Recurrent Networks in Hierarchical Deep Reinforcement Learning

Ifigeneia Apostolopoulou
Machine Learning
Carnegie Mellon University
Pittsburgh, PA 15213
`iapostol@andrew.cmu.edu`

Abstract

Learning goal-directed behavior with sparse feedback from complex environments is a fundamental challenge in artificial intelligence. A hierarchical structure of two learning modules which operate under different time-scales, one of which represents the agent's intrinsic motivation, can encourage sufficient exploration to overcome this challenge. Adding memory in the learning modules so that the agent can cope with longer dependencies, can further facilitate learning.

1 Introduction

Deep reinforcement learning has been shown effective in playing various Atari games due to the powerful neural network function approximator. However, some games may have sparse environmental feedback, e.g., 'Montezuma's Revenge'. To deal with sparse feedback, Kulkarni [1] proposed a hierarchical deep reinforcement learning approach. In order for the agent to be able to deal with the sparse rewards provided by a complicated environment, it has to define intrinsic rewards which facilitate exploration. Consequently a sequence of goals should be learned to achieve the final extrinsic reward which is sparse. Therefore, the entire problem is decomposed to learning two policies; the one is related to choosing the next goal which has to be realized (based on the environmental reward). Given the next goal, a second policy which is choosing the next action to be taken, has to be learned based on the internally defined rewards. The concept of hierarchical deep reinforcement learning is also explored in other works, e.g., [3] in which they designed a separate deep skill network. Hausknecht [2] proposed to replace the first fully connected layer in the convolutional neural network with a Long Short Term Memory (LSTM) to deal with a Partially Observable Markov Decision Processes (POMDPs) environments in which case the environmental state cannot be fully represented. In this setting, the resulting Deep Recurrent Q Network(DRQN) outperformed the previously proposed Deep Q Network (DQN). In the hierarchical DQN (h-DQN), the inclusion of a recurrent network could further improve the learning performance of the agent by learning the longer range dependencies if the history of the goals and/or the sequence of the frames is also considered. Accordingly, in this project we propose to combine both hierarchical deep reinforcement learning and deep recurrent Q-learning to facilitate learning in a POMDP environment with sparse feedback.

2 Model and Method

2.1 Background: Deep Reinforcement Learning -The DQN Architecture

A Markov Decision Process (MDP) is defined as a tuple (S, A, T, R, γ). At each timestep t an agent interacting with the MDP observes a state $s_t \in S$, and chooses an action $a_t \in A$ which determines the reward $r_t \sim R(s_t, a_t)$ and next state $s_t \sim T(s_t, a_t)$. Reinforcement Learning [6] is concerned with learning the control policy of an agent $\pi(a_t|s_t)$ which defines the probability of executing action a_t in s_t while interacting with the unknown environment. The Q-value function given a policy π is defined as $Q^{\pi}(s,a) = E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots | s_t = s, a_t = a, \pi]$ and represents the expected future discounted reward when starting in a state s, executing a, and then following policy π until a terminal state is reached. The optimal state-value function $Q^*(s,a)$ is defined as $Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a), \forall a \in A, \forall s \in S$. The optimal Q-function can be rewritten as a Bellman equation:

$$Q^*(s,a) = E_{s_{t+1} \sim T(.|s_t,a_t)}[r_t + \gamma max_{a_{t+1} \in A} Q^*(s_{t+1}, a_{t+1})] \tag{1}$$

An optimal policy can be constructed from the optimal Q-function by choosing, for a given state, the action with the highest Q-value. Q-learning [7] is a model-free off-policy algorithm for estimating the long-term expected return of executing an action from a given state. In many environments, the state space is too large to tractably store a separate estimate for each $S \times A$. In the case of Deep Q-Learning [4],[5], the Q-function is parametrized by a neural network and its values are estimated by querying the output nodes of the network after performing a forward pass given a state input. Such Q-values are denoted by $Q(s,a|\theta)$, where θ the weights of the network. Instead of updating individual Q-values, updates are now made per iteration i, on the parameters of the network in order to minimize a differentiable loss function:

$$L(s,a|\theta_i) = (r + \gamma \max_{a_{t+1} \in A} Q(s_{t+1}, a_{t+1}|\theta_i) - Q(s,a|\theta_i))^2 \tag{2}$$

