# Learning Long-range Temporal Abstractions in Hierarchical Deep Reinforcement Learning

**Pravish Sainath**
260849153
pravish.sainath@mail.mcgill.ca

**Shivendra Bhardwaj**
260905550
shivendra.bhardwaj@mail.mcgill.ca

## Abstract

Using good temporal abstractions has been known to scale up reinforcement learning algorithms by reducing the complexity of taking actions and improve exploration in the environment. We study different frameworks that learn value functions with temporal abstractions : Deep Recurrent Q-Network (DRQN) [1], Hierarchical Deep Q-Network (h-DQN) [2] and Hierarchical Recurrent Deep Q-Network (h-DRQN)[3]. We investigate the ability of these three different models to learn temporal abstractions by empirically verifying the level to which they can learn a set of goal states. We delve into examining the effect of recurrent value network structure and hierarchical control structure in learning these long-term dependencies. By analyzing the results of training these models in two time-sparse reward settings, we conclude that h-DRQN is the most powerful framework among the three and that it learns a robust value function by exploring well with a better set of goal states.

## 1 Introduction

### 1.1 Temporal Abstractions

Temporal abstraction denotes a form of encapsulation of several atomic actions into a single higher-level action with temporally extended courses of action. This method of knowledge representation through more abstract concepts has proven to significantly enhance both the tasks of learning and planning for many complex problems.

A formalism of temporal abstractions was established using a Semi-Markov Decison Process (SMDP)[4] which is a generalization of the Markov Decision Process with actions that can take a variable amount of time to complete. The agent operating at the lower-most level will follow a MDP. These are integrated using the options framework [4] as shown in Figure 1.

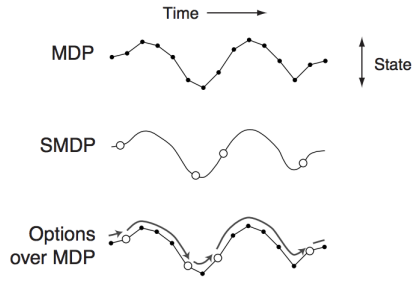


Figure 1: A depiction of the SMDP over the actual MDP with the options framework [4]

### 1.2 Deep Recurrent Q-Networks

DRQNs use a recurrent neural network in the deep Q-network to remember information for long periods of time. This kind of augmentation to the agent's DQN helps approximating the Q-value function of states at a given time, based on the values of the respective states from several steps back in time. Q-networks very well can approximate actual Q-values from sequences of observations, leading to good policies in partially observed environments as these networks can learn good internal representation of states.

As seen in Section 1.1, the larger problem that the agent should learn for a complex task is semi-Markovian. Thus, recurrent neural networks could be used to learn Q-value functions for such non-Markovian tasks, irrespective of the level of observability (full or partial). The ability of RNNs to learn long-term dependencies could be used to see if they can model higher-level temporal abstractions operating at a different time-scale.

### 1.3 Hierarchical Deep Reinforcement Learning

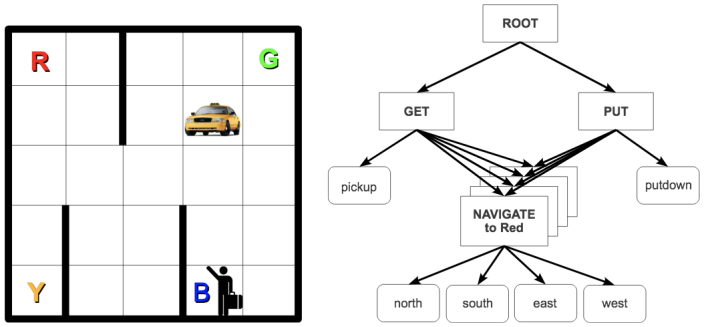


Figure 2: An example of hierarchies in the actions of the OpenAI Taxi domain [7]

By hierarchical decomposition, the task of learning can be essentially broken down into multiple reinforcement learning sub problems, each having their own value function to learn individual optimal policies. Thus, effectively the larger problem uses a hierarchical value function composed of these sub-value functions and learns a hierarchical policies that operate at different time-scales. This kind of formulation enables using a controller style architecture for using prior knowledge and for better interpretability.

