Exploring Effects of Intrinsic Motivation in Reinforcement Learning Agents

Nitish Srivastava T Sudhamsh Goutham

April 25, 2010

Abstract

We explore Intrinsic Motivation as a reward framework for learning how to perform complicated tasks. Most reinforcement learning tasks assume the existence of a critic who rewards the agent for its actions. However, taking inspiration for biological agents, we can say that the real critic is the agent itself. We experiment with a model where the rewards are generated by the agent using a process which models the emotion of surprisal. The agent is modeled as intrinsically motivated to look for surprising events. We demonstrate how the agent learns under such a framework. Results show that even in the absence of any external reward, the agent is able to learn skills and perform complicated tasks.

1 Introduction

Motivation is the driving force for us to do anything. Motivation can be distinguished as two types - *extrinsic motivation* and *intrinsic motivation*. Extrinsic motivation means doing something for some reward given by an external critic. Intrinsic motivation motivation, on the other hand, means to do something for the sake of inherent happiness it gives. Intrinsic motivation is what causes people to perform activities like play, explore and activities which are driven by curiosity. These activities help in building broad competence rather than acting as steps in an externally directed goal. These competence levels are not acquired by specific problems but are helpful in solving the problems that arise specificially in an agent's lifetime. The skills acquired are used as building blocks which are used to form new solutions to problems as they arise. That is, instead of trying to form a solution from low level primitives step-by-step, it can form a solution by combining these high level skills. This increases the efficiency of problem solving in animals and we wish to see if such efficiency can be achieved with machines.

We use *Reinforcement Learning(RL)* method to implement an agent illustrating the learning of these broad competence levels. We implement the experiment as given in [1]. We implement a Playroom Environment and see how the agent learns to do different things in the playroom on its own in the absence

of an external reward, simply driven by its curiosity. The inspiration for the reward method used in this implementation come from neuroscience. The neuromodulator dopamine plays a major role in the extrinsic motivational control of behaviors aimed at explicit rewards and also in the intrinsic motivational control of behaviors associated with novelty and exploration. For example, salient, novel sensory stimuli inspire the same sort of phasic activity of dopamine cells as unpredicted rewards. However, this activation extinguishes more or less quickly as the stimuli become familiar. This may underlie the fact that novelty itself has rewarding characteristics. This fact helps us in determining the reward mechanism.

2 Reinforcement Learning

Reinforcement Learning is a sub-area of *machine learning* dealing with how an *agent* should take *actions* in an *environment* so as to maximize the long-term *reward*. RL algorithms try to find a *policy* that maps *states* of the world to the actions the agent should take in those states. The environment is typically formulated as a Markov Decision Process(MDP) and RL algorithms in this context are like dynamic programming techniques.

Reinforcement Learning is different from other machine learning techniques like supervised learning in that there is no learning set given with correct input output pairs. Formally, the RL problem modeled as an MDP consists of

1. a set of states S, in which the environment can be in,
2. a set of actions A, and
3. a set of rewards $r \in \Re$.

At each time t, the agent is in state s_t and has a set of actions possible in that state $A(s_t)$. From that set it chooses an action $a \in A(s_t)$ to perform and moves to a new state s_{t+1}. The environment gives a reward r_t associated with this transition. The goal of RL algorithm is to find a policy $\pi : \ S \times T \longrightarrow A$ (where T is the set of maximum possible time indices) which maximizes the reward $R \ = \ r_0 + r_1 + ... + r_n$.

According to standard view of RL, the the agent tries to change the environment and receives a reward from the critic (who is in the environment) in the process. The agent learns to improve its skill by trying to maximize the reward obtained from the critic. It should be noted that the critic is not outside the agent, in the sense that, the external environment sends proper signals to the internal environment inside the agent which has critic in it converting those signals into rewards[3]. Fig 1. shows a simple view of RL and also an elaborated view of the environment.

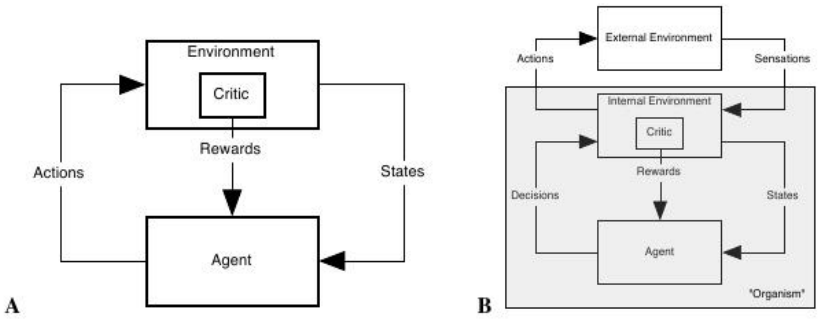

Figure 1: Agent-Environment view in RL. *A*: A simple view. *B*: An elaborated view. (Images adapted from [1])

In usual implementation of RL algorithms, the problem the agent needs to learn is formulated and a proper reward function associated with the problem is defined. But here, in intrinsic motivation, we need to have a reward function independent of the task at hand.That is where we depart from the usual path taken by RL methods. We will look at Semi-Markov Decision Processes (SMDPs) and Option Models before further moving on to define the skills and reward functions.