$$\theta_{i+1} = \theta_i + \alpha \nabla_{\theta} L(\theta_i) \tag{3}$$

,where α the learning rate. However, because the same network is generating the next state target Q-values that are used in updating its current Q-values, such updates can diverge [7]. To avoid this, the DQN (Deep Q-Network) approach employs two techniques. First, experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ are recorded in a replay memory M and then sampled uniformly during the training time. Second, two networks are actually maintained in the architecture. The online network Q (whose weights are represented by θ_i from now on) which is used for the weight updates in Equation 3, and the target network $\hat{Q}$ (whose parameters are represented by θ_i^- from now on) , which generates the target values $r_t + \gamma \max_{a_{t+1} \in A} \hat{Q}(s_{t+1}, a_{t+1}|\theta_i^-)$ in Equation 2. The target network is only periodically updated and set equal to the online network, thereby reducing correlations between the target and current Q-values. Therefore, the loss function which summarizes the two previous changes can then be stated as:

$$L(\theta_i) = E_{(s_t,a_t,r_t,s_{t+1}) \sim M}[(r_t + \gamma \max_{a' \in A} \hat{Q}(s_{t+1}, a'|\theta_i^-) - Q(s_t, a_t|\theta_i))^2] \tag{4}$$

and its gradient, which can then be replaced in Equation 3 in order to update the online network's parameters is given by the equation below:

$$\nabla L(\theta_i) = E_{(s_t,a_t,r_t,s_{t+1}) \sim M}[(r_t + \gamma \max_{a' \in A} \hat{Q}(s_{t+1}, a'|\theta_i^-) - Q(s_t, a_t|\theta_i)) \nabla Q(s_t, a_t|\theta_i)] \tag{5}$$

2.2 Model with Temporal Abstractions -The h-DQN Architecture

Learning and operating over different time-scales is a key challenge in tasks involving long-term planning [9], [10], [11], [12]. In addition, the concept of intrinsic motivation is incorporated into the computational theory of Reinforcement Learning [13], [14], [15], [16] to address the lack or sparsity of external rewards so that the agent continues to explore and learn in that kind of environments.

The proposed framework in [1] uses two hierarchically organized DQNs over different levels of temporal abstraction. The top-level module (the meta-controller) chooses a new goal g_t given the environmental state s_t. The low-level module (the controller) uses both the state s_t and the chosen goal g_t to select the next action s_t either until the goal is reached or the episode is terminated. The meta-controller then chooses another goal and the previously described procedure is repeated. The internal critic is responsible for evaluating whether a goal has been reached and providing an appropriate reward r_t to the controller. The controller has to maximize the cumulative, discounted intrinsic reward. On the other side, the meta-controller should chose the next goal so that it maximizes the cumulative extrinsic reward, based on the reward signals f_t that it receives from the environment. As in [11], a different policy $\pi_{ag} = P(a|s, g)$ is defined for each goal g. However, instead of preserving a different Q-value function for each goal, the goal is treated as part of the input of the controller's network, similar to what followed in [12]. In a stricter mathematical formulation, the controller's network estimates the following Q-value function:

$$Q_1^*(s, g, a) = \max_{\pi_{ag}} E[r_t + \gamma max_{a_{t+1} \in A} Q_1^*(s_{t+1}, g, a_{t+1}) | s_t = s, a_t = a, g_t = g, \pi_{ag}] \quad (6)$$

, and similarly, the meta-controller's network estimates the

$$Q_2^*(s, g) = \max_{\pi_g} E[\sum_{t'=t}^{t+N} f_t' + \gamma max_{g' \in G} Q_2^*(s_{t+N}, g') | s_t = s, g_t = g, \pi_g] \quad (7)$$

, where N denotes the number of time-steps until the controller halts given the current goal set by the metacontroller, and $\pi_g = P(g|s)$ is the policy over the goals. It should be highlighted that the transitions (s_t, g_t, f_t, s_{t+N}) generated by Q_2 run at a slower rate than those $(s_t, a_t, g_t, r_t, s_{t+1})$ of Q_1.

2.3 Model with Temporal Abstractions and Memory -The h-RDQN Architecture

The h-DQN structure described in the previous section is limited in the sense that the metacontroller learns only from the previous state and goal that was set. In real-world environments a sequence of goal has to be learned until a sparsely distributed environmental reward is obtained. For example and based on the MDP environment described in the following section, if the agent is in state s_3 and just accomplished the previously picked goal, when deciding the next goal, it will be helpful to be aware of whether the bonus state (and the corresponding goal) has been previously met (something that may have occured many states before) , information which is not conveyed only by the current goal and state in order to decide whether it should move right (to get closer to the bonus state) or left (to get closer to the terminal state). Incorporating memory in the metacontroller's network will help the agent distinguish the two cases.