In the case of the Taxi problem as shown in Figure 2, the sub-problems GET and PUT would learn policies about when to perform the pickup and putdown tasks respectively, interleaved with navigation actions. The Taxi agent would now use this hierarchical policy to execute its actions in the environment.

**Intrinsic Motivation** In environments in which rewards are sparse or delayed, the agent cannot learn by only relying on the reward signal. It needs to be equipped with ways of having some intrinsic mechanism to generate a reward and update its value function estimates, in order learn with the extrinsic rewards. It is worth noting that this setting can be viewed as a hierarchical RL problem.

## 2 Related works

Some early works that set the stage for temporal abstractions and hierarchical models for reinforcement learning are Hierarchical abstract machines (Parr et al. 1998 [8]), MAXQ (Dietterich et al. 1998 [7]), Feudal networks (Dayan Hinton 1992 [9]) and Options (Sutton, Precup Singh, 1999 [4], Precup 2000 [5]).

The success of a hierarchical deep reinforcement learning framework with intrinsic motivation setting has been demonstrated by Kulkarni et al. [2]. The described framework uses an agent that operates at two levels (meta-controller and controller) to learn different two different policies. The top meta-controller policy determines the sub-goal (goal state) to be reached and the lower-level policy carries out low-level atomic actions to reach the selected sub-goal. The controller learns a value function by receiving intrinsic rewards (pseudo-rewards) from an internal critic as shown in Figure 3. These internal rewards are very important for the controller to learn to reach the goal states. Learning with this hierarchical control emulates behavioral programs that have a call-and-return style execution like options [4].

Le et al. used this 2-level control structure adopting the recurrent architecture used in [1] to formulate hierarchical Deep Recurrent Q-Network (h-DRQN). Like [1], they also focused on using this framework in a partially-observable setting, but the ideas apply equally well to other cases. In this work, they use two variants : h-DRQN-v1 and h-DRQNv2. The v1 uses recurrent connections only in the Q-network of the meta-controller, whereas the v2 version uses them at both levels of the meta controller and the controller.

The mathematical formulations of the models that we consider are outlined in Section 3.

## 3 Models

Among the many frameworks proposed in the literature that use temporal abstractions, we specifically study three models : DRQN, h-DQN and h-DRQN to evaluate their abilities to learn good value functions and perform in complex environments.

### 3.1 DRQN

The Q-value of the state at time $t$ depicted by $s_t$ is expressed as a function of a hidden representation of the previous states upto a certain $\tau$ steps back in time, $s_{(t-\tau:t-1)}$.

$$Q_{\theta_t}(s, a) = Q(s_t, a; h_{(t-1)})$$

where, the next hidden representation is $h_t = f(s_t, h_{(t-1)}; \theta_t)$ with the information from the remaining states $s_{(t-\tau:t-1)}$ compressed into $h_{(t-1)}$.

The neural network is updated during each training phase by computing a Q-learning target given by :

$$y_t = \begin{cases} r & s' \; is \; terminal \\ r + \gamma \; max_{a'} Q_{\theta_{(t-t')}}(s', a') & otherwise \end{cases}$$

The Q-function used in the target value corresponds to a copy of the Q-network $t'$ steps back in time, also called as the *target network*. This is used to decouple the The neural network $Q$ is trained on uniformly drawn samples from the replay buffer to update the parameters at time $t$, $\theta_t$ using the loss function given by :

$$L(\theta_t) = \mathrm{E}_{(s,a,r,s') \sim D}\left[(y_t - Q_{\theta_t}(s, a))^2\right]$$

**Algorithm 1** Learning algorithm for DQN

```
Initialize experience replay memory D and parameters θ of the network Q_θ.
Initialize exploration probability ε.
for t = 1 to num_episodes do
    Initialize game and get start state description s
    while s is not terminal do
        a ← EPSILON-GREEDY(ε, s, A, Q_θ)
        Execute a and obtain next state s' and reward r from environment
        Store transition (s, a, r, s') in D
        TRAIN-NETWORK(Q_θ, D)
        s ← s'
    end while
    Anneal ε as a decreasing function of t.
end for
```

