tasks. In our experiments, we show (1) the ability of the Minecraft agent to learn a DSN on a Minecraft task termed the one-room domain, shown in Figure 3. We then show (2) the ability of the agent to reuse this DSN to solve a new task, termed the two-room domain shown in Figure 5, by learning a Hierarchical Deep RL Network (H-DRLN) which incorporates this DSN as an action output. Finally, we show (3) the potential to transfer knowledge between related tasks without any additional learning.

**Deep Network Architecture** - The deep network architecture used to represent the DSN and H-DRLN is the same as that of the vanilla DQN architecture [Mnih et al., 2015]. The
H-DRLN however has a different Q-layer with a DSN as an output. **State space** - As in Mnih et al. (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 image which is then vectorized. **Actions** - The action space for the DSN consists of three actions: (1) Move forward (F), (2) Rotate left by 30° (L) and (3) Rotate right by 30°. The H-DRLN contains the same set of actions as well as the DSN as a fourth action output. **Rewards** - The agent gets a negative reward that is proportional to its distance to the goal, where the goal is to exit the room. In addition, upon reaching the goal the agent receives a large positive reward. **Training** - the agent learns in epochs. Each epoch starts from a random location in the domain and terminates after the agent makes 30 (60) steps in the one (two)-room domain. **Evaluation** - In all of the simulations, we evaluated the agent during training using the current learned architecture every 20k (5k) optimization steps. During evaluation, we averaged the agent’s performance over 500 (1k) steps.

### 4.1 Training a DSN

Our first experiment involved training a DSN in the one room domain (Figure 3). To do so we used the Vanilla DQN parameters that worked on the Atari domain (Mnih et al., 2015) as a starting point and then performed a grid search to find the optimal parameters for learning a DSN for the Minecraft one-room domain. The best parameter settings that we found include: (1) a higher learning ratio (iterations between emulator states, n-replay = 16), (2) higher learning rate (learning rate = 0.0025) and (3) less exploration (eps endt - 400K). We implemented these modifications, since the standard Minecraft emulator has a slow frame rate (approximately 400 ms per emulator timestep), and these modifications enabled the agent to increase its learning between game states. We also found that a smaller experience replay (replay.memory - 100K) provided improved performance, probably due to our task having a relatively short time horizon (approximately 60 timesteps). The rest of the parameters from the Vanilla DQN remained unchanged, since Minecraft and Atari (Mnih et al., 2015) share relatively similar in-game screen resolutions.

![Single Room](https://github.com/h2r/burlapcraft)

Figure 3: Minecraft Task 1: (a) The one-room domain. (b) The goal at the exit of the one-room domain.

---

**Skill Reusability/Knowledge Transfer**: We trained the H-DRLN architecture as well as the vanilla DQN on the two-room domain. The average reward per epoch is shown in Figure 6. We noticed two important observations. (1) The H-DRLN architecture solves the task after a single epoch and generates significantly higher reward compared to the vanilla DQN as seen in the figure. This is because 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. The DSN is also able to navigate

---

**Figure 4**: Training a DSN in the single room domain.
the agent to the exit of the second room and completes the task. The DSN is a temporally extended action as it lasts for multiple timesteps and therefore increases the exploration of the RL agent enabling it to learn to solve the task faster than the vanilla DQN. A video showing the performance of the Minecraft agent using the learned H-DRLN in the two room domain is available online. (2) The vanilla DQN fails to solve the task after 39 epochs. Since the domain contains ambiguous looking walls, the agent tends to get stuck in sub-optimal local minima. The agent completes the task approximately 80% of the time using the H-DRLN whereas it completes the task approximately 5% of the time using the vanilla DQN after 39 epochs.

**Temporal Extension combined with Primitive Actions:** Figure 7 shows the action distribution of the agent’s policy during training. We can see that the H-DRLN learns when to make use of temporal extension by reusing the DSN and when to use primitive actions. The DSN runs for 10 timesteps or terminates early if the agent reaches the goal. 10 timesteps is not sufficient to reach the exit of either room and therefore the agent needs to rely on primitive actions, as well as the DSN, in order to solve the given task.

**Knowledge Transfer without Learning:** We then decided to evaluate the DSN (which we trained on the one-room 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 for the two-room domain for 39 epochs as shown in Figure 8. The DSN performance is not optimal compared to the H-DRLN architecture as seen in the figure 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.

5 **Discussion**

We have provided the first results for learning Deep Skill Networks (DSNs) in Minecraft, a lifelong learning domain. DSNs are learned using a Minecraft-specific variation of the DQN [Mnih et al., 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. In addition, we show that the H-DRLN provides superior learning performance and faster convergence compared to the vanilla DQN, by making use of temporal extension [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.