3 Options and SMDPs

Human decision making involves choice among temporally extended courses of action over a broad range of time scales. Hence temporal abstarction in AI is very important. [2] brings temporal abstarction into the framework of reinforcement learning techniques using MDPs. The conventional MDPs do not involve temporally extended actions. All the actions in MDPs start at t and end at $t + 1$. There is no notion of an action persisting over a period of time. A minimal extension to MDPs to use temporal abstarction is to build on the concept of Semi-Markov Decision Processes (SMDPs).

SMDPs are a special kind of MDPs designed to model continuous-time discrete events. The actions in SMDPs take variable amount of time and are intended for modeling temporal courses of action. SMDP planning ia also well known and is similar to MDP planning. However the the conventional model of SMDPs has a disadavantage in that the extended courses of actions are treated as indivisible and unknown units. [**?**] tries to address this issue by looking at conventional MDPs where there are some courses of actions which are variable time bound. The term *options* is used to describe these extended courses of action. In fact, the single step actions can be seen as a special case of options where the time taken is unity. As described in Fig 2., a set of options defines an

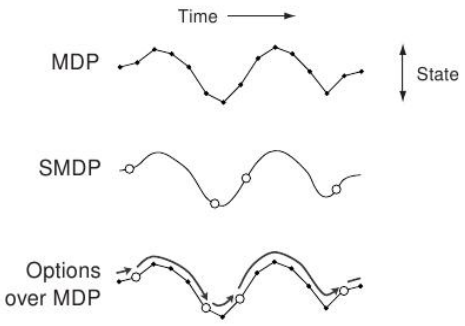

Figure 2: The state trajectory of MDPs consists of small, discrete time transitions, whereas that of SMDP consists of large, continuous time transitions. Options allow MDP trajectory be interpreted in both ways. Image adapted from [2])

SMDP embedded within an MDP. It can be seen that by using options, MDPs can be interpreted in both ways, as single step transitions and as an SMDP over options. Now [**?**] extends the usual RL framework over MDPs to MDPs with options.

3.1 Bellman Equations for MDPs

We use the following notation in this section.

1. Set of states S
2. A state at time t, $s_t \in S$
3. Set of Actions possible in state s_t, A_{s_t}
4. Action taken at time t, $a_t \in A_{s_t}$
5. Set of Actions $A = \cup_{s \in S} A_s$

Now if S and A are finite, the environment can be modeled as the one-step state transition probabilities

$$p^a_{ss'} = Prs_{t+1} = s' | s_t = s, a_t = a \tag{1}$$

and the one-step reward functions

$$r^a_s = Er_{t+1} | s_t = s, a_t = a \tag{2}$$

for all $s, s' \in S$ and $a \in A$. These two quantitites constitute the *one-step model* of the environment.

The agent objective is to learn an optimal Markov policy $\pi : S \times A \longrightarrow [0, 1]$, which maximizes a discounted long-term reward from each state s

$$V^{\pi}(s) = E[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ...|s_t = s, \pi] \tag{3}$$

where γ is a dicount factor $\in [0, 1]$. The RHS of the equation gives the summation of all discounted future rewards received by following the policy π when the current state is s_t. The above equation can be rewritten as

$$V^{\pi}(s) = \sum_{a \in A_s} \pi(s, a)[r_s^a + \gamma \sum_{s'} p_{ss'}^a V^{\pi}(s')] \tag{4}$$

where $\pi(s, a)$ is the probability with which policy π chooses action a in state s. V^{π} is called the *state-value function* for policy π. Similarly for an *optimal* policy the state value function would be

$$V^*(s) = \max_{a \in A_s}[r_s^a + \gamma \sum_{s'} p_{ss'}^a V^{\pi}(s')] \tag{5}$$

So given a V^*, an optimal policy can be found by choosing any action in s that maximizes equation 5. Equations 4 and 5 which recursively relate themselves are called Bellman Equations.