### 3.2 h-DQN

The block diagram of the 2-level operation of this model is shown in Figure 3. The overall value function learned by the agent is decomposed into two functions $Q_c$ and $Q_m$ with their respective parameters. At time $t$, the meta-controller is at state $s_t$ and computes a goal $g_t$ from its value function $Q_m$, which could be optimal or random to carry out exploration. For this time, the controller's overall state becomes a tuple $(s_t, g_t)$. It performs an exploratory/optimal action according to its value function $Q_c$ to generate next action $a_{t+i}$ that is acted upon the environment. The internal critic gives intrinsic reward $r_{t+i+1}$ to the controller for it to update its value function. This process repeats for N time-steps till the goal is satisfied by the controller, when the critic would signal a termination.

For a chosen goal $g$, an environment state $s$ and the action policy for the combination of the goal and state, $\pi_{ag} = P(a|s, g)$. The optimal value function for the controller is given by :

$$Q_c^*(s, a; g) = \max_{\pi_{ag}} \mathrm{E}\big[r_t + \gamma \, max_{a_{t+1}} Q_1^*(s_{t+1}, a_{t+1}; g) \mid s_t = s, a_t = a, g_t = g, \pi_{ag}\big]$$

For the meta-controller, the goals are essentially the 'actions' that it takes by learning a policy $\pi_g = P(g|s)$

$$Q_m^*(s, g) = max_{\pi_g} \mathrm{E}\big[\sum_{t'=t}^{t+N} f_{t'} + \gamma \, max_{g'} Q_m^*(s_{t+N}, g') \mid s_t = s, g_t = g, \pi_g\big]$$

The experiences are stored at both levels in separate replay buffers as shown in Algorithm 2. The DQN parameters are updated by optimizing the loss function used by the controller in a similar manner as described in Section 3.1 :

$$L_c(\theta_{c,t}) = \mathrm{E}_{(s,a,g,r,s') \sim D_c}\big[(y_c^{(t)} - Q_c(s, a; \theta_c^{(t)}, g))^2\big]$$

$$L_m(\theta_m) = \mathrm{E}_{(s,g,f,s') \sim D_m}\big[(y_c^{(t)} - Q_m(s; \theta_m^{(t)}))^2\big]$$

