Universität Hamburg
Department Informatik
Knowledge Technology, WTM

Hierarchical Reinforcement Learning

Seminar Paper

Knowledge Processing

Cuong Nguyen Viet
Matr.Nr. 6756488
5cnguyen@informatik.uni-hamburg.de

07.12.2015

Abstract

The hierarchical structure of behaviour and reward based learning has been long of interest and been considered for the development of intelligent systems using reinforcement learning. In this paper, we gives a brief overview of the classical reinforcement learning concept and a set of approaches to the concept of hierarchical reinforcement learning. A closer look at the components of the hierarchical reinforcement learning suggests how discrete tasks can be divided into sequences of subtasks and simple actions. This leads to new possibilities to solve a wide range of problems. We then look into details of two implementations and discuss the extension of these models to real world problems. Finally, the paper addresses open challenges facing the development of reinforcement learning in a hierarchical setting.

Contents

1 Introduction

Reinforcement Learning (RL) is a branch of learning that concerns with how an agent can achieve the maximum accumulative rewards and avoid punishment. Such learning is inspired by nature processes such as searching for food while exploring new area and avoiding danger at the same time. Each action of the agent at a time will affect the subsequent states of the environment and actions and has a certain reward value associated. The agent learns the behaviour through trials and errors in order to achieve maximum reward. This technique of learning is closely related to the dopaminergic activity in human brain and the operation of neural structures such as the basal ganglia and frontal cortex, which allows human, through the brain's reward, to take action to move towards desired goals.

Reinforcement learning has such a profound and sustained impact, it became well-known in the scientific community. There are lots of research and study on the reinforcement learning algorithms in which an agent can learn behaviour strategy by updating its behaviour from receiving rewards after interacting directly with the external environment. However, like any other, it suffers from a number of factors that limit the applicability of RL. One among these factors is the scaling problem, also known as the "curse of dimensionality". As the task domain grows, the set of possible world states and actions become much larger, thus the memory and computational requirements grow exponentially. This scaling problem has implication on the ability of RL algorithm in complex behavioural context displayed in human and animals.

There are a number of approaches which try to tackle this problem. One of the approaches is to break down the complex reinforcement learning into a hierarchical structure. Complex task can be considered as high level behaviour and broken down into a group of interrelated subtasks. These subtasks can be solved individually and combined together in order to solve the complex task and complete the learning. In this kind of RL, the subtasks is carried out while the high level task is shielded from all the low level actions. This hierarchical structure of RL is referred to as hierarchical reinforcement learning (HRL). For example, a high level task "adding sugar" abstracts over a sequence of low-level steps such as grasping a spoon, using the spoon to scoop sugar, moving the spoon the cup and pouring the sugar.

The hierarchical structure in HRL conforms with the hierarchical structure in goal-directed behaviour and relates closely to the cortical function. Many researches suggest that the framework provided by HRL is similar to and can simulate the role of hierarchical structure in human and animal behaviour. It is also the inspiration for implementation of HRL in neural model network to show how the behavioural structure and HRL are closely related.

The objective in the paper is to consider RL and extension of HRL. We begin, in the following section, by introducing the basis of RL. Then we focus on HRL itself by examining some of the HRL methods like Options (Sutton 1999) [9], a non-neural actor-critic implementation (Botvinick 2009) [2] and a neural implementation of HRL (Rasmussen 2014) [7]. We will look closely at how these methods

address the hierarchical structure of RL tasks and discuss on the feasibility of these methods to solve complex tasks.

2 Background

2.1 Reinforcement Learning

Reinforcement learning (RL) is learning what to do so as to maximize a numerical reward signal [8]. There are two important characteristics of RL, trial-and-error and delayed reward. It is different from other learning is that the learner is not told what action to take but it must try out all possible actions and compute the reward yielding by carrying out the action. The objective of a learner is then to learn the optimal sequence of actions depending on the states in order to maximize the final reward.

RL has four main sub-elements: a policy, a reward function, a value function and optionally, a model of the environment.

A *policy* is how the learning agent behaves at a given time. In general, it is a mapping between the perceived states of the environment to actions to be taken in those states. The policy taking action a when in state s is denoted as $\pi(s, a)$.