More important for learning are the Bellman Equations for *action-value functions*. The value of taking action a in state s and following π henceforth, denoted by $Q^{\pi}(s, a)$ is given by

$$Q^{\pi}(s, a) = E[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ...|s_t = s, a_t = a, \pi] \tag{6}$$

which can be rewritten as

$$Q^{\pi}(s, a) = r_s^a + \gamma \sum_{s'} p_{ss'}^a \sum_{a'} \pi(s', a') Q^{\pi}(s', a') \tag{7}$$

The RHS of the equation 7 means the reward of taking action a in state s plus the discounted future reward given by summing up all possible transitions from state s to any other state s' by taking action a and then following π from state s'. A similar *optimal* action value function would be

$$Q^*(s, a) = r_s^a + \gamma \sum_{s'} p_{ss'}^a \max_{a'} Q^*(s', a') \tag{8}$$

3.2 Epsilon-Greedy Strategy

The above update equations ensure that the agent behaves optimally once it has learnt the action-value values. However, while the agent is learning, it should not follow very strictly its half-learned policy but explore the state space, even if it means a departure from from what is optimal *at that time*. This intuition is captured by an Epsilon-Greedy action selection method. It is used to set

the exploitation-exploration tradeoff. We use this method to allow the agent to explore its state-action space better before trying to exploit its learned strategy for generating reward. In this method the optimal action choice at any given time is taken with probability $1-\epsilon$ and a random action is taken with probability ϵ.

4 Experiment Design

The experimental setup is the same as used in Singh et al. [1]. We choose this setup to facilitate comparison of results. Also the nature of the setup is extremely relevant to the task at hand.

The environment consists of a Playroom. The room is a 4×4 grid. The room has a few objects (toys) which the agent can play with. These are: a light switch, a ball, a bell, movable buttons for turning music on and off and a monkey. The agent consists of an eye, a hand and a marker. These can be situated in different grid cells of the room. The agent is capable of moving any of these three entities to the other two. In addition it can move its eye to a random object or to any one adjacent grid cell. Also if it has both its hand and eye on the same object, it can use the toy to perform the special action associted with it. Each object has an associated special action. The light switch can be used turn lights on or off. The ball when kicked moves in a straight line to the marker. The bell is rung when a ball is kicked into it. There are separate colored buttons for turning music on and off respectively. These colors become manifest only when the light is on. The monkey is the most complicate dobject to engage. It makes a noise if the light is off, music is on and the bell is rung by kicking the ball into it. Thus different levels of difficulty are associated with engaging different objects. This structure makes it possible to observe the learning of options for performing sub-tasks that lead to the final goal. Rewards are associated with salient events. A salient event is one which involves change in primary sensory input such as change in light and sound intensity. These are considered hard-wired into the agent (possible by an evolutionary process).

The experiments consist of episodes of fixed length (200). The positions of objects and the agent are reset at the end of each episode. The learning models are retained. We experiment with different complexity levels in the Playroom. We decrease the complexity by constraining the number of objects, or the capabilities of the agent.

4.1 Simple Q-Learning

We first experiment with the simple task of Q-Learning the policy to turn lights on and off when no other objects are present in the room. A constant reward is generated each time the light is switched on. Figure 4.1 shows the result of our experiment. As the number of episodes for which the agent is run increases, the amount of reward collected increases vary rapidly till it attains a high constant value, indicating that the optimal policy has been learnt. The policy corresponds

Figure 3: The Playroom environment (Image adapted from [1])

to switching lights on and off in succession. This is reminiscent of the problem, where the simple model of giving an extrinsic reward for achieving a desirable action does not really lead to the best policy in terms of what the intention of the reward signal is. In this case if the intention was to make the agent learn how to switch on the light, the optimal policy doesn't really inforce that the agent should stop doing the action once the goal is reached.

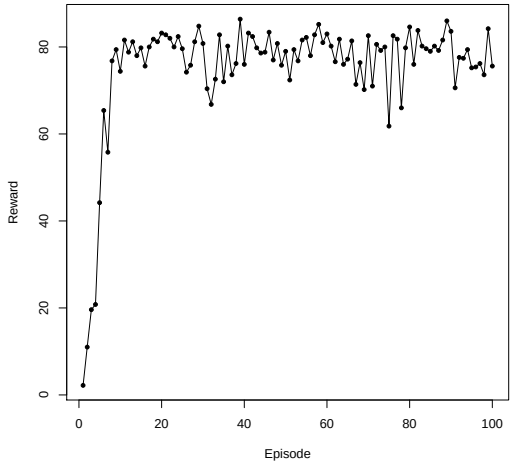

Figure 4: Simple Q-Learning for turning lights on. A constant reward is given for turning lights on.