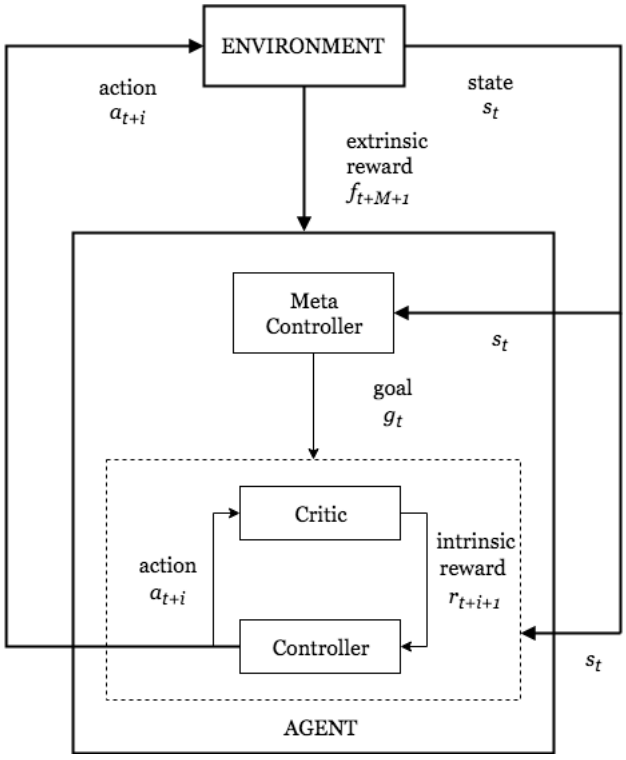


Figure 3: hDQN model

**Algorithm 2** Learning algorithm for h-DQN

Initialize experience replay memories $\{\mathcal{D}_c, \mathcal{D}_m\}$ and parameters $\{\theta_c, \theta_m\}$ for the controller network $Q_c$ and meta-controller network $Q_m$ respectively.
Initialize exploration probability $\epsilon_{c,g} = 1$ for the controller for all goals $g \in \mathcal{G}$ and $\epsilon_m = 1$ for the meta-controller.
**for** $t = 1, num\_episodes$ **do**
    Initialize game and get start state description $s$
    $g \leftarrow$ EPSILON-GREEDY$(\epsilon_m, s, \mathcal{G}, Q_m)$
    **while** $s$ is **not** terminal **do**
        $F \leftarrow 0$
        $s_0 \leftarrow s$
        **while not** ($s$ is terminal **or** goal $g$ reached) **do**
            $a \leftarrow$ EPSILON-GREEDY$(\epsilon_{c,g}, \{s, g\}, \mathcal{A}, Q_c)$
            Execute $a$ and obtain next state $s'$ and extrinsic reward $f$ from environment
            Obtain intrinsic reward $r(s, a, s')$ from internal critic
            Store transition $(\{s, g\}, a, r, \{s', g\})$ in $\mathcal{D}_c$
            TRAIN-NETWORK$(Q_c, \mathcal{D}_c)$
            TRAIN-NETWORK$(Q_m, \mathcal{D}_m)$
            $F \leftarrow F + f$
            $s \leftarrow s'$
        **end while**
        Store transition $(s_0, g, F, s')$ in $\mathcal{D}_m$
        **if** $s$ is **not** terminal **then**
            $g \leftarrow$ EPSILON-GREEDY$(\epsilon_m, s, \mathcal{G}, Q_m)$
        **end if**
    **end while**
    Anneal $\epsilon_m$ and adaptively anneal $\epsilon_{c,g}$ using average success rate of reaching goal $g$.
**end for**

### 3.3 h-DRQN

h-DRQN follows the same mechanisms of the h-DQN described in Section 3.2. It combines the formalisms of Section**??** with Section3.2.

We consider the h-DRQN model that uses recurrent connections in the deep Q-networks of both the meta-controller $Q_m$ and controller $Q_c$.

This model is also trained using the same Algorithm 2

## 4 Experiments and Results[1]

We performed our experiments with three different environments which yield rewards only under complex conditions (delayed reward setting) to suit the study of temporal abstractions.

The first two environments are variants of the Discrete Stochastic Decision Process (DSDP) considered in [2]. The third environment is a recast of the game *Montezuma's Revenge* into a simple and small grid world.

The DQNs we consider for both the cases are MLPs with 2-hidden layers with ReLU activation. For DRQN and h-DRQN, the first layers of the MLPs had RNN units. The networks were trained using RMSprop optimizer with a smooth L1 loss (Huber Loss) as indicated by [6].

[1]The code is available at `https://github.com/pravishsainath/HDRL-Experiments`.

The set of possible goals $\mathcal{G}$ were considered to be the set of states $\mathcal{GS}$. The internal critic reward function that penalized -1 every step without reaching the goal state and +1 if it is reached. Formally :

$$r_t(s_t + i, g_t) = \begin{cases} +1 & s_t = g_t \\ -1 & otherwise \end{cases}$$

The performance of the three models described in Section 3 in each of these three environments are compared and analyzed below.

### 4.1 Discrete Stochastic Decision Process (DSDP)

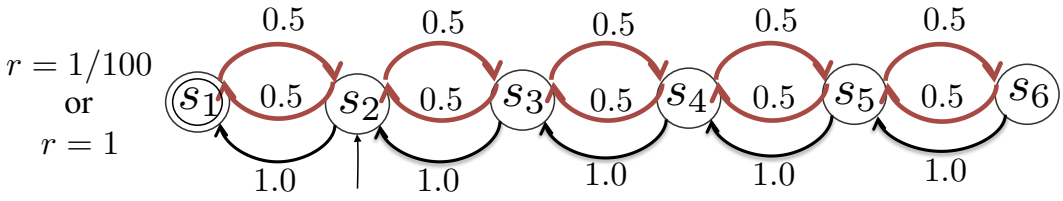


Figure 4: A discrete stochastic decision process (from [2]) that starts from state $s_2$ and terminates at state $s_1$ giving a reward of 1 if a condition is satisfied and $1/100$ otherwise
**2-DSDP condition** : should have visited $s_6$ at least once
**3-DSDP condition** : should have visited $s_6$ at least once and $s_4$ at least twice after visiting $s_6$