A *reward function* defines the numerical value of a perceived state of the environment or a state-action pair. The reward indicates desirability of the state, similar to pleasure and pain in biology system. The reward can be used to decide if the policy should be updated. For example, if the reward follows a given action is low, the policy can change to select some other action.

A *value functions* is the total reward that learning agent an accumulate over future starting from that state. It takes into consideration the immediate reward as well as rewards of the following states.

A *model* of the environment is used to predict the next state and resultant reward given a state and action. The model is closely related to RL methods such as dynamic programming, Monte Carlo or temporal-difference.

RL concerns values for decision making and chooses actions which bring about the highest value so that the greatest amount of reward is obtained in the long run. However, values are hard to determine since they are predictions of values. Maximum values must be estimated and re-estimated from all possible state-action sequence the agent makes. As the result, all RL algorithms try to find an efficient method to estimate values. These algorithms will be introduce later in section.

In general, RL is a computational approach for goal-directed learning and decision making. It emphasises on learning by individual interaction with its environment without any supervision or complete model of the world. Using a formal framework defining the interaction with the environment in terms of states, actions, and rewards, it is able learning for long-term goals.

2.2 Markov Decision Process

Markov Decision Process (MDP) is the principle component in RL. It is a stochastic control model for decision making of a random system. At a discrete time step, the system changes its state by choosing any action available in the current state. The next state is only dependent on its current state and not previous states.

State

A state s is the representation of environment input. It is the information available to agent and represent by values or configuration of the system. The current state of the environment at discrete time step t is denoted as s_t.

Action

Action a is a set of possible actions agent can perform in current state s. The action performed at time t after observing state s is denoted as a_t.

Reward

The reward measures how good the action is in short term after agent performing the action. The reward r_{t+1} is given when the environment transitions to state s_{t+1} after performing action a_t.

Markov property

A stochastic process is said to have the Markov Property if the future states of the environment depends only on the the present state, not on any previous states or actions. Hence the conditional distribution probability of the state at time $t+1$ depends only on the state and action at time t.

$$Pr = \{s_{t+1}, r_{t+1} | s_t, a_t\} \tag{1}$$

Markov decision process

Markov Decision Process (MDP) formally describe an environment for RL where the environment is fully observable. Almost all RL problems can be formalised as MDP.

A particular finite MDP is defined by its state and action sets and by the one-step dynamics of the environment. Given any current state s and action a, the probability of transition from current state s into the next state s_{t+1} after taking action a at a discrete time step t is:

$$\mathcal{P}^a_{ss'} = P_r(s_{t+1} = s' | s_t = s, a_t = a) \tag{2}$$

And the expected value of the next reward is:

$$\mathcal{R}^a_{ss'} = E(r_{t+1} | s_t = s, a_t = a, s_{t+1} = s') \tag{3}$$

Based on these two quantities, the MDP can define a policy for the decision maker $\pi(s, a)$ which fixes an action a for each state s.

2.3 Reinforcement learning model

In RL, the goal of the agent is formalised in terms of rewards and the agent maximises the total amount of rewards it receives. At each time step, the agent interacts with the environment. Depending on the input of the current state s of the environment, the agent chooses an action a. The action changes the state of the environment and the agent is given a reward. The new environment state is then observed.

Return

The agent's goal is to maximize the reward it receives in the long run. This can be formally defined as:

$$R(t) = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{4}$$

where γ is the discount rate and $0 \leq \gamma \leq 1$. The discount rate determines the present value of future rewards. The reward received in k time step in the future is not as valuable as the reward to be received immediately. Hence it is discounted by a factor of γ^{k-1}. It is also used to address the problem of continuous RL task by keeping the total reward finite. In addition, it reflects animal/human behaviour which shows reference for immediate rewards over future rewards.