Figure 1 shows the structure of our proposed hierarchical deep recurrent reinforcement learning. In this figure, the meta-controller is in charge of choosing the next goal given the environmental rewards. On the other hand, the controller has to select the next action in order to achieve the goal indicated by the meta-controller by using the intrinsic rewards. In the proposed scheme, the last layer of the neural network used in the meta-controller's network is replaced with a LSTM (Long Short Term Memory) layer. The LSTM in the meta-controller will handle the longer range dependencies in the history of previous goals. As it corroborated by the experiments presented later in this report, the metacontroller indeed learns much faster the correct sequence of goals.

Finally, training a recurrent deep network, requires each backward pass to contain many goal-steps encountered. The parameters can be updated either sequentially (full episodes are sampled from the metacontroller's memory while the LSTM's hidden state is preserved) or randomly (the samples are drawn from random points within an episode, in which case the hidden state of the network is zeroed). The sequential update may compromise the DQN's random sampling policy while the random updates make it harder for the LSTM to learn long goal sequences.

3 Experimental Results

3.1 Experimental Setup

For the evaluation of our models, we consider a stochastic decision process as the one described in [1]. More specifically, the discrete-state MDP (Markov Decision Process) is characterized by delayed rewards, stochastic transitions and the sparse extrinsic reward depends on the history of the visited states. There are 6 possible states and the initial state is s_1. The actions are *move left*, and *move right*. The *move left* action is deterministic while the *move right* succeeds with probability 0.5. The terminal state is s_0. The award received is 1 if the agent has visited s_5 before reaching s_0 and 0.01 otherwise.

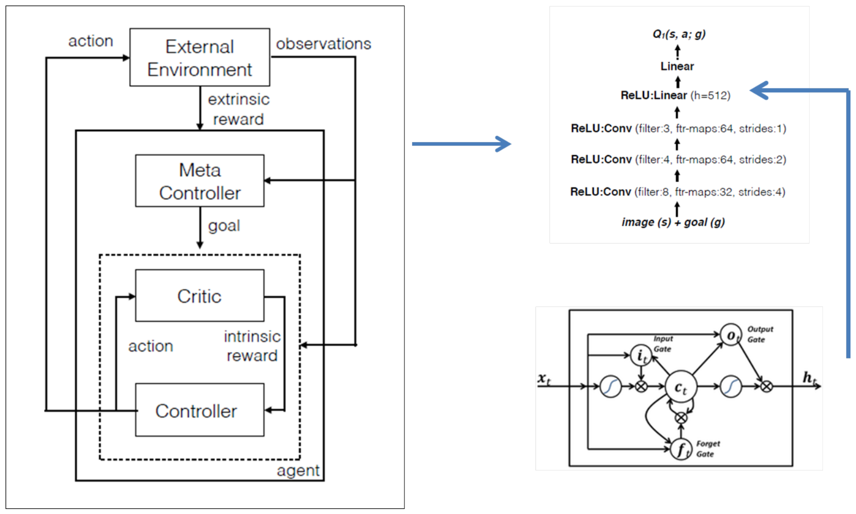


Figure 1: Structure of proposed hierarchical deep recurrent reinforcement learning.

In all of the experiments, a deep neural network did not affect the performance of each approach; the DQN (Deep Q-learning Network) architecture was still unable to learn to reach s_5 in order for the agent to receive larger reward, while for both h-DQN architectures (with and without the LSTM) the testing reward was nearly the same when a deep network was used for the controller's and/or the metacontroller's network. Therefore, in this report extensive results achieved by linear networks are included. Moreover, we adopted the variation of the Q-Learning algorithm of [4],[5] which maintains two Q-networks (the online and the target network) as described earlier. The batch size was set to be 64. The size of the replay memory was 1000000 and the memory was burnt in after 30000 episodes. A uniform random policy was used during the burn-in phase of the replay buffer. For the evaluation, an $\epsilon-$greedy policy was adopted with $\epsilon = 0.01$. During the training, an $\epsilon-$greedy policy was considered where the exploration probability was adaptively annealed (see each case for a detailed description). The learning rate was always set to $\alpha = 0.00025$ and the discount factor was $\gamma = 0.99$. The online network was updated after each new interaction with the environment, once the agent has exited the burn-in phase. The target network was synced after 1000 updates of the online network. These parameters refer to all networks; both the controller and the metacontroller. Any discrepancy is otherwise noted per case.

In all cases, we include the loss function after each update of the online network. The network is evaluated every 1000 training episodes of training for 10 testing episodes. The progress realized by the learning algorithm is demonstrated by the average length of the episode (the agent prefers to explore other states first before reaching s_0) and the total extrinsic reward. A reward of 0.1 implies that in all of the testing episodes, the agent failed in reaching the bonus-state s_5. Finally, the total number of visits of states s_3, s_4 and s_5 during the last 1000 episodes of training are also illustrated.