**Setup** The 6-state DSDP is shown in Figure 4 has a terminal (accepting) state $s_1$ and a starting state $s_2$. The reward is obtained only at the terminal state and the value of the reward depends on the satisfaction of a specific condition about the history of the states visited by the agent. The conditions are different for each of the two DSDPs and they are described below in their respective subsections and also in Figure 4
The environment supports a deterministic *left* action and a stochastic *right* action succeeding with probability 0.5 (it goes *left* otherwise).
If the agent reaches the terminal state $s_1$ without visiting the states in the order imposed by the respective condition in each case, the episode ends with a reward of $0.001$. In the case when the condition is satisfied, it receives a higher reward of $1$.

#### 4.1.1 2-level Discrete Stochastic Decision Process (6-DSDP)

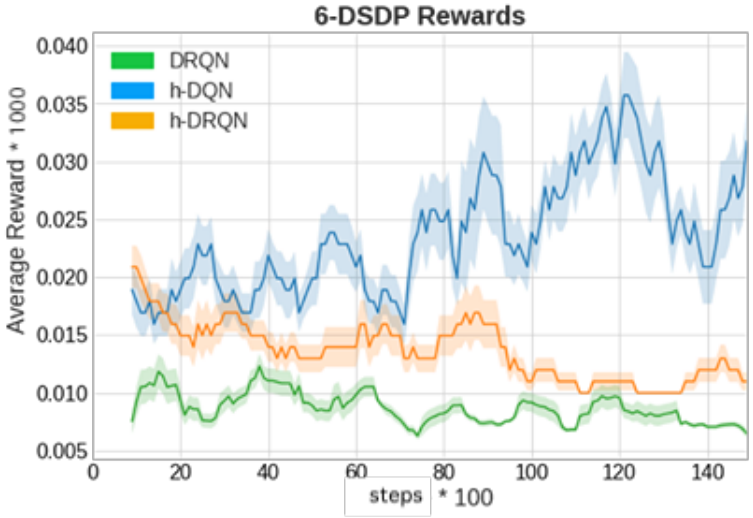


Figure 5: Average rewards from 6-DSDP for agents trained with the three frameworks

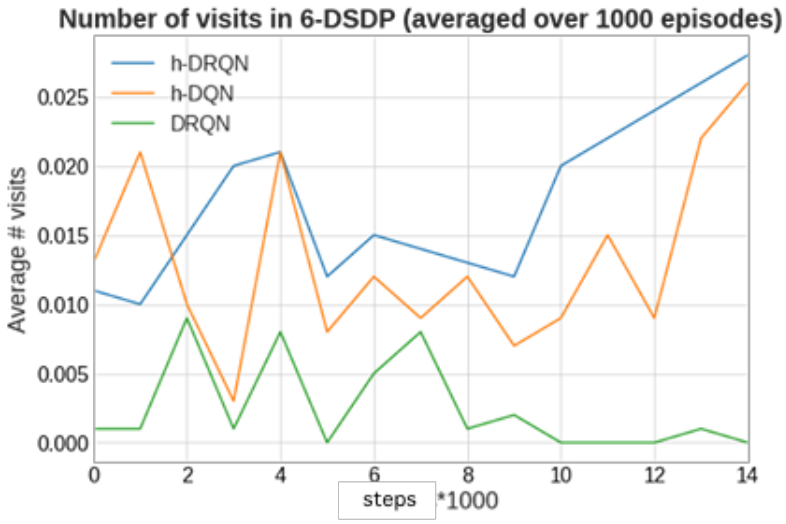


Figure 6: The average number of visits to state $s_6$ in 6-DSDP

**Results** From Figiure 5, it is evident that DRQN performs poorly as the reward accumulated by it near zero. The DRQN however fares slightly better than the simple Q-learning used in [2] is due to its richer 'memory' capacity as it models the Q-value functions at each time-step as a function of past Q-values.