State value function

The state-value function estimates how good it is for the agent to be in a given state, in terms of future rewards or expected returns. State-value functions are defined with respect to particular policies. The value of a state s under a policy π is formally defined as:

$$V^{\pi}(s) = E_{\pi}\left\{R_t | s_t = s\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s\right\} \tag{5}$$

The optimal state-value function V^* for the optimal policy is defined as:

$$V^*(s) = \max_{\pi} V^{\pi}(s) \tag{6}$$

Action value function

The action-value function or the Q-value function, similarly estimates the expected accumulated return when the agent takes action a under state s and hence follow policy π.

$$Q^{\pi}(s,a) = E_{\pi}\left\{R_t | s_t = s, a_t = a\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s, a_t = a\right\} \tag{7}$$

The optimal action value function Q^* for the optimum action for the current state is defined as:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \tag{8}$$

Bellman equation

In RL, the value functions have the property of a recursive relationship. This property is used through out RL and dynamic programming which is the essence for most RL algorithms. The recursive relationship between the value of a state and the values of its successor states is expressed as:

$$V^{\pi}(s) = E_{\pi}\left\{R_t | s_t = s\right\} = \sum_{a} \pi(s,a) \sum_{s'} \mathcal{P}^{a}_{ss'}[\mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s')] \tag{9}$$

We can obtain the optimal policy after finding the optimal policy function which satisfies the Bellman optimality equations choosing an action a in state s not by following the policy but by choosing the maximum:

$$V^{*}(s) = \max_{a} \sum_{s'} \mathcal{P}^{a}_{ss'}\left[\mathcal{R}^{a}_{ss'} + \gamma V^{*}(s')\right] \tag{10}$$

For the action-value function there is a Bellman equation as well. This best action-value function in the notion of Bellman optimality equation is:

$$Q^{*}(s,a) = \max_{a} \sum_{s'} \mathcal{P}^{a}_{ss'}\left[\mathcal{R}^{a}_{ss'} + \gamma \max_{a} Q^{*}(s',a')\right] \tag{11}$$

The Bellman optimality equation is a set of equations, one for each state s. If the dynamics of the environment $(\mathcal{P}^{a}_{ss'}, \mathcal{R}^{a}_{ss'})$ are known, then it can be solved using any models for solving system of nonlinear equations. Once V^{*} is solved, the optimal policy can be easily determined. Any policy that assign non-zero probability to actions at which the maximum of Bellman optimality equation is obtained is an optimal policy.

Explicitly solving the Bellman optimality equation relies on some assumptions: 1) we know the dynamics of the environment, 2) we have enough computational resources to complete the computation, 3) the Markov property [8]. The second assumption is a great problem as resources is required for complete state-action sets, computing probability of all state-action pair as well as the expected rewards.

2.4 Temporal-difference Learning

The RL methods of *temporal-difference* (TD) learning to predict total amount of reward expected has the combined properties of *dynamic programming* (DP) and *Monte Carlo* (MC). The agent can learn from experience of interaction with the environment without a model for dynamics of the environment. It also updates the estimated value based on other learned estimates. TD methods update the value $V(s_t)$ at time step $t+1$ after visiting nonterminal state s_t.

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \tag{12}$$

TD learning methods have the advantage over DP is that they do not require a model for the dynamics of the environment, of its reward and next-state probability distributions. TD learning methods also converge to the optimal value function faster than MC methods on stochastic tasks.

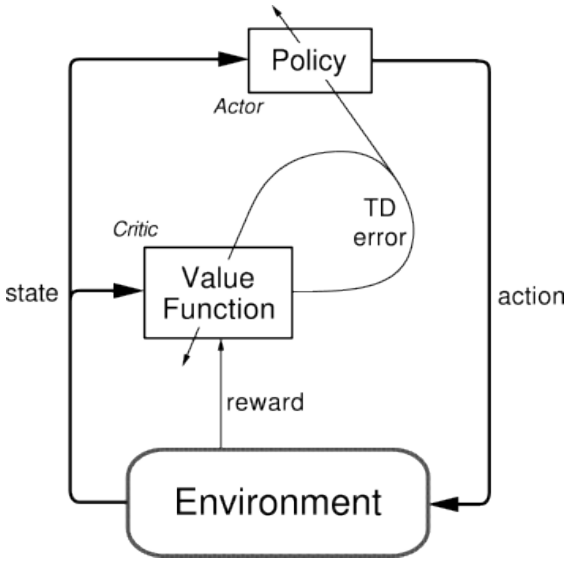

Figure 1: The actor-critic architecture [9]

2.4.1 SARSA

It is an on-policy TD control. The algorithm learns an action-value function and estimates $Q^{\pi}(s, a)$ for policy π and for all states s and actions a. The update for action values after transition of nonterminal state s is as followed:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{13}$$

Since it is an on-policy TD control, the convergence property depend on the nature of policy and Q which in turn depends on the selection methods such as ϵ-greedy or ϵ-soft policies.

