Autonomous Learning Challenge

Introduction

Autonomous learning requires that a system learns without prior knowledge, prespecified rules of behavior, or built-in internal system values. The system learns by interacting with the environment and uses occasional reward signals to determine which actions are the most rewarding. Until recently, such systems could only use hierarchical reinforcement learning or its derivatives with added artificial curiosity. While such systems can learn efficiently in the environment that does not change its rules as a result of the agent's action, they are not efficient in the environments where as a result of the agent's action the environment changes either making it easier or more difficult for the agent to accomplish his goals.

If the environment changes dynamically as a result of the agent's action, the agent should be able to observe such a change and learn from it. For instance, if the agent is rewarded for heating up his house, he may want to pick up wood to burn it. But as a result of his action there is less and less wood and it takes longer to find it in order to maintain proper house temperature. But the agent may discover that when he is out of the wood he can buy a full load of the wood from sawmill. The agent does not get a reward for this action, yet it helps him to modify the environment to his advantage. Since he now needs money to buy the wood, he may learn that working in the factory may give him money he needs. Notice, that the agent not only learns proper actions, but introduces new goals (buy wood, get money) for which he is not directly rewarded.

The Black Box Scenario Challenge

We challenge you to test your reinforcement learning algorithm and see if your agent can "survive" in a dynamic environment represented in this black box scenario. If your agent can average a reward higher than 0.8 for 10000 iterations you may be a winner in reinforcement learning category. Please send us your results. We will list results for the best performing algorithms.

Black Box Environment Scenario:

What is the Black Box environment?

The black box is a simulation environment designed to compare different Reinforcement Learning algorithms. The software code of this black box scenario is in the attached link.

Black Box software

Why use it?

It allows us to use a standardized testing environment from which to gather data and evaluate results in a normalized format. By making it available to everyone we can further expand our knowledge of machine learning algorithms.

Description:

The black box environment operates by configuring a simple environment for an agent to operate within. The agent receives simple state information from the environment and a reward signal (or signals in the case of multiple primitives), and uses that information to generate an action.

The agent should not have any a-priori information about the environment (other than the state array characteristics/size and number of possible actions). The agent only receives information about the sensory input (vector generated by environment values) and the reward (a scalar value). The response from the agent is single output activation (much like one hot encoding).

For the current incarnation of the experiment, we have set it up so there is only a single "primitive" reward input. Any intrinsic reward will have to be determined entirely by the agent, based on the responses from the environment.

The Scenario:

The scenario is a generic one. Each 'level' consists of a sensory-motor pair that resolves a potential need. In the described configuration, actions range in value from 1-64 (or levels^2). A value of '0' indicates 'Do Nothing'. How you translate to and from the action value is up to you, so long as the environment receives values in the aforementioned range.

The agent will receive a reward value as generated by the environment, and the state of the resources in the environment (in terms of their quantities). The RL agent should then process this input and determine what action to take in response.

Ideally, the agent should be able to learn to manage the environment and maximize its reward.

The reward signal is computed by the environment and is available to the agent after each iteration. Our primary method of evaluating results is via the averaged reward signal. This signal is normalized and is displayed after completing the simulation.

Using the code:

In particular there will be three things you will have to do to use the code (line numbers refer to the main.m file):

1. Initialize your agent. Example code using a basic Q-Learning mechanism is present on lines 49-58
2. Call the agent
   Line 74 performs the call by sending the environment state, reward, and the other information to the RL code.
   persit_data is used to hold information needed for the RLfunc to operate (such as the Q-table in our example of RLfunc.m). You can organize this data in any way that supports your RL agent. persit_data is updated from iteration to iteration within RLfunc.m.
3. Set the user controlled parameters on lines 18-21. (Line 22 is for your convenience, since it specifies the number of actions that are possible in the environment.)
4. Run the program. Email us RLresults_**Xp0**.mat and a reference to your learning algorithm (where **Xp0** is the rate setting you used).