The h-DQN and h-DRQN agents have consistently increased accumulating their rewards and this scale is way higher compared to the DRQN. These two agents should have converged to their optimal value functions. But, among these two, h-DRQN has learned the optimal policy very early enough to figure out the path to the higher reward.

This can be verified in Figure **??**, where the visit countes to state $s_6$ for the h-DRQN and h-DQN agents is increasing with time-steps but for DRQN, it is decreasing. So, DRQN was unable to find the path to the higher reward while the two other models were able to.

#### 4.1.2 3-level Discrete Stochastic Decision Process (6,4-DSDP)

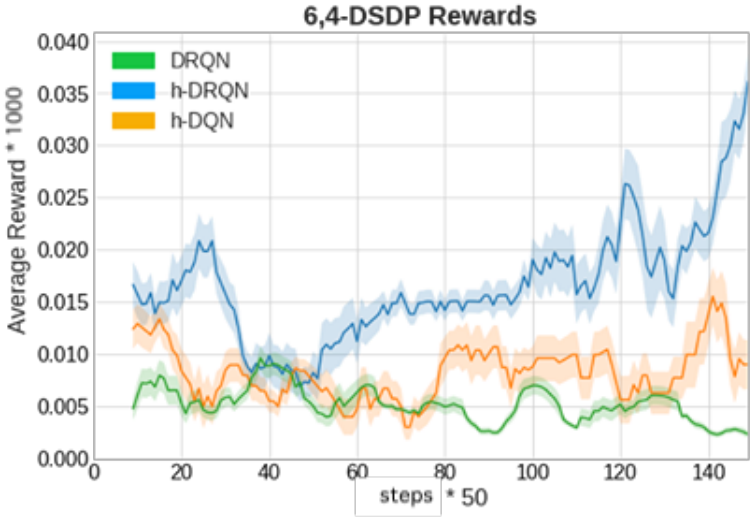


Figure 7: Average rewards from 6-DSDP for agents trained with the three frameworks

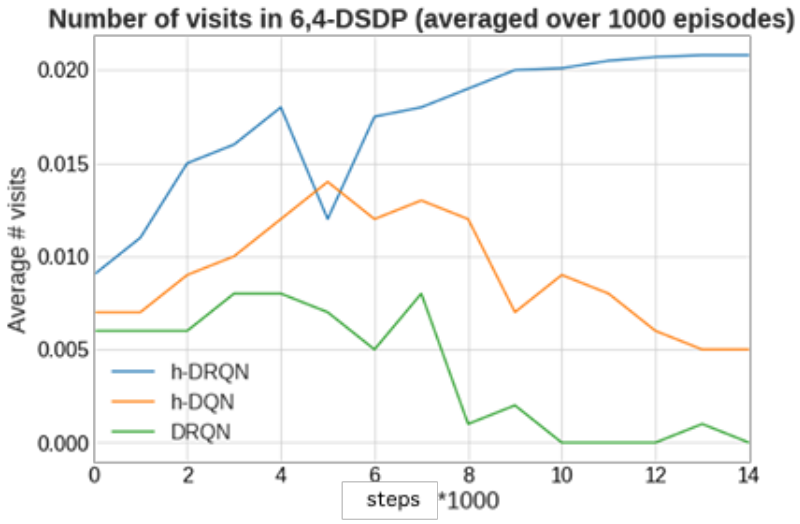


Figure 8: The average number of visits to state $s_4$ from 64-DSDP

**Results** For this DSDP, there is a significant reduction in performance for all the 3 models. This is due to the fact that in order to get receive the reward from the environment, more states need to be visited in certain order.

The DRQN clearly is unable to learn the optimal policy as the capacity of its recurrent layer to recall past states is limited. The performance of hDQN is also sub-optimal indicating that its meta-controller might not have been able to generate good goal states for the controller. The h-DRQN agents performs well compared to the other agents and it has converged to a good value function.