2.4.2 Q-Learning

It is an off-policy TD control. If the policy is not known, Q-learning can be used. Q-learning estimates the action-value function Q by selecting action using policy-free selection method and update the action-value function. The Q-learning is defined as:

$$(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right] \tag{14}$$

2.4.3 Actor-Critic

It is an on-policy TD learning with a separate memory structure for policy independent of the value function. As shown in Fig. 1 The policy structure is called *actor* and used for selecting policy-dependent actions. The estimated value function is the *critic* which criticizes the actions made by *actor* in a form of TD error. This TD error is then used to update policy structure in *actor*.

The learning is always on-policy. The *critic* is a state-value function. After each action selection, the *critic* evaluate on the new state to determine if action should be selected. The evaluation or TD error is:

$$\delta = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{15}$$

If TD error is positive, the tendency for choosing action a_t should be strengthened for the future, if TD error is negative, the tendency should be weakened. The strengthening or weakening can be done by:

$$p(s_t, a_t) = p(s_t, a_t) + \beta\delta_t \tag{16}$$

3 Approaches to Hierarchical Reinforcement Learning

3.1 Semi-Markov Decision Process

In MDP, the amount of time between state transition is uniform and equal, action is carried out in discrete time step. Semi-Markov decisions processes (SMDPs) extends MDP by allowing actions that can take multiple time steps to complete. Time becomes a factor in SMDPs framework. Time between state transition can vary and is a random variable. In a *discrete-time* SMDP (Howard, 1971), state transition can only occur at (positive) integer multiples of an underlying time step. Let τ denote the waiting time for state s when action a is executed. The transition probability now take time into consideration as $P(s', \tau|s, a)$. The expected immediate reward now gives the accumulative amount of rewards over the waiting time τ in s given action a.

$$r = \sum_{t=0}^{\tau-1} \gamma^t r(s, a, t) \tag{17}$$

The SDMPs framework is important for HRL, as the time delays can be used to encapsulate the activity of subpolicy. In standard RL, the TD error is estimated immediately in the preceding time step. HRL evaluates at the completion of a subpolicy. The reward cannot be observed immediately but over a sequence of time steps. Hence information needs to be preserved overtime while that subpolicy executes, so that the TD error can be computed. The HRL setting is longer a MDP but rather a SMDPs.

3.2 Options

This section introduces the formalize HRL approach by Sutton(1999) in which activities are notion as options [1].

In the options framework, the concept of temporal abstraction is introduced based on the theory of SMDPs. Options are abstract actions in a SMDPs and each abstract action specifies a partial policy or a subpolicy to be followed. It is defined

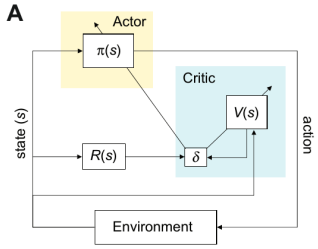

Figure 2: Basic actor-critic architecture [2]

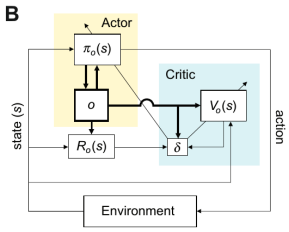

Figure 3: HRL actor-critic [2]

by three components: initiation set of states from which option can be invoked, a termination function is a set of states that will trigger the termination of the action, and the option-specific policy while the option is executing. Primitive actions are defined as one-step options. On any time-step, the agent may select options instead of actions. Once the option is selected, it follows its policy and select actions until the option terminates. Through learning option policies, subsgoals or subtasks can be achieved, where a subgoal is often a state, or a region of the state space, such that reaching that state or region is assumed to facilitate achieving the overall goal of the task.

3.3 Non-neural Actor-critic Implementation of Hierarchical Reinforcement Learning

This section introduces the implementation of HRL by Botvinick(2009)[2] which is based on the options approach by Sutton(1999) [9] but modified to actor-critic.