Remember that this is an example and you may need to alter the main file to use your code, however, the agent should still receive only the state information and reward (and any persistent information) and return only an action. Granted, you may store whatever else you need in whatever structure you use to whole persistent information.

User parameter definitions:

- env_params.niter – Specifies how many iterations to run. We usually use around 10,000 to ensure the simulation has plenty of time to equilibrate.
- env_params.inf_toplevel – Specifies whether the highest level resource is infinite. Having this set to true, makes it theoretically possible for an agent to recover to optimal state even if it depletes all other resources. (Many of the algorithms we have tested would perform worse if not for having this set.)
- env_params.nbatch – Specifics the number of consecutive runs to execute and average together to generate the results (use at least 20 for good statistics).
- env_const.primrate – Specifies the rate at which the primitive resource is depleted. Has significant effect on the level of pressure the agent has to operate under.

The example function (RLfunc.m):

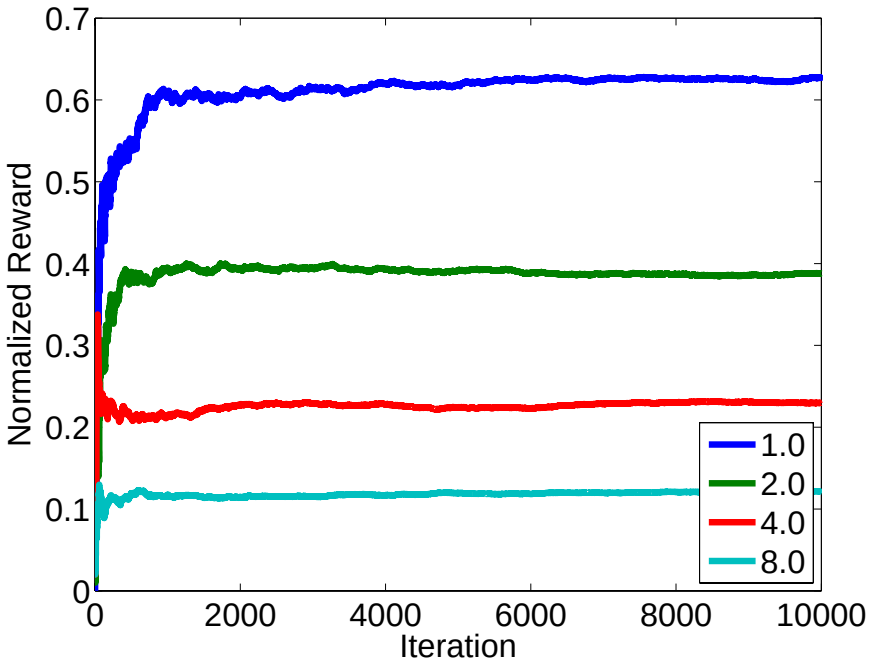
A simple example function is provided in RLfunc.m for you to see how Q-learning works via a lookup table containing a list of successful state-action pairs and the rewards they obtained. Thus, if an action provides a reward the state-action pair can be added to the table. If the known state occurs in the future, the agent can take the proper action (95% of the time – 5% of the time it will take a random action). Note that persit_data structure is used to maintain the Q-table and other needed information between iterations.

Interpreting the Results:

As stated, the performance of an algorithm is evaluated based on the rewards it obtains.

In the experiment, we vary S_{Rate}, which impacts the rate of change in the environment, or more directly, the pressure under which the agent has to operate. Hence a higher value for S_{Rate} will tend to lead to lower performance. For instance, if $S_{inp} = 40$ and $S_{Rate} = 1$, it will take 40 iterations for the resource to deplete (and make the maximum reward available). Higher values of S_{Rate} will mean it will take less time for the resource to be depleted. In the plot above, you can see the effect of varying S_{Rate} from 1 to 8. Try to run your code for various rates from 0.5 to 10 to see changes in the resulting average rewards. If an agent maximizes the total reward available the normalized average reward will approach (or equal) 1.0 during that time.

Results for the basic Q-Learning algorithm for four different rates are shown below.