This is supplanted by the plot in Figure 8, which shows that the h-DRQN agent visited state $s_4$ increasingly as it has figured out the pattern to get the higher reward. But comparing with 6-DSDP, the h-DRQN agent seems to have learned the optimal policy much later than it could. It is understandable as it has to visit a longer sequence of states in this case. While the value is decreasing for the other two agents as they could not figure it out.

### 4.2 Simplified Montezuma's Revenge (SMR)

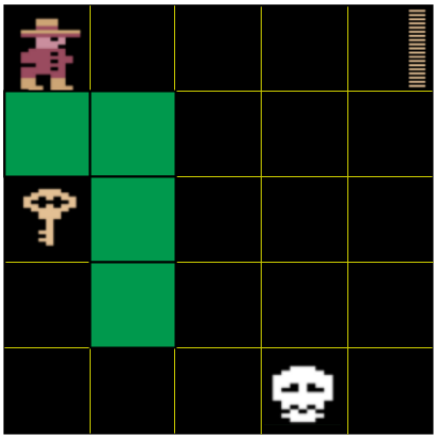

Figure 9: A sample screen from the Simplified Montezuma's Revenge (SMR) environment with the door, key and ghost

**Setup** The environment considered was a mock environment of the classic ATARI game *Montezuma's Revenge*. As we were mostly interested in studying the ability of different agents to perform in a complex goal-directed task, we eliminated the complexity due to the pixel space.

This simplified environment reduces to a grid world domain with multiple goals and sparse rewards that is used in many experiments in reinforcement learning. A snapshot of the environment at a certain point of time is shown in Figure 9.

There is one state in the grid with the key that is blocked by walls around it, forcing only one way of entering near the ghost. The ghost moves freely in the bottom grids with equal probability at each step. Procuring the key gives a reward of +100 and the reaching the door state with the key ends the episode with a reward of +400. The episode times out in atmost 20 time-steps.

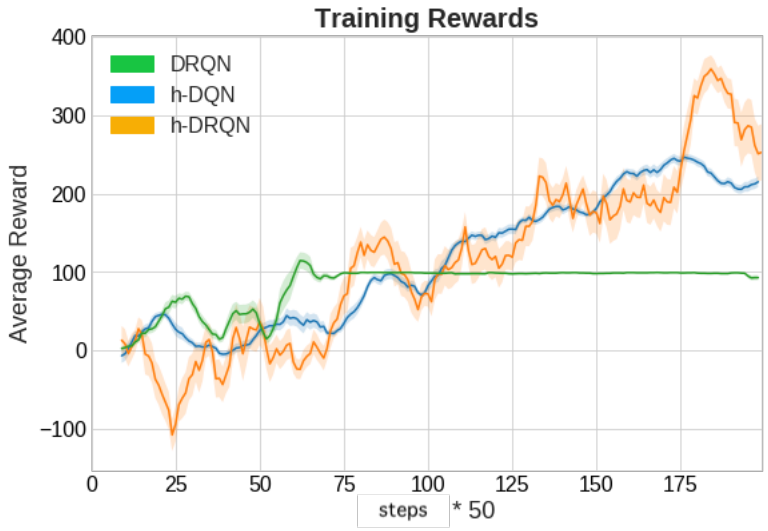


Figure 10: Average rewards from 6-DSDP for agents trained with the three frameworks

**Results** All the three agents capture some long-range dependencies to make progress in the task.

It can be seen that the DRQN agent is able to locate the key, but the curve remains almost static with that value indicating that it was unable to make any further progress and could not discover the exit door. The curves of the h-DQN and h-DRQN agents indicate a progression with increasing timesteps, meaning that these agents were able to solve the challenge by making the player to pick the key and exit through the door.

However, it can be remarked that h-DRQN overall performs slightly better than h-DQN. Both these models performs way better than DRQN.