From Figure 2 and Figure 3 we can see the extension to the basic actor-critic architecture by incorporating temporal abstraction or options in order to obtain hierarchical structure. The basic actor-critic architecture learning is introduce in section 2.4.3. The HRL actor-critic work in the similar way but now maintains a representation of options in control of behaviour. In Figure 3, o: currently controlling option, $R_o(s)$: option dependent reward function, $V_o(s)$: option specific value functions, δ: temporal difference prediction error, $\pi_o(s)$: option specific policies.

Each option is associated with (1) an initiation set, indicating the states where the option could be selected; (2) a termination function, returning the probability of terminating the option in each state; and (3) a set of option-specific strengths W_o, containing one weight for each action (primitive or abstract) at each state [2].

Figure 4 shows the RL task, the agent has to reach the goal state G from the starting state S. In order to reach goal G, the agent must pass through the doorway in any optimal policy. Hence, reaching the doorway can be identified as the *abstract option* or *subgoal* and associated with a *pseudo-reward* (Dietterich, 2000) [3]. The *pseudo-reward* is to be distinguish from the external reward for following the full policy and reaching the final goal state.

For the four-rooms simulations, two options could be initiated in each room,

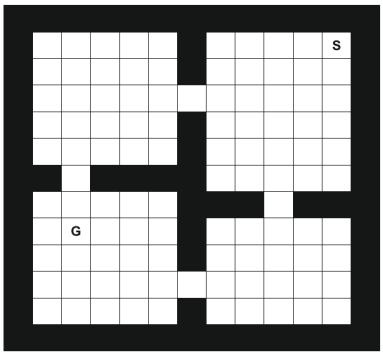

Figure 4: Four rooms problem, Sutton et al.(1999) [2]

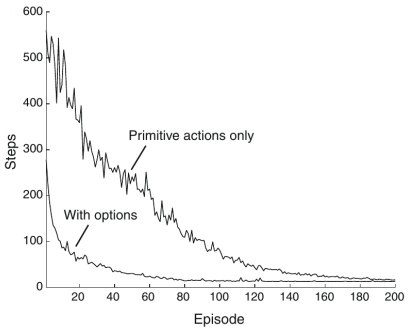

Figure 5: Learning curves [2]

with the initiation set being a particular state in each room (at the corner or middle of the room) and termination set at the room's door. Each option has a pseudo reward at termination state and is used for option-specific value functions. Option-specific policy is learnt through the actor-critic as usual. The agent selects an option. A prediction error is computed after each option-specific action and is used to update option-specific value and action strength of the state preceding the actions. When option reaches termination, a prediction error is computed for the entire option and this is used to update the value and weight of the initiation state of the option.

Botvinick et al.(2009) first trained the model only with the pseudo reward of the subgoal states and then with the reward at the goal state. Figure 5 shows the learning curves of the basic actor-critic RL model and the HRL model. We could observe that with the options included, the model converges much faster to the goal state. Then HRL maybe effective in saving training time on learning.

3.4 Neural Implementation of Hierarchical Reinforcement Learning

Botvinick et el.(2009) discusses how the actor-critic architecture could be extended to support HRL. However, their model itself was not implemented in a neural network.

The HRL architecture implemented by Rasmussen (2014) has layered structure with two systems, the high level systems and the low level systems. Each system has two set of neural populations that carries out RL in SMDPs framework. At the top is the population of neurons that represent the current state s and implementing leaky integrate-and-fire (LIF) neurons. These neurons take input state and convert it to firing activity to the second set of neural populations. The second set of neural populations compute the Q values following the SARSA algorithm. Selection of highest action-value function is done by a *Selection* network which take in the consideration of time-delayed reward in the SMDPs framework. The selected value is delivered to the error calculation network, E. This network com-