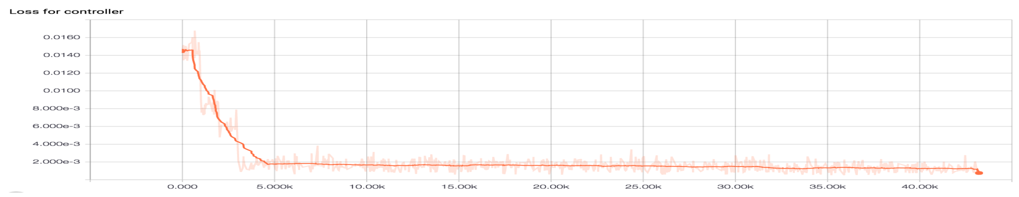
In the implementation, the high-level neural networks API Keras was used, which was running on top of Tensorflow.

3.2 Baseline: Off-Policy Temporal Difference Learning (Q-Learning)

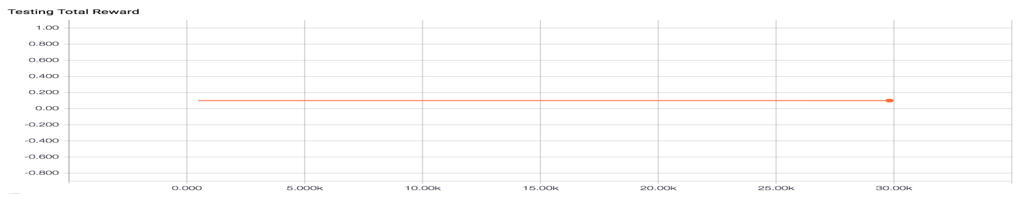
As it can be derived from Figure 2 , the vanilla Q-learning architecture does not succeed in discovering the bonus of reaching state s_5. The total testing reward is always 0.1 and the episode length is always 1, since the agent directly jumps in the terminal state. During the training, the number of times state s_5 is visited, is decreasing and it is initially high due to the high exploration probability of the agent. The ϵ in the linearly decayed policy was annealed from 1.0 to 0.1 across 50.000 steps.

3.3 Milestone 1: Off-Policy Temporal Difference Learning with Intrinsic Motivation (Hierarchical Q-Learning)

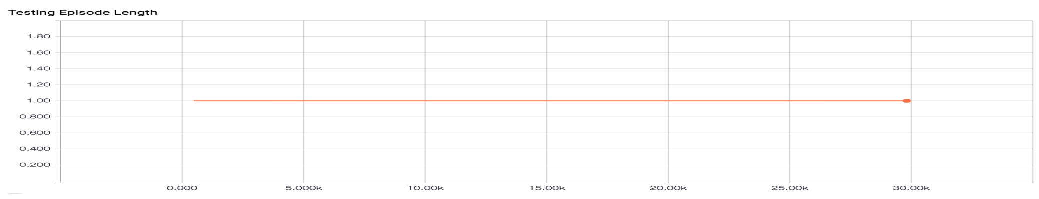
Figure 3 shows the performance of the hierarchical structure. As it can be concluded, after roughly 8000 episodes, the agent obtains consistently total testing award of 1.0 or higher (which equivalently



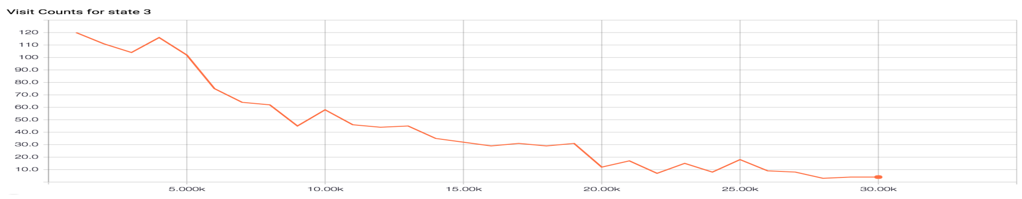
(a) The loss function for training the network. The x-axis represents the number of network updates.