## 5 Conclusion

We understood temporal abstractions and hierarchical deep reinforcement learning, specifically using them in an intrinsic motivation setting. We studied and implemented several frameworks : DRQN, h-DQN and h-DRQN. From our study and experiments, we conclude the following :

- The ability of a recurrent neural network mechanism to model temporal abstractions as temporal dependencies over states in a deep Q-network is *not sufficient* to perform optimally in multiple-level goal-directed tasks.
- Introducing a hierarchical structure into the deep reinforcement learning problem by training deep Q-networks at different hierarchical levels of control gives the models the ability to learn to solve the task at different time-scales. This happens because, with this framework, the agent is able to perform much better exploration that is extended over time.
  This is, by virtue of these temporal abstractions, the agent is able to maintain the 'context' of exploration in a much better manner than a *flat* recurrent deep Q-network case.
- In addition to the levels of control, intrinsic motivation used in the hDQN and hDRQN frameworks with the help of an internal critic, drives the agent to efficiently carry out such an exploration. The design of the intrinsic reward function is a crucial factor that can impact exploration and learning.
- Hierarchical deep reinforcement learning models can remarkably improve learning and performance if the hierarchy (hierarchical depth) is sufficient enough to model the sub-goals of the task under the condition that good sub-goals have been discovered.
  The task of sub-goal discovery is crucial for ensuring that HRL agents perform optimally.
- Deep reinforcement learning models with hierarchical deep recurrent Q-networks for action-value function approximation to learn over multiple time-steps makes them very effective in capturing the temporal abstractions and hence perform very well, turning out to be the best among the models considered.

We, thus conclude that the combination of hierarchical agent structure and recurrent deep Q-networks are powerful to learn the long-range temporal abstractions very well, compared to using them individually as stated in the literature.

**Acknowledgments**

We would like to express our gratitude to Google Inc. for the free GPU resource provided on Google Colab environment. We thank our instructor Dr. Doina Precup for the lectures on temporal abstraction that sparked our interest in the topic and the entire team of CS-767 for giving an opportunity to learn and explore ideas in reinforcement learning.

## References

[1] Matthew Hausknecht & Peter Stone (2015) Deep Recurrent Q-Learning for Partially Observable MDPs *Proceeding of Association for the Advancement of Artificial Intelligence Conference* arXiv:1507.06527

[2] Kulkarni, T. D. Narasimhan, K. R. Saeedi, A. & Tenenbaum, J. B. (2016) Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation, *Advances in Neural Information Processing Systems*, pp. 3675–3683.

[3] Tuyen P. Le Ngo Anh Vien & TaeChoong Chung (2018) A Deep Hierarchical Reinforcement Learning Algorithm in Partially Observable Markov Decision Processes, *IEEE Access*, vol. 8, pp. 49089–49102.

[4] R. S. Sutton, D. Precup, & S. Singh (1999) Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning, *Artificial Intelligence*, vol. 112, pp. 181–211.

[5] Precup, Doina (2000) Temporal Abstraction in Reinforcement Learning

[6] Volodymyr Mnih Koray Kavukcuoglu David Silver Alex Graves Ioannis Antonoglou Daan Wierstra Martin Riedmiller (2014) Playing Atari with Deep Reinforcement Learning

[7] T. G. Dietterich (2000) Hierarchical reinforcement learning with the MAXQ value function decomposition, *Journal of Artificial Intelligence Research*, vol. 13, pp. 227–303.

[8] R. E. Parr & S. J. Russell, (1998) Reinforcement learning with hierarchies of machines, *Advances in Neural Information Processing Systems*, pp. 1043–1049.

[9] P. Dayan & G. E. Hinton, (1993) Feudal reinforcement learning, *Proceedings of Advanced Neural Information Processing Systems*, pp. 271–278.

[10] R. E. Parr & (1998) Hierarchical control and learning for Markov decision processes, Ph.D. dissertation, Dept. Comput. Sci., Univ. California, Berkeley, Berkeley, CA, USA.

[11] A. G. Barto & S. Mahadevan, (2003) Recent advances in hierarchical reinforcement learning *Discrete Event Dynamical Systems : Theory and Applications*, vol. 13, pp. 41–47.

[12] Richard S. Sutton and Andrew Barto, *Reinforcement Learning : Second Edition*, The MIT Press