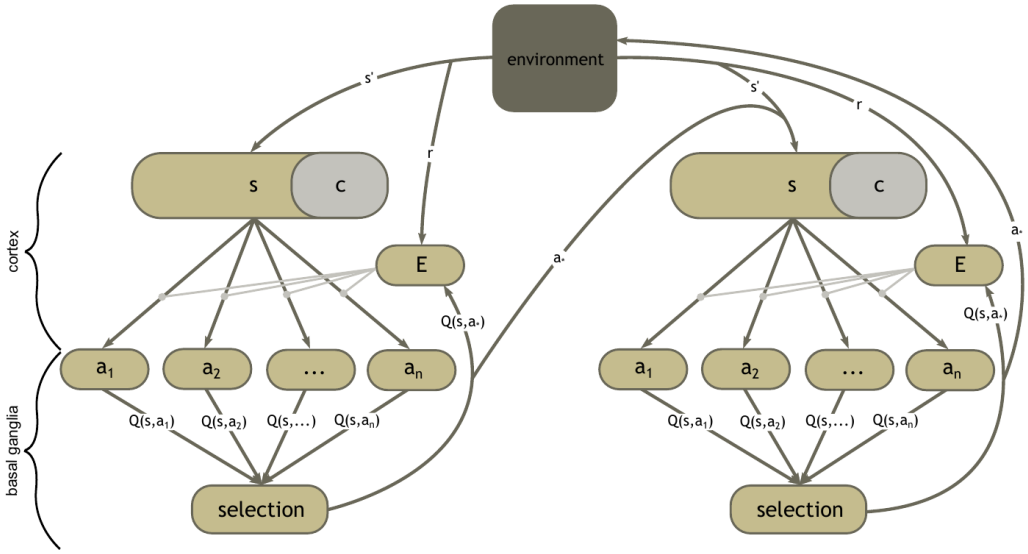

Figure 6: Hierarchical reinforcement learning architecture [7]

putes the prediction error δ and uses this to update the weights to set of actions a in state s.

To introduce hierarchical structure, Rasmussen includes a representation of current context C indicating the different policies. From Figure 6, the high level system on the left can influence the context of the state set in the lower system and hence carry out the policy-specific actions. For e.g. high level system output "go to grocery store" will set context of the lower level system to "grocery store" and the lower system would choose actions according to the "grocery store" policy.

The task used for this model is a delivery task. The agent must go to the package, pick it up and drop it off at a second location. The agent is trained through two steps: the low level system is trained for navigating to the target location in the environment and the high level system is trained to determined the current state and location, pickup or dropoff. The training process is as followed: the agent will determine current state (current location, with or without package) and compute the action-value functions. The action selected will either be "go to pickup location" or "go to drop off location". These will be converted into context input for the lower level system. The agent now learn to navigate to the target location which is depend on the current context through SARSA algorithm with internal reward given whenever it achieves the goal set by the high level system - which is the targeted location.

Figure 8 shows the total reward accumulated of a RL agent and a HRL agent compared to the optimal simulated agent. It can be seen that both RL and HRL agents begin with near similar performance, HRL remarkably improved the performance later on. By the end of training, the total reward accumulated by HRL is 52% of optimal.

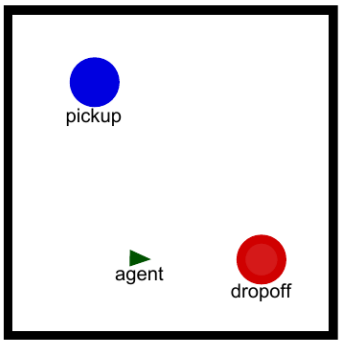

Figure 7: Delivery task [7]

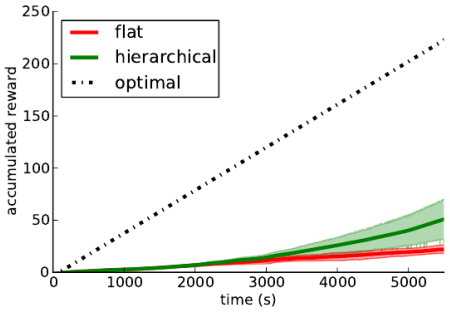

Figure 8: Total reward accumulated [7]

4 Approaches Analysis and Comparison

4.1 Biological Motivation

Both the RL and HRL model are motivated from the biological reward based learning in animal and human. The process is associated with the basal ganglia, frontal cortex and thalamus. These components operate together as parts of a circuit to perform reinforcement learning.