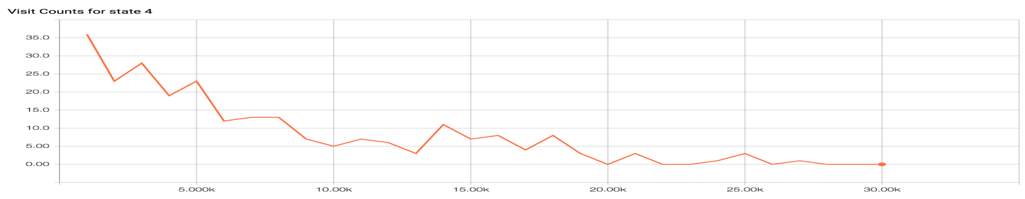
(b) The total testing reward achieved for 10 testing episodes.



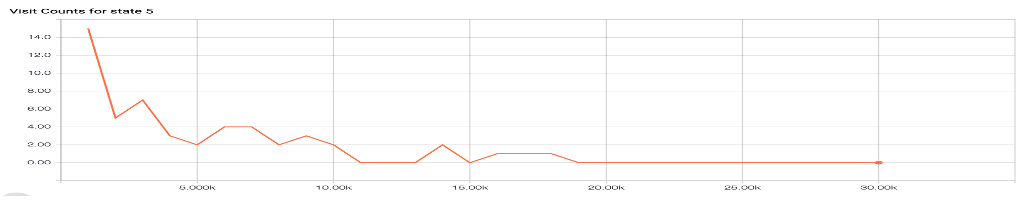
(c) The average episode length across 10 testing episodes.



(d) The number of 3-state visits of the SDP in the last 1000 training episodes.



(e) The number of 4-state visits of the SDP in the last 1000 training episodes.



(f) The number of 5-state visits of the SDP in the last 1000 training episodes.

Figure 2: Performance of the Q-Learning Algorithm. Unless otherwise noted, the x-axis represents the total number of training episodes. The network is evaluated every 1000 training episodes.

means that in at least one run of the game, it visits the bonus state). Correspondingly, the episode length becomes higher than 2. This implies that the agent attempts to reach s_6, although due to the stochasticity of the *move right* action, it may eventually fail. The number of agent's visits at states s_3, s_4, s_5 is increasing and after around 20000 episodes of training, it plateaus.

The policy of the metacontroller, which is responsible for the selection of the next goal, is adaptively annealed from 1.0 to 0.1 across 50000 updates of the metacontroller's network. One point that should be highlighted, and which may not be obvious from the description of the hierarchical model, is that a separate $\epsilon-$greedy policy was considered for each one of the goals in the controller's network.The intuition behind this, is that for a given goal the environment may have been sufficiently explored which may not be the case for a different goal. Therefore, the exploration probability for a goal g, ϵ_g was annealed using the average success rate of reaching goal g. This is equivalent to keep exploring if the goal has not been encountered sufficiently many times.

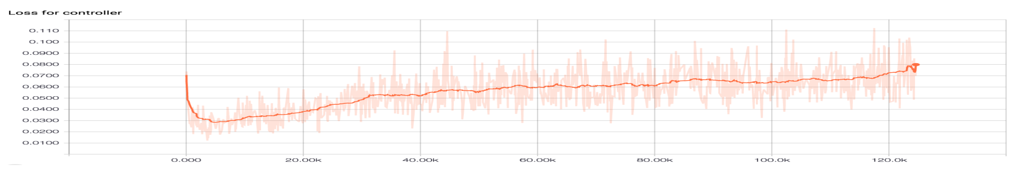
3.4 Milestone 2: Off-Policy Temporal Difference Learning with Intrinsic Motivation and Long Short-Term Memory (Recurrent Hierarchical Q-Learning)

We have augmented the architecture described in Section 3.2, in order to incorporate LSTM (Long Short Term Memory) in the network of the metacontroller. More specifically, its network consists of a layer of a stateful LSTM with 4 hidden units, and a ReLU (Rectified Linear Unit) activation function. A linear layer outputs the Q-value for each action of the agent. Bootstrapped Sequential Updates where used for the training of the recurrent architecture. Until a batch of 64 samples was gathered from the replay buffer, a new full episode was obtained. The LSTM'S hidden state were preserved forward throughout the training episode and was zeroed among different training or testing episodes. The rest of the architecture's parameters are the same with those presented in Sections 3.2 and 3.3.