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. In addition, it is possible to distill much of the knowledge from multiple teacher networks into a single student network [Parisotto et al., 2015; Rusu et al., 2015]. We wish to perform a similar technique as well as add auxiliary tasks to train the teacher networks (DSNs) [Suddarth and Kergosien, 1990], ultimately guiding learning in the student network (our H-DRLN).
References

[Ammar et al., 2014] Haitham B Ammar, Eric Eaton, Paul Ruvolo, and Matthew Taylor. Online multi-task learning for policy gradient methods. In Proceedings of the 31st International Conference on Machine Learning, 2014.

[Ammar et al., 2015] Haitham Bou Ammar, Rasul Tutunov, and Eric Eaton. Safe policy search for lifelong reinforcement learning with sublinear regret. arXiv preprint arXiv:1505.05798, 2015.

[Bai et al., 2015] Aijun Bai, Feng Wu, and Xiaoping Chen. Online planning for large markov decision processes with hierarchical decomposition. ACM Transactions on Intelligent Systems and Technology (TIST), 6(4):45, 2015.

[Bellemare et al., 2015] Marc G Bellemare, Georg Ostrovski, Arthur Guez, Philip S Thomas, and Rémi Munos. Increasing the action gap: New operators for reinforcement learning. arXiv preprint arXiv:1512.04860, 2015.

[Bengio et al., 2009] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In Proceedings of the 26th annual conference on machine learning, pages 41–48. ACM, 2009.

[Brunskill and Li, 2014] Emma Brunskill and Lihong Li. Pac-inspired option discovery in lifelong reinforcement learning. In Proceedings of the 31st International Conference on Machine Learning (ICML-14), 2014.

[da Silva et al., 2012] B.C. da Silva, G.D. Konidaris, and A.G. Barto. Learning parameterized skills. In Proceedings of the Twenty Ninth International Conference on Machine Learning, June 2012.

[Eaton and Ruvolo, 2013] Eric Eaton and Paul L Ruvolo. Ella: An efficient lifelong learning algorithm. In Proceedings of the 30th international conference on machine learning (ICML-13), pages 507–515, 2013.

[Hauskrecht et al., 1998] Milos Hauskrecht, Nicolas Meuleau, Leslie Pack Kaelbling, Thomas Dean, and Craig Boutilier. Hierarchical solution of markov decision processes using macro-actions. In Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, pages 220–229, 1998.

[Krizhevsky et al., 2012] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012.

[Lin, 1993] Long-Ji Lin. Reinforcement learning for robots using neural networks. 1993.

[Mankowitz et al., 2016a] Daniel J. Mankowitz, Timothy A. Mann, and Shie Mannor. Adaptive Skills, Adaptive Partitions (ASAP). arXiv preprint arXiv:1602.03351, 2016.

[Mankowitz et al., 2016b] Daniel J. Mankowitz, Timothy A. Mann, and Shie Mannor. Iterative Hierarchical Optimization for Misspecified Problems (IHOMP). arXiv preprint arXiv:1602.03348, 2016.

[Mann and Mannor, 2013] Timothy A. Mann and Shie Mannor. Scaling up approximate value iteration with options: Better policies with fewer iterations. International Conference on Machine Learning, 2013.

[Mnih et al., 2015] Volodymyr Mnih, koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. Nature, 518(7540), 2015.

[Parisotto et al., 2015] Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. arXiv preprint arXiv:1511.06342, 2015.

[Precup and Sutton, 1997] Doina Precup and richard S Sutton. Multi-time models for temporally abstract planning. In Advances in Neural Information Processing Systems 10 (Proceedings of NIPS’97), 1997.

[Rusu et al., 2015] Andri A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, koray Kavukcuoglu, and Raia Hadsell. Policy distillation. arXiv preprint arXiv:1511.06295, 2015.

[Schaul et al., 2015] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. arXiv preprint arXiv:1511.05952, 2015.

[Stolle and Precup, 2005] Martin Stolle and Doina Precup. Learning options in reinforcement learning. Springer.

[Stone et al., 2000] Peter Stone, Manuela Veloso, Park Ave, and Florham Park. Layered Learning. EMCL 2000 Proceedings of the 17th International Conference on Machine Learning, 1810(June):369–381, 2000.

[Stone et al., 2005] Peter Stone, richard S Sutton, and Gregory Kuhlmann. Reinforcement learning for robocup soccer keepaway. Adaptive Behavior, 13(3):165–188, 2005.

[Suddarth and Kergosien, 2000] Steven C Suddarth and YL Kergosien. Rule-injection hints as a means of improving network performance and learning time. In Neural Networks, pages 120–129. Springer, 1990.

[Sutton et al., 1999] 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), August 1999.

[Thrun and Mitchell, 1995] Sebastian Thrun and Tom M Mitchell. Lifelong robot learning. Springer, 1995.

[Van Hasselt et al., 2015] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. arXiv preprint arXiv:1509.06461, 2015.

[Wang et al., 2015] Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling network architectures for deep reinforcement learning. arXiv preprint arXiv:1511.06581, 2015.

[Zahavy et al., 2016] Tom Zahavy, Nir Ben Zrihem, and Shie Mannor. Graying the black box: Understanding dqns. arXiv preprint arXiv:1602.02658, 2016.