The basal ganglia (BG) and frontal cortex operate in order to execute goal directed behaviours. There are studies suggest that the neural-networks within BG circuits consist of parallel and integrative networks with reciprocal connections and non-reciprocal connections, allowing transfer of information within these connected structures. According to Haber et al.[4], the BG are involved in several aspect of goal-directed behaviour, not only control of movement but also the processes processes that lead to movement, including the elements that drive actions, such as emotions, motivation and cognition. Also ventral regions of the basal ganglia play a key role in reward and reinforcement [4].

The frontal cortex is a massive layer of neural tissue that form the functionally organized pathways in the BG circuit. Its main function is preprocessing information or perception that involve motivation and planning complex actions. It sends input signals through the BG where action selection and reward based learning occur through dopaminergic activity. Functionally defined regions of frontal cortex project outputs through the basal ganglia, to thalamus, and back to cortex as illustrated in Figure 9.

The role of BG focuses on the selection and execution of actions through planning input or learnt behaviour from the cortex. However, Haber (2003) also suggests that BG are critical in reinforcing new behavioural rules and emphasize its importance in reinforcing and adapting past learning through predicting future outcomes. In order for this to happen, Haber suggests that there is communication across functionally distinct circuits.

It is observed that some neural network systems extend beyond connecting adjacent regions. Projections from distinct functional areas in these neural network

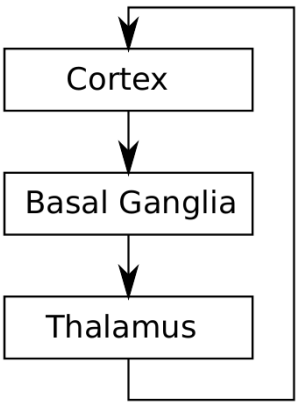

Figure 9: Reinforcement learning circuit

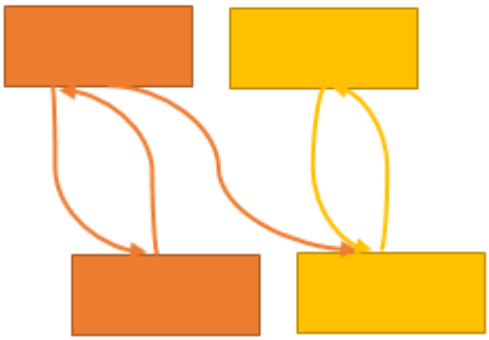

Figure 10: Spiralling projections

systems not only form a close reciprocal loop, they also feed forward to others forming a spiral. The spiral continues through the projections. In this way, certain regions can influence other regions via spiralling projections. This mechanism is illustrated in Figure 10 where projections occurs between distinct functionally regions. In Figure 10 same functional regions is indicated by same colour and communication/projection is indicated by the arrows. Cross functional communication is indicated by the orange arrow to the yellow region.

4.2 Discussion

In previous section 3, we have demonstrated two attempts to implement the HRL architecture in order to perform hierarchical learning, as well as enhancing reinforcement learning ability. Both models use the concept of temporal abstraction and layered architecture for building hierarchical structure of a task and employed reward in SMDPs framework. As the result, both implementations show the advantage of the hierarchical structure in speeding up learning compared to a flat RL model.

In both models, optimal policies can be learned using the subgoal reward function and standard RL methods. The high-level task or abstraction encapsulate the primitive actions and learning occurs in both level using subgoal reward and external reward. However, the hierarchical structure and subgoal reward are predefined and built in as they are in the model. As a result, it is task-specific and limits the agent ability to solve the task. This can be seen from the two tasks described in previous section. For both tasks, the primitive or low level actions are to navigate the agent to targeted location. However, in the four rooms problem, the subgoal is always to reach the doorway first. This gives rise to problem if the optimal path does not required the agent to pass through doorway, hence a direct path to reach goal G from starting state s. In Figure 11, the problem is modified so that there is a "shortcut" opened, allows the agent to reach the goal in only 9 steps instead of 11 steps in the subgoal policy. However, because of the predefined subgoal, the