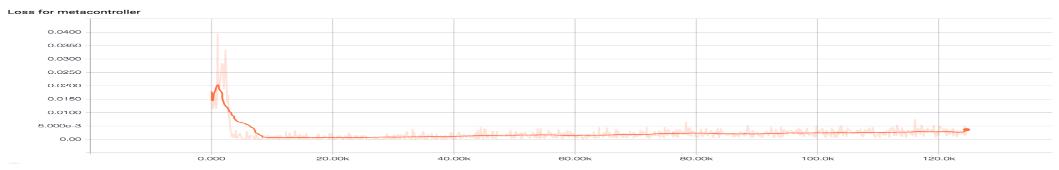
As we can see in Figure 4, the agent indeed learns faster than the one with no LSTM in it's metacontroller network. More specifically, the agent learns to reach s_5, and achieve total testing reward 1.0 or higher after around 2000 episodes (which is significantly faster than the learning performance presented in Section 3.2 according to which the agent learnt to reach s_5 after 8000 training episodes). However, the agent is incapable of significantly increasing its testing total reward. The previously mentioned architecture was the only one explored for the scope of this project. A different number of hidden units, type of activation function or a memory for history not only of goals but also actions could yield better results. We also expect a larger difference in the performance of h-DQN and h-RDQN for more difficult environments (such as a longer stochastic decision process or the 'Montezuma's Revenge' ATARI game). Finally, the recurrent metacontroller exhibits larger learning instability (i.e observe the oscillations in the number of visits in states s_3, s_4, s_5). This can be due to the sequential updates. It is expected that a larger batch size (which corresponds to greater number of episodes randomly picked from the replay) could mitigate these effects.

4 Conclusion

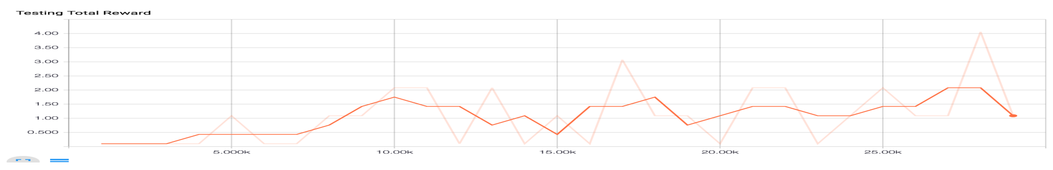
In this project, we have evaluated the performance of three different Reinforcement Learning architectures in environments with sparse feedback. The fixed-target DQN agents does not succeed in sufficiently exploring the environment and reaching the state which will eventually yield higher extrinsic reward after a sequence of states visited. The temporal decomposition of the value function and the incorporation of intrinsic motivation manages to solve this problem. More specifically, it encourages the agent's exploration for its own sake, regardless the reward received from the environment, and it eventually helps him solve the problem posed by the environment too. Finally, adding recurrency to the metacontroller's network helps the agent to determine the right, and potentially long sequence of goals that have to be accomplished before receiving the delayed rewards, and eventually learn significantly faster. The difference in the learning rate is expected to be larger in more complex or difficult environments. However, some stability issues in the learning performance of the agent have to be tackled. Finally, evaluating the performance of the h-rdqn architecture on notoriously difficult ATARI games such as the 'Montezuma Revenge' is a further extension of this project.



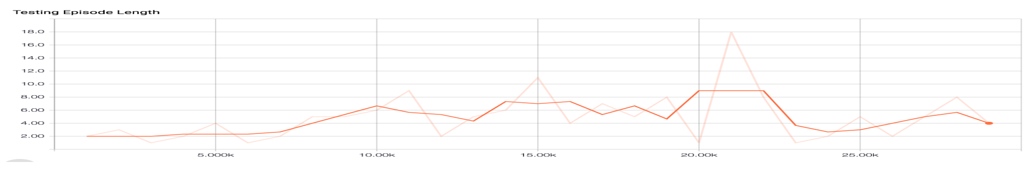
(a) The loss function for training the controller's network. The x-axis represents the number of network updates.



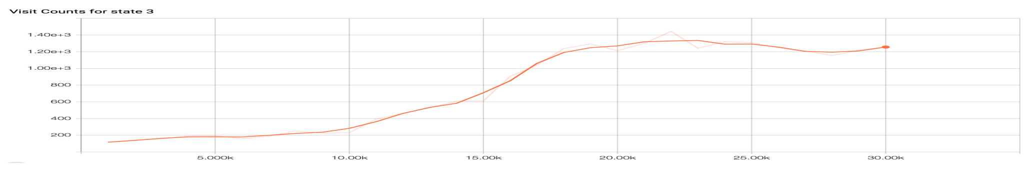
(b) The loss function for training the metacontroller's network. The x-axis represents the number of network updates.



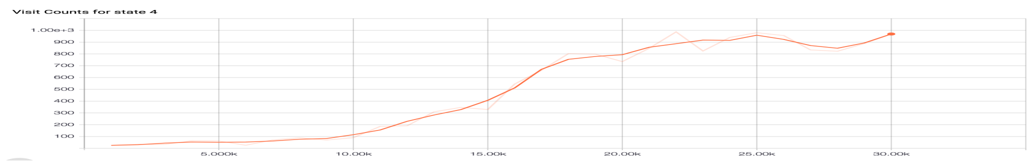
(c) The total testing reward achieved for 10 testing episodes.