4.2 Option learning for single option

In this experiment we replace the simple Q-Learning algorithm of the previous experiment with an SMDP learning algorithm for learning options. The reward is now inversely proportional to the probability of predicting the salient event. This means that once the option model for the option has been learnt and the agent can predict the outcome of its action when operating under that option, the amount of reward generated goes down.

$$r_t^i = \tau \left(1 - P(s_{t+1}|s_t, o)\right)$$

Figure 4.2 shows the results for our experiments. Agent learns to switch light on/off quickly as indicated by the high peak in the reward plot. The rewards then fall off because the agent can predict the salient event. But in the absence of any other event, it does not need to use that option any more. Hence the number of times the salient event is reached also goes down as seen in the plot on the right.

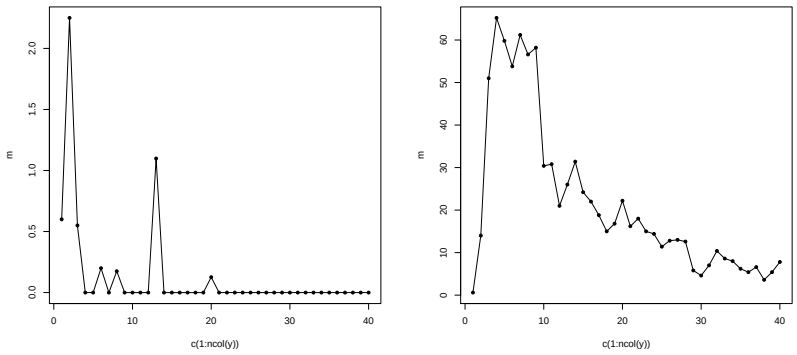

Figure 5: SMDP Q-Learning for turning lights on with single option. The figure shows a plot of a) reward and b) number of hits to the salient event with number of episodes. The intrinsic reward given for turning lights on depends inversely on the probability of correctly predictig the result of the action which leads to a salient state. As the agent learns the option, this probablity increases and hence the reward decreases. In the absence of any other object in the room, the number of hits falls off.

4.3 Option learning for more than 1 option

Next, we add more objects : the music on and off buttons. Now, instead of just 1 salient event being possible, another event that causes change in sound intensity is possible. The agent learns with an SMDP Q-Learning algorithm with intra-option learning. Figure 4.3 shows the results for the salient event

LightOn. Here, though the reward falls off, the number of times the salient event occurs does not fall, showing that the agent is able to use the option model for turing the lights on for turing music on. Even though the agent does not get any reward for turning the lights on, it infers that using that option, turning music on is faster to achieve. This demonstrates that the agent learns to use options for performing more complicated tasks.

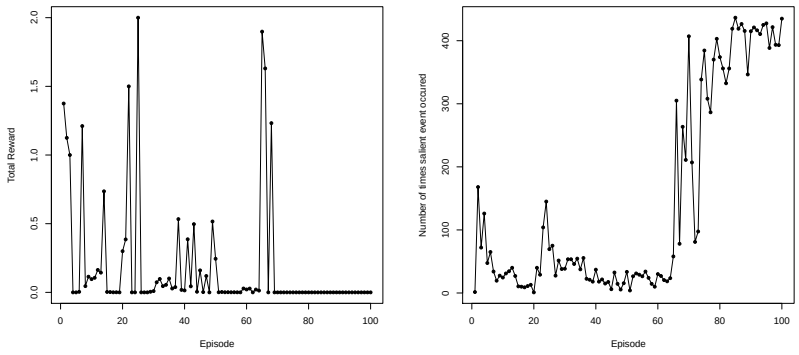

Figure 6: SMDP Q-Learning for turning lights on with multiple salient events.The figure shows a plot of a) reward and b) number of hits to the salient event with number of episodes. The intrinsic reward given for turning lights on depends inversely on the probability of correctly predicting the result of the action which leads to a salient state. As the agent learns the option, this probablity increases and hence the reward decreases, but since other events require the execution of the LightsOn option, the number of hits does not fall, even though the reward is small.

5 Conclusion

Results show that the agent can learn simple tasks very efficiently using Q-Learning. The rewards generated for a salient event fall off gradually as the agent is able to predict it better. As the complexity of the domain is increased and multi-option models are available to the agent, it is able to use them for performing more complicated tasks.

6 Acknowledgments

We extend our sincere gratitude to Dr Amitabha Mukerjee for supervising the project. We used the RL-Glue framework [4] for implementing the agent, environment and the experiment.

References

[1] Satinder P. Singh, Andrew G. Barto, and Nuttapong Chentanez. Intrinsically motivated reinforcement learning. In *NIPS*, 2004.

[2] Richard Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

[3] Richard S. Sutton and Andrew G. Barto. Reinforcement learning : An introduction, 1998.

[4] Brian Tanner and Adam White. RL-Glue : Language-independent software for reinforcement-learning experiments. *Journal of Machine Learning Research*, 10:2133–2136, September 2009.