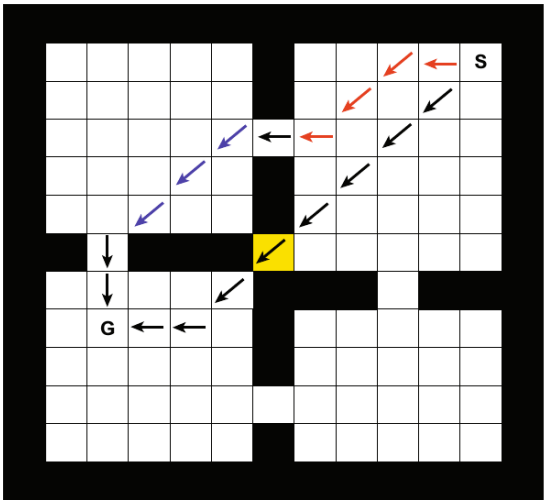

Figure 11: Performance when "shortcut" is opened [2]

agent will completely ignore the optimal new path and hence perform worse than the random flat RL agent. This is the problem of *exploration* when the subgoals are not helpful in reaching the goal state. The agent is stuck in the local minima and always exploring suboptimal policy.

Another constraint that is not only limit to these two approaches but also to other such as HAMs (Parr 1998) [6] and MAXQ (Dietterich 2000) [3] is that having the hierarchical structure built in and not learnt by the agent. The components of the hierarchy and the abstractions used are decided in advance. The question is how to form task hierarchy automatically. One method developed by McGovern and Barto [5] analyses the external reward of action sequence and detect frequently visited states that the agent passes through on successful runs. These states are good candidates to become subgoals. Rasmussen suggests the use of intrinsic motivation to discover rewarding state that can be used for development of subpolicies. This still remains as one of the biggest problem for HRL application especially for tasks with complex problems.

One distinguish advantage of the neural HRL implementation by Rasmussen is the transfer of knowledge. By using context-dependent subtask, the agent is able to transfer knowledge between tasks that overlap - having similar detailed action pattern. For e.g. picking a pencil or a pen have overlapping actions such as the rotation of the hand, the opening and closing action for grabbing, the raising and lowering action etc. These low level actions can be trained first for the context of picking up a pencil and later transfer to new problems such as picking up a pen or object with similar shape. This would be very hard using the options framework to capture overlapping among tasks.

Finally, there is insufficient evidence and data to demonstrate HRL effectiveness in addressing the scaling problem of RL as well as its application with large complex tasks since HRL is still in the early development stage with a small number of studies.

5 Conclusion

The goal of this paper is to familiarize with the reinforcement learning and hierarchical reinforcement learning concept. HRL has been developed to mimic the learning processes inspired by the connections in the basal ganglia and the frontal cortex. The concept of temporal abstractions and high-level, low-level systems enable hierarchical structure for RL tasks and thus proved to be useful in improving effectiveness of RL. Also, implementing HRL models shows much correlation between RL and specific neural structures involved in the reward based learning in human. However there is still limitation of implementing HRL. Further work is required to find out how to build task hierarchies and integrate into system to build it automatically. In addition, the idea of HRL extension to improve RL is promising but much further work is required to certify its effectiveness.

References

[1] Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(1-2):41–77, January 2003.

[2] Matthew M. Botvinick, Yael Niv, and Andrew C. Barto. Hierarchically organized behavior and its neural foundations: A reinforcement learning perspective. *Cognition*, 113(3):262 – 280, 2009. Reinforcement learning and higher cognition.

[3] Thomas G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *J. Artif. Int. Res.*, 13(1):227–303, November 2000.

[4] Suzanne N. Haber. The primate basal ganglia: parallel and integrative networks. *Journal of Chemical Neuroanatomy*, 26(4):317 – 330, 2003. Special Issue on the Human Brain - The Structural Basis for understanding Human Brain function and dysfunction.

[5] Amy McGovern and Andrew G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 361–368, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[6] Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. In *Proceedings of the 1997 Conference on Advances in Neural Information Processing Systems 10*, NIPS '97, pages 1043–1049, Cambridge, MA, USA, 1998. MIT Press.

[7] Daniel Rasmussen and Chris Eliasmith. A neural model of hierarchical reinforcement learning. *Proceedings of the 36th Annual Conference of the Cognitive Science Society*, pages 1252 – 1257, 2014.

[8] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[9] Richard S. Sutton, Doina Precup, and Satinder Singh. Between {MDPs} and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(12):181 – 211, 1999.