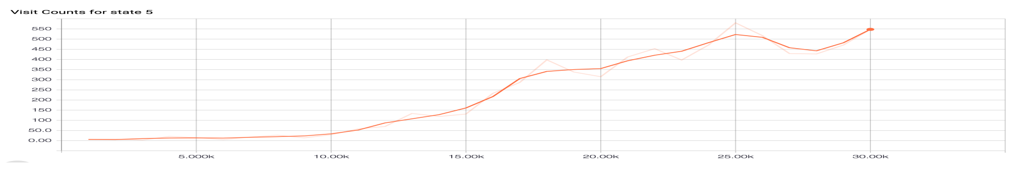
(d) The average episode length across 10 testing episodes.



(e) The number of 3-state visits of the SDP in the last 1000 training episodes.

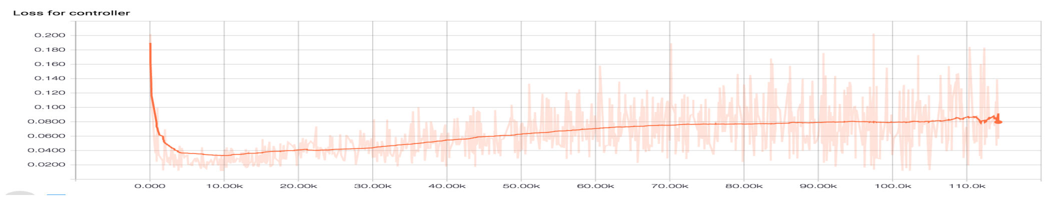


(f) The number of 4-state visits of the SDP in the last 1000 training episodes.

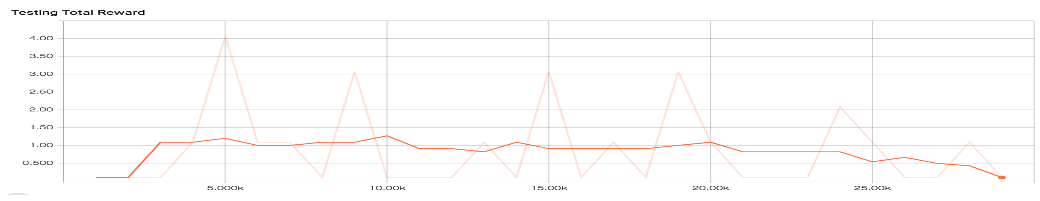


(g) The number of 5-state visits of the SDP in the last 1000 training episodes.

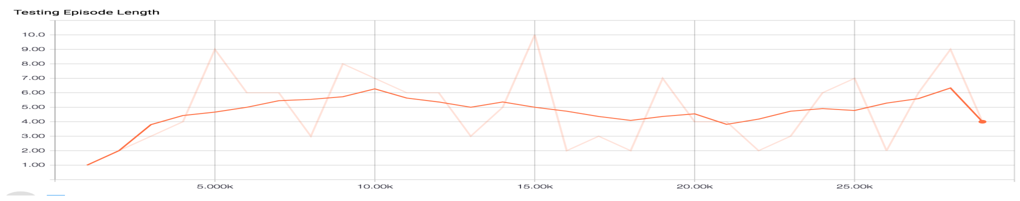
Figure 3: Performance of the Hierarchical Q-Learning Algorithm. Unless otherwise noted, the x-axis represents the total number of training episodes. The network is evaluated every 1000 training episodes.



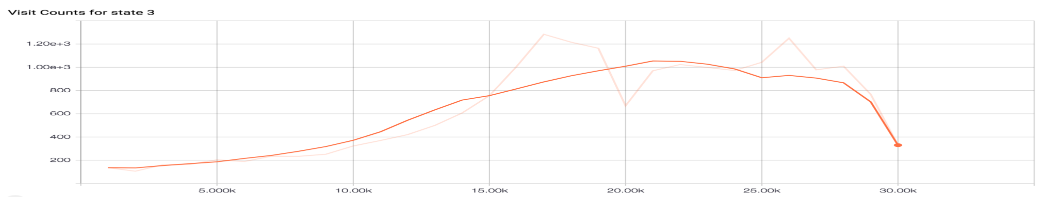
(a) The loss function for training the controller's network. The x-axis represents the number of network updates.