The Dynamic Environment Challenge

This challenge is to invite your contribution to compare your method against motivated learning method. If you can train the agent to learn and correctly operate in the dynamically changing environment we describe here, let us know by sending an email to daniel.jachyra[at]gmail.com or starzykj[at]gmail.com.

If you think that your program (possibly with subgoals) can handle this situation we will be glad to exchange with you our data and results as well as acknowledge your contribution.

The Scenario

An agent working in the environment is capable of performing several actions (like eat, buy, kick, work, study, etc.). Each action can be performed on any object (subject) that is

in the environment, so for instance the agent can buy food, eat it, kick it etc. Only some actions are beneficial to the agent (either directly or indirectly), and typically cause a change in the environment. For instance if the agent works in the factory, he can get money for his work, so money is available to him. Most of the actions will use some resources (for instance if the agent buys food he uses money), therefore changes in the environment may increase one resource, while decreasing another one. The agent may also work on another subject (for instance punch it) in order to change its behavior (for instance if the subject steals food from the agent).

Thus the agent responds to both changes of resources and other subject's actions.

A simple example scenario is illustrated below. The table illustrates useful motor actions that result at beneficial changes in the environment. The agent is rewarded for eating food. The food is disappearing when the agent eats it, however it can be restored, when the agent fills in the bowl with food from the bucket. When the food in the bucket is gone, the agent may buy the food using money. When the money is gone, the agent may get more money by working with the hammer. Hammers are also used up when the agent works and he can get a new job (with new hammers) after studying for it. The agent gets tired of studying, so he can regain his energy to study when he plays for joy with a beach ball.

We assume that the beach ball is always available to the agent.

List of Resources, useful Resource-Motor pairs and their outcome

Motor action	Resource name	Agent's pains	Outcome		
			Increase	Decrease	Pain reduce
Eat food from	Bowl	Lack of food in Bowl		Food in Bowl	Hunger
Take food from	Bucket	Lack of food in Bucket	Food in Bowl	Food in Bucket	Lack of food in Bowl
Buy food with	Money	Lack of Money	Food in Bucket	Money	Lack of food in Stockpile
Work for money with	Hammer	Lack of Job	Money	Hammer	Lack of Money
Study for job with	Book	Lack of School	Hammer	Book	Lack of Job
Play for joy with	Beach ball	Lack of Joy	Book	Beach ball	Lack of School
		Hunger – primitive pain			
		Curiosity – primitive pain			

Actions other than those listed in the table produce no useful effect.

Each time the agent performs an action (useful or otherwise) he uses one unit of a resource that he needs to perform the action. For instance if he plays with the hammer he gets nothing but uses up one hammer.

Initially the environment offers the agent 10 bowls of food to eat. The agent gradually gets hungry and must eat, for which he gets a reward. The agent gets hungry every 3 cycles, where a cycle is a completion of one task (for instance working with a hammer).

Once all 10 bowls of food are gone, the environment gives the agent 10 buckets of food. The agent can get 3 balls of food from each bucket, providing that he will do this action (take the food from the bucket). However, each action on the bucket (right or wrong) will use one bucket, so for instance if instead of taking food from it, the agent will kick the bucket, one bucket is gone and he gets nothing.

After all 10 buckets are gone the environment gives the agent 10 dollars – each dollar if properly used can buy 3 buckets of food.

After all money is gone the environment gives the agent 10 hammers. If the agent works with a hammer he can get 3 dollars.

After all hammers are gone the environment gives the agent 10 books. If the agent studies a book he can get 3 hammers.

After all books are gone the environment gives the agent beach balls. If the agent plays with a beach ball he can get 3 books. Number of beach balls is unlimited.

The performance in this challenge is measured by how often the agent eats (how many cycles between eating food).

Notice that the environment gives consecutive resources to the agent only once and except for the beach ball they are limited in number and require proper action from the agent to renew such resource.

This scenario is illustrated by a simulation shown on a youtube video accessible from the link on page: http://www.ncn.wsiz.rzeszow.pl/index.php/tools-software