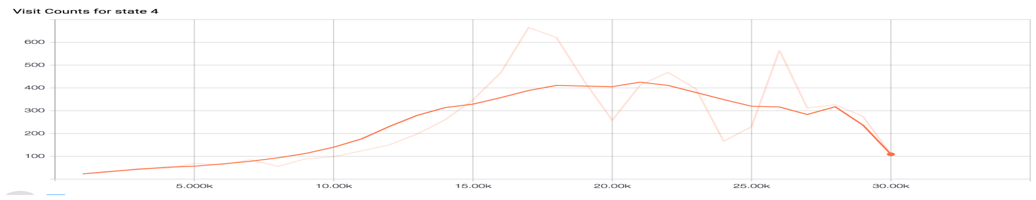
(b) The total testing reward achieved for 10 testing episodes.



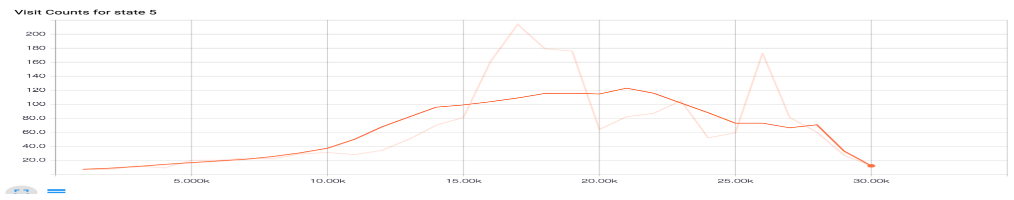
(c) The average episode length across 10 testing episodes.



(d) The number of 3-state visits of the SDP in the last 1000 training episodes.



(e) The number of 4-state visits of the SDP in the last 1000 training episodes.



(f) The number of 5-state visits of the SDP in the last 1000 training episodes.

Figure 4: Performance of the hierarchical Q-Learning Algorithm with a recurrent network. Unless otherwise noted, the x-axis represents the total number of training episodes. The network is evaluated every 1000 training episodes.

5 References

[1]Kulkarni, Tejas D., et al. "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation." Advances in Neural Information Processing Systems. 2016.

[2]Hausknecht, Matthew, and Peter Stone. "Deep recurrent q-learning for partially observable mdps." arXiv preprint arXiv:1507.06527 (2015).

[3]Tessler, Chen, et al. "A deep hierarchical approach to lifelong learning in minecraft." arXiv preprint arXiv:1604.07255 (2016).

[4]Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M., 2013. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

[5]Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G. and Petersen, S., 2015. Human-level control through deep reinforcement learning. Nature, 518(7540), pp.529-533.

[6]Sutton, R.S. and Barto, A.G., 1998. Reinforcement learning: An introduction (Vol. 1, No. 1). Cambridge: MIT press.

[7]Watkins, C.J. and Dayan, P., 1992. Q-learning. Machine learning, 8(3-4), pp.279-292.

[8]Tsitsiklis, J.N. and Van Roy, B., 1997. An analysis of temporal-difference learning with function approximation. IEEE transactions on automatic control, 42(5), pp.674-690.

[9]Dayan, P. and Hinton, G.E., 1993. Feudal reinforcement learning. In Advances in neural information processing systems (pp. 271-271). Morgan Kaufmann Publishers.

[10]Dietterich, T.G., 2000. Hierarchical reinforcement learning with the MAXQ value function decomposition. J. Artif. Intell. Res.(JAIR), 13, pp.227-303.

[11]Sutton, R.S., Precup, D. and Singh, S., 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. Artificial intelligence, 112(1-2), pp.181-211.

[12]Schaul, T., Horgan, D., Gregor, K. and Silver, D., 2015. Universal value function approximators. In Proceedings of the 32nd International Conference on Machine Learning (ICML-15) (pp. 1312-1320).

[13]Mohamed, S. and Rezende, D.J., 2015. Variational information maximisation for intrinsically motivated reinforcement learning. In Advances in neural information processing systems (pp. 2125-2133).

[14]Singh, S.P., Barto, A.G. and Chentanez, N., 2004, December. Intrinsically Motivated Reinforcement Learning. In NIPS (Vol. 17, No. 2, pp. 1281-1288).

[15]Singh, S., Lewis, R.L., Barto, A.G. and Sorg, J., 2010. Intrinsically motivated reinforcement learning: An evolutionary perspective. IEEE Transactions on Autonomous Mental Development, 2(2), pp.70-82.

[16]Stadie, B.C., Levine, S. and Abbeel, P., 2015. Incentivizing exploration in reinforcement learning with deep predictive models. arXiv preprint arXiv:1